STRIVER A2Z DSA SHEET

Ultimate Interview Preparation Guide — LeetCode Dark Theme

Better & Optimal Solutions in C++ | Exactly 1 Question Per Page

Topic Coverage (Starting from Sorting Techniques):

- **Sorting:** Selection, Bubble, Insertion, Merge, Quick Sort, Recursive Sorts
- **Arrays:** Easy, Medium, Hard (Kadane's, Two Sum, Dutch Flag, Boyer-Moore, Inversions)
- **Binary Search:** 1D Arrays, 2D Matrices, Search Space Reduction
- **Strings:** Anagrams, Isomorphic, Palindromes, KMP/Z-function
- **Linked Lists:** Singly, Doubly, Fast-Slow Pointers, Reversal, Flattening
- **Recursion & Backtracking:** Subsets, Combinations, N-Queens, Sudoku
- **Bit Manipulation:** Power Sets, XOR Operations, Single Numbers
- **Stacks & Queues:** Monotonic Stack, Next Greater/Smaller, Sliding Window Max
- **Sliding Window & 2-Pointers:** Longest Substring, Fruit Baskets, Subarrays
- **Heaps / Priority Queues:** Kth Elements, Median from Stream, Merging
- **Greedy Algorithms:** N Meetings, Job Sequencing, Jump Game, Non-overlapping
- **Binary Trees & BSTs:** Traversals, Diameter, LCA, Views, Serialization
- **Graphs:** BFS, DFS, Dijkstra, Bellman-Ford, Floyd-Warshall, Prim's, Kruskal's, Topo
- **Dynamic Programming:** 1D, 2D, Grid, Subsequences, MCM, Stocks, LIS
- **Tries:** Prefix Search, Autocomplete, Max XOR

Layout Rule: All Brute Force skipped. Every question has its problem statement and complete syntax-highlighted solution self-contained on its own single page.

Q1. Selection Sort

EASY

Topic: Learn Important Sorting Techniques > Sorting-I

Practice Link

Example 1:
Input: N = 6, array[] = {13,46,24,52,20,9}
Output: 9,13,20,24,46,52
Explanation: After sorting the array is: 9, 13, 20, 24, 46, 52

Example 2:
Input: N=5, array[] = {5,4,3,2,1}
Output: 1,2,3,4,5
Explanation: After sorting the array is: 1, 2, 3, 4, 5

[OPTIMAL SOLUTION] Approach — Time Complexity:O(N^2),Selection sort runs in O(N²) time in the best, average, and worst cases due to its nest

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to perform selection sort
void selection_sort(int arr[], int n) {
    // Traverse the array
    for (int i = 0; i < n - 1; i++) {
        int mini = i; // Assume current index holds the minimum value

        // Find the index of the smallest element in the remaining array
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[mini]) {
                mini = j; // Update mini if smaller element is found
            }
        }

        // Swap the found minimum element with the first element of unsorted
          part
        int temp = arr[mini];
        arr[mini] = arr[i];
        arr[i] = temp;
    }

    // Print the sorted array
    cout << "After selection sort: " << "\n";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << "\n";
}
```

Q2. Bubble Sort

EASY

Topic: Learn Important Sorting Techniques > Sorting-I

Practice Link

Example 1:
Input: N = 5, array[] = {5,4,3,2,1}
Output: 1,2,3,4,5
Explanation: After sorting we get 1,2,3,4,5

Example 2:
Input: N = 6, array[] = {13,46,24,52,20,9}
Output: 9,13,20,24,46,52
Explanation: After sorting we get 9,13,20,24,46,52

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity:O(N2) for the worst and average cases and O(N) for the best case. Here, N = size of the array.

```cpp
#include <bits/stdc++.h>
using namespace std;

class BubbleSort {
public:
    void bubble_sort(vector<int>& arr) {
        int n = arr.size();
        for (int i = n - 1; i >= 0; i--) {
            int didSwap = 0; //Use two nested loops to iterate over the array
            for (int j = 0; j <= i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    swap(arr[j], arr[j + 1]);  //Swap arr[j+1] with arr[i]
                    didSwap = 1;
                }
            }
            if (didSwap == 0) {
                break;
            }
        }

        cout << "After Using Bubble Sort:\n";
        for (int num : arr) {
            cout << num << " ";
        }
        cout << endl;
    }
};
```

Q3. Insertion Sorting

EASY

Topic: Learn Important Sorting Techniques > Sorting-I

Practice Link

Example 1:
Input: nums = [7, 4, 1, 5, 3]
Output: [1, 3, 4, 5, 7]
Explanation: The array is sorted in non-decreasing order: $1 \leq 3 \leq 4 \leq 5 \leq 7$.

Example 2:
Input: nums = [5, 4, 4, 1, 1]
Output: [1, 1, 4, 4, 5]
Explanation: The array is sorted in non-decreasing order: $1 \leq 1 \leq 4 \leq 4 \leq 5$.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n^2), where n is the number of elements in the array. This is because, in the worst case, w

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to sort the array using insertion sort
    vector<int> insertionSort(vector<int>& nums) {
        int n = nums.size(); // Size of the array

        // For every element in the array
        for (int i = 1; i < n; i++) {
            int key = nums[i]; // Current element as key
            int j = i - 1;

            // Shift elements that are greater than key by one position
            while (j >= 0 && nums[j] > key) {
                nums[j + 1] = nums[j];
                j--;
            }

            nums[j + 1] = key; // Insert key at correct position
        }

        return nums;
    }
};
```

Q4. Merge Sorting

MEDIUM

Topic: Learn Important Sorting Techniques > Sorting-II

Practice Link

Input : N=7,arr[]={3,2,8,5,1,4,23}
Output : {1,2,3,4,5,8,23}
Explanation : Given array is sorted in non-decreasing order.

Input : N=5, arr[]={4,2,1,6,7}
Output : {1,2,4,6,7}
Explanation : Given array is sorted in non-decreasing order.

[OPTIMAL SOLUTION] General — Time Complexity: O(N*logN), merging two arrays take linear time and array is recursively divided into halves (

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to merge two halves of the
      array
    void merge(vector<int>& arr, int low, int
      mid, int high) {
        // Create temp arrays
        vector<int> temp;
        int left = low, right = mid + 1;

        // Merge two sorted halves
        while (left <= mid && right <= high) {
            if (arr[left] <= arr[right])
                temp.push_back(arr[left++]);
            else
                temp.push_back(arr[right++]);
        }

        // Copy remaining elements from left
          half
        while (left <= mid)
            temp.push_back(arr[left++]);

        // Copy remaining elements from right
          half
        while (right <= high)
            temp.push_back(arr[right++]);

        // Copy sorted elements back to
          original array
        for (int i = low; i <= high; i++)
            arr[i] = temp[i - low];
    }

    // Recursive merge sort function
    void mergeSort(vector<int>& arr, int low,
      int high) {
        if (low >= high)
            return;

        // Find the middle index
        int mid = (low + high) / 2;

        // Recursively sort left half
        mergeSort(arr, low, mid);

        // Recursively sort right half
        mergeSort(arr, mid + 1, high);

        // Merge the two sorted halves
        merge(arr, low, mid, high);
    }
};
```

Q5. Recursive Bubble Sort

EASY

Topic: Learn Important Sorting Techniques > Sorting-II

Practice Link

Example 1:
Input: N = 6, array[] = {13,46,24,52,20,9}
Output: 9,13,20,24,46,52
Explanation: After sorting we get 9,13,20,24,46,52

Example 2:
Input: N = 5, array[] = {5,4,3,2,1}
Output: 1,2,3,4,5
Explanation: After sorting we get 1,2,3,4,5

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N2) for the worst and average cases and O(N) for the best case. Here, N = size of the array

```cpp
#include <bits/stdc++.h>
using namespace std;

// Recursive Bubble Sort with early exit optimization
void bubble_sort(int arr[], int n) {
    // Base Case: if the size is 1, it's already sorted
    if (n == 1) return;

    int didSwap = 0; // Flag to check if any swap occurred in this pass

    // One pass: push the largest element to the end
    for (int j = 0; j <= n - 2; j++) {
        if (arr[j] > arr[j + 1]) {
            // Swap arr[j] and arr[j + 1]
            int temp = arr[j + 1];
            arr[j + 1] = arr[j];
            arr[j] = temp;
            didSwap = 1; // Mark that we did a swap
        }
    }

    // If no swap happened, array is already sorted
    if (didSwap == 0) return;

    // Recurse for the remaining unsorted portion
    bubble_sort(arr, n - 1);
}
```

Q6. Recursive Insertion Sort

EASY

Topic: Learn Important Sorting Techniques > Sorting-II

Practice Link

Example 1:
Input: N = 6, array[] = {13,46,24,52,20,9}
Output: 9,13,20,24,46,52
Explanation: After sorting we get 9,13,20,24,46,52

Example 2:
Input: N = 5, array[] = {5,4,3,2,1}
Output: 1,2,3,4,5
Explanation: After sorting we get 1,2,3,4,5

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N2), (where N = size of the array), for the worst, and average cases. Space Complexity: O(N

```cpp
#include <bits/stdc++.h>
using namespace std;

// Recursive insertion sort function
void insertion_sort(int arr[], int i, int n) {
    // Base Case: if index reaches the size, return
    if (i == n) return;

    int j = i;
    // Shift the element to the left while it's smaller than its predecessor
    while (j > 0 && arr[j - 1] > arr[j]) {
        // Swap arr[j] and arr[j-1]
        int temp = arr[j - 1];
        arr[j - 1] = arr[j];
        arr[j] = temp;
        j--;
    }

    // Recur for the next index
    insertion_sort(arr, i + 1, n);
}
```

Q7. Quick Sorting

EASY

Topic: Learn Important Sorting Techniques > Sorting-II

Practice Link

Input: N = 5, Arr[] = {4,1,7,9,3}
Output: {1, 3, 4, 7, 9}
Explanation: After sorting the array in ascending order it becomes 1, 3, 4, 7, 9

Input: N = 8, Arr[] = {4,6,2,5,7,9,1,3}
Output: {1, 2, 3, 4, 5, 6, 7, 9}
Explanation: After sorting the array in ascending order it becomes 1, 2, 3, 4, 5, 6, 7, 9

[OPTIMAL SOLUTION] General — Time Complexity: O(N*logN), At each step, we divide the whole array, for that we take logN time and n steps ar

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to perform quicksort
    void quickSort(vector<int>& arr, int low,
      int high) {
        // Base case: if low is not less than
          high
        if (low < high) {
            // Find pivot index after
              partitioning
            int pivotIndex = partition(arr,
              low, high);

            // Recursively sort elements
              before pivot
            quickSort(arr, low, pivotIndex -
              1);

            // Recursively sort elements after
              pivot
            quickSort(arr, pivotIndex + 1,
              high);
        }
    }

    // Function to partition the array
    int partition(vector<int>& arr, int low,
      int high) {
        // Choose the last element as pivot
        int pivot = arr[high];

        // Initialize i to place smaller
          elements
        int i = low - 1;

        // Traverse the array
        for (int j = low; j < high; j++) {
            // If element is smaller than or
              equal to pivot
            if (arr[j] <= pivot) {
                // Increment i and swap with j
                i++;
                swap(arr[i], arr[j]);
            }
        }

        // Place pivot in correct position
        swap(arr[i + 1], arr[high]);

        // Return pivot index
        return i + 1;
    }
};
```

Q8. Largest Element

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Example 1:
Input: arr[] = {2, 5, 1, 3, 0}
Output: 5
Explanation:
5 is the largest element in the array.

Example 2:
Input: arr[] = {8, 10, 5, 7, 9}
Output: 10
Explanation:
10 is the largest element in the array.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the size of the array, as we are iterating through the array once.

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to find the largest element in the array
int findLargestElement(int arr[], int n) {
    int max = arr[0];  // Initialize max with the first element in the array

    // Iterate through the array to find the maximum element
    for (int i = 1; i < n; i++) {
        if (max < arr[i]) {  // If the current element is greater than max,
          update max
            max = arr[i];
        }
    }

    return max;  // Return the largest element found
}
```

Q9. Second Largest Element

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Example 1:
Input: [1, 2, 4, 7, 7, 5]
Output:
Second Smallest : 2
Second Largest : 5
Explanation: The elements are sorted as 1, 2, 4, 5, 7, 7.
Hence, the second smallest element is 2, and the second largest element is 5.

Example 2:
Input: [1]... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N), we do two linear traversals in our array. Space Complexity: O(*

```cpp
#include<bits/stdc++.h>
using namespace std;

// Function to find the second smallest and
  second largest elements in the array
void getElements(int arr[], int n)
{
    // Edge case: when the array has less than
      2 elements
    if (n == 0 || n == 1)
        cout << -1 << " " << -1 << endl;  //
          If only one element, print -1 for
          both second smallest and second
          largest

    // Initialize variables to track the
      smallest, second smallest, largest, and
      second largest elements
    int small = INT_MAX, second_small =
      INT_MAX;
    int large = INT_MIN, second_large =
      INT_MIN;
    int i;

    // Find the smallest and largest elements
      in the array
    for (i = 0; i < n; i++) {
        small = min(small, arr[i]);  // Update
          the smallest element
        large = max(large, arr[i]);  // Update
          the largest element
    }

    // Find the second smallest and second
      largest elements
    for (i = 0; i < n; i++) {
        // If the current element is smaller
          than second_small and not equal to
          the smallest, update second_small
        if (arr[i] < second_small && arr[i] !=
          small)
            second_small = arr[i];

        // If the current element is larger
          than second_large and not equal to
          the largest, update second_large
        if (arr[i] > second_large && arr[i] !=
          large)
            second_large = arr[i];
    }

    // Output the second smallest and second
      largest elements
    cout << "Second smallest is " <<
      second_small << endl;
    cout << "Second largest is " <<
      second_large << endl;
}
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N), we do two linear traversals in our array.*

```cpp
#include<bits/stdc++.h>
using namespace std;
// Function to find the second smallest
  element in the array
int secondSmallest(int arr[], int n) {
    // Edge case: if the array has fewer than
      2 elements
    if (n < 2)
        return -1;
    int small = INT_MAX;
    int second_small = INT_MAX;
    // Loop through the array to find the
      second smallest element
    for (int i = 0; i < n; i++) {
        // Update the smallest and second
          smallest values
        if (arr[i] < small) {
            second_small = small;
            small = arr[i];
        }
        else if (arr[i] < second_small &&
          arr[i] != small) {
            second_small = arr[i];
        }
    }
    return second_small; // Return the second
      smallest element
}
// Function to find the second largest element
  in the array
int secondLargest(int arr[], int n) {
    // Edge case: if the array has fewer than
      2 elements
    if (n < 2)
        return -1;
    int large = INT_MIN, second_large =
      INT_MIN;
    // Loop through the array to find the
      second largest element
    for (int i = 0; i < n; i++) {
        // Update the largest and second
          largest values
        if (arr[i] > large) {
            second_large = large;
            large = arr[i];
        }
        else if (arr[i] > second_large &&
          arr[i] != large) {
            second_large = arr[i];
        }
    }
    return second_large; // Return the second
      largest element
}
```

Q10. Check if the Array is Sorted II

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Example 1:
Input: N = 5, array[] = {1,2,3,4,5}
Output: True.
Explanation: The given array is sorted i.e Every element in the array is smaller than or equals to its next values, So the answer is True.

Example 2:
Input: N = 5, array[] = {5,4,6,7,8}
Output: False.
Explanation: The given array is Not sorted i.e Every element in the array is not smaller than or equal to its next values, So the answer is False.
Here element 5 is not smaller than or equal to its future elements.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), as it checks each adjacent pair once in a single pass through t

```cpp
#include<bits/stdc++.h>
using namespace std;

// Function to check if the array is sorted
bool isSorted(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        if (arr[i] < arr[i - 1])  // If any element is smaller than the
          previous one, return false
            return false;
    }

    return true;  // Return true if the array is sorted
}
```

Q11. Remove duplicates from Sorted array

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Input: arr[]=[1,1,2,2,2,3,3]
Output: [1,2,3,_,_,_,_]
Explanation: Total number of unique elements are 3, i.e[1,2,3] and Therefore return 3 after assigning [1,2,3] in the beginning of the array.

Input: arr[]=[1,1,1,2,2,3,3,3,3,4,4]
Output: [1,2,3,4,_,_,_,_,_,_,_]
Explanation: Total number of unique elements are 4, i.e[1,2,3,4] and Therefore return 4 after assigning [1,2,3,4] in the beginning of the array.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), We traverse the entire original array only once. Space Comp

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to hold the solution logic
class Solution {
public:
    // Function to remove duplicates from sorted array in-place
    int removeDuplicates(vector<int>& nums) {
        // If array is empty, return 0 directly
        if (nums.empty()) return 0;

        // Pointer for the position of last unique element
        int i = 0;

        // Traverse the array starting from the second element
        for (int j = 1; j < nums.size(); j++) {
            // If current element is different from last unique element
            if (nums[j] != nums[i]) {
                // Move pointer for unique element forward
                i++;
                // Place the new unique element at the next position
                nums[i] = nums[j];
            }
        }

        // i is index of last unique element, count = i + 1
        return i + 1;
    }
};
```

Q12. Left Rotate Array by One

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Example 1:
Input: nums = [1, 2, 3, 4, 5]
Output: [2, 3, 4, 5, 1]
Explanation: Initially, nums = [1, 2, 3, 4, 5]
Rotating once to the left results in nums = [2, 3, 4, 5, 1].

Example 2:
Input: nums = [-1, 0, 3, 6]
Output: [0, 3, 6, -1]
Explanation: Initially, nums = [-1, 0, 3, 6]
Rotating once to the left results in nums = [0, 3, 6, -1].

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the size of the input array. This is because we traverse the array once to s

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
public:
    void rotateArrayByOne(vector<int>& nums) {
        // Store the first element in a temporary variable
        int temp = nums[0];

        // Shift elements to the left
        for (int i = 1; i < nums.size(); ++i) {
            nums[i - 1] = nums[i];
        }

        // Place the first element at the end
        nums[nums.size() - 1] = temp;
    }
};
```

Q13. Left Rotate Array by K Places

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Input : nums = [1, 2, 3, 4, 5, 6, 7], k = 2, right
Output : [6, 7, 1, 2, 3, 4, 5]
Explanation : rotate 1 step to the right: [7, 1, 2, 3, 4, 5, 6]
rotate 2 steps to the right: [6, 7, 1, 2, 3, 4, 5]
Input : nums = [1, 2, 3, 4, 5, 6], k=2, left
Output : [3, 4, 5, 6, 1, 2]
Explanation :rotate 1 step to the left: [2, 3, 4, 5, 6, 1]
rotate 2 steps to the left: [3, 4, 5, 6, 1, 2]

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), We reverse parts of the array each reverse takes linear time. So total work is 3 × O(N)

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Helper function to reverse a subarray
      from index start to end
    void reverseArray(vector<int>& nums, int
      start, int end) {
        while (start < end) {
            swap(nums[start], nums[end]);
            start++;
            end--;
        }
    }

    // Function to rotate array left or right
      by k steps
    vector<int> rotateArray(vector<int>& nums,
      int k, string direction) {
        int n = nums.size();

        // Edge case: empty or no rotation
        if (n == 0 || k == 0) return nums;

        // Normalize k
        k = k % n;

        if (direction == "right") {
            // Step 1: Reverse the whole array
            reverseArray(nums, 0, n - 1);

            // Step 2: Reverse first k
              elements
            reverseArray(nums, 0, k - 1);

            // Step 3: Reverse remaining n-k
              elements
            reverseArray(nums, k, n - 1);
        }
        else if (direction == "left") {
            // Step 1: Reverse first k
              elements
            reverseArray(nums, 0, k - 1);

            // Step 2: Reverse remaining n-k
              elements
            reverseArray(nums, k, n - 1);

            // Step 3: Reverse the whole array
            reverseArray(nums, 0, n - 1);
        }

        return nums;
    }
};

// Driver code
```

Q14. Move Zeros to End

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Input: 1 ,0 ,2 ,3 ,0 ,4 ,0 ,1
Output: 1 ,2 ,3 ,4 ,1 ,0 ,0 ,0
Explanation: All the zeros are moved to the end and non-negative integers are moved to front by maintaining order

Input : 1,2,0,1,0,4,0
Output: 1,2,1,4,0,0,0
Explanation : All the zeros are moved to the end and non-negative integers are moved to front by maintaining order

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), we can move all zeroes to end in linear time. Space Complex

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to move zeroes to the end
    void moveZeroes(vector<int>& nums) {
        // Pointer to the first zero
        int j = -1;

        // Find the first zero
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] == 0) {
                j = i;
                break;
            }
        }

        // If no zero found, return
        if (j == -1) return;

        // Start from the next index of first zero
        for (int i = j + 1; i < nums.size(); i++) {
            // If current element is non-zero
            if (nums[i] != 0) {
                // Swap with nums[j]
                swap(nums[i], nums[j]);
                // Move j to next zero
                j++;
            }
        }
    }
};
```

Q15. Linear Search

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Example 1:
Input: arr[] = 1 2 3 4 5, num = 3
Output: 2 `
Explanation: 3 is present at the 2nd index of the array.

Example 2:
Input: arr[] = 5 4 3 2 1, num = 5
Output: 0
Explanation: 5 is present at the 0th index of the array.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the number of elements in the array. This is because we traverse the entire

```c
#include<stdio.h>

// Function to search for a number in the array
int search(int arr[], int n, int num)
{
    int i;

    // Loop through the array to find the number
    for(i = 0; i < n; i++)
    {
        // If the current element matches the number, return its index
        if(arr[i] == num)
            return i;
    }

    // If the number is not found, return -1
    return -1;
}
```

Q16. Union of two sorted arrays

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Input:n = 5,m = 5 arr1[] = {1,2,3,4,5} arr2[] = {2,3,4,4,5}
Output: {1,2,3,4,5}
Explanation: Common Elements in arr1 and arr2 are: 2,3,4,5
Distnict Elements in arr1 are : 1
Distnict Elemennts in arr2 are : No distinct elements.
Union of arr1 and arr2 is {1,2,3,4,5}
Input:n = 10,m = 7,arr1[] = {1,2,3,4,5,6,7,8,9,10}arr2[] = {2,3,4,4,5,11,12}
Output: {1,2,3,4,5,6,7,8,9,10,11,12}
Explanation: Common Elements in arr1 and arr2 are: 2,3,4,5
Distnict Elements in arr1 are : 1,6,7,8,9,10
Distnict El... (cont.)

[OPTIMAL SOLUTION] Optimal Approach - Two Pointers — Time Complexity: O(m+n), Because at max i runs for n times and j runs for m times. When there are no common el

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find union of two sorted
      arrays using two pointers
    vector<int> findUnion(int arr1[], int
      arr2[], int n, int m) {
        // Vector to store union elements
        vector<int> Union;
        // Initialize two pointers for both
          arrays
        int i = 0, j = 0;
        // Loop until either pointer reaches
          the end
        while (i < n && j < m) {
            // If current element in arr1 is
              smaller
            if (arr1[i] < arr2[j]) {
                // Add element if union empty
                  or different from last added
                if (Union.empty() ||
                  Union.back() != arr1[i])
                    Union.push_back(arr1[i]);
                i++;  // Move pointer in arr1
            }
            // If current element in arr2 is
              smaller
            else if (arr2[j] < arr1[i]) {
                // Add element if union empty
                  or different from last added
                if (Union.empty() ||
                  Union.back() != arr2[j])
                    Union.push_back(arr2[j]);
                j++;  // Move pointer in arr2
            }
            else {
                // Elements are equal, add
                  once if not duplicate
                if (Union.empty() ||
                  Union.back() != arr1[i])
                    Union.push_back(arr1[i]);
                i++; j++;  // Move both
                  pointers
            }
        }
        // Append remaining elements from arr1
        while (i < n) {
            if (Union.empty() || Union.back()
              != arr1[i])
                Union.push_back(arr1[i]);
            i++;
        }
        // Append remaining elements from arr2
        while (j < m) {
            if (Union.empty() || Union.back()
              != arr2[j])
                Union.push_back(arr2[j]);
            j++;
        }
        // Return the final union vector
        return Union;
    }
};
```

Q17. Find missing number

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```
int missingNum(vector<int>& arr) {
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```
#include <iostream>
#include <vector>
using namespace std;

int missingNum(vector<int>& arr) {
    int n = arr.size() + 1;

    // Iterate from 1 to n and check
    // if the current number is present
    for (int i = 1; i <= n; i++) {
        bool found = false;
        for (int j = 0; j < n - 1; j++) {
            if (arr[j] == i) {
                found = true;
                break;
            }
        }

        // If the current number is not
          present
        if (!found)
            return i;
    }
    return -1;
}
```

Q18. Maximum Consecutive Ones

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Example 1:
Input: prices = {1, 1, 0, 1, 1, 1}
Output: 3
Explanation: There are two consecutive 1's and three consecutive 1's in the array out of which maximum is 3.

Example 2:
Input: prices = {1, 0, 1, 1, 0, 1}
Output: 2
Explanation: There are two consecutive 1's in the array.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), since we scan the array once. Space Complexity: O(1), as only const

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find maximum consecutive 1's in an array
    int findMaxConsecutiveOnes(vector<int> &nums) {
        // Variable to store current count of consecutive 1's
        int cnt = 0;
        // Variable to store maximum consecutive 1's
        int maxi = 0;

        // Traverse the array
        for (int i = 0; i < nums.size(); i++) {
            // If current element is 1, increment count
            if (nums[i] == 1) {
                cnt++;
            } else {
                // If element is 0, reset count
                cnt = 0;
            }

            // Update maximum if current count is greater
            maxi = max(maxi, cnt);
        }

        // Return maximum consecutive 1's
        return maxi;
    }
};
```

Q19. Find the number that appears once, and other numbers twice.

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Example 1:
Input Format: arr[] = {2,2,1}
Result: 1
Explanation: In this array, only the element 1 appear once and so it is the answer.

Example 2:
Input Format: arr[] = {4,1,2,1,2}
Result: 4
Explanation: In this array, only element 4 appear once and the other... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N)+O(N)+O(N), where N = size of the array. One O(N) is for finding*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the single
      non-repeating element using a hash array
    int getSingleElement(vector<int>& arr) {
        int n = arr.size();

        // Step 1: Find the maximum value in
          the array
        int maxi = arr[0];
        for (int i = 0; i < n; i++) {
            maxi = max(maxi, arr[i]);
        }

        // Step 2: Declare and initialize hash
          array of size maxi+1
        vector<int> hash(maxi + 1, 0);

        // Step 3: Count the frequency of each
          number
        for (int i = 0; i < n; i++) {
            hash[arr[i]]++;
        }

        // Step 4: Find the element that
          occurs exactly once
        for (int i = 0; i < n; i++) {
            if (hash[arr[i]] == 1)
                return arr[i];
        }

        return -1; // fallback, shouldn't
          occur in valid input
    }
};
```

[OPTIMAL] Optimised Approach

Complexity: *Time Complexity: O(N). Where N is the size of the array Space*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the single
      non-repeating element using XOR
    int getSingleElement(vector<int>& arr) {
        int n = arr.size();
        int xorr = 0;

        // XOR all elements. Duplicates cancel
          out, leaving the single element.
        for (int i = 0; i < n; i++) {
            xorr = xorr ^ arr[i];
        }

        return xorr;
    }
};
```

Q20. Longest subarray with given sum K(positives)

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Example 1:
Input: nums = [10, 5, 2, 7, 1, 9], k = 15
Output: 4
Explanation: The longest sub-array with a sum equal to 15 is [5, 2, 7, 1], which has a length of 4. This sub-array starts at index 1 and ends at index 4, and the sum of its elements (5 + 2 + 7 + 1) equals 15. Therefore, the length of this sub-array is 4.

Example 2:
Input: nums = [-3, 2, 1], k = 6
Output: 0
Explanation: There is no sub-array in the array that sums to 6. Therefore, the output is 0.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the size of the array. The algorithm traverses the array once with two point

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution{
public:
    // Function to find the length of longest
      subarray having sum k
    int longestSubarray(vector<int> &nums, int
      k){
        int n = nums.size();

        // To store the maximum length of the
          subarray
        int maxLen = 0;

        // Pointers to mark the start and end
          of window
        int left = 0, right = 0;

        // To store the sum of elements in the
          window
        int sum = nums[0];

        // Traverse all the elements
        while(right < n) {
            // If the sum exceeds K, shrink
              the window
            while(left <= right && sum > k) {
                sum -= nums[left];
                left++;
            }

            // store the maximum length
            if(sum == k) {
                maxLen = max(maxLen, right -
                  left + 1);
            }

            right++;
            if(right < n) sum += nums[right];
        }

        return maxLen;
    }
};
```

Q21. Longest subarray with sum K

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Easy

Practice Link

Example 1:
Input: N = 6, array[] = {9, -3, 3, -1, 6, -5}
Result: 5
Explanation: The following subarrays sum to zero:
- {-3, 3}
- {-1, 6, -5}
- {-3, 3, -1, 6, -5}
The length of the longest subarray with sum zero is 5.

Example 2:
Input: N = 8, array[] = {6, -2, 2, -8, 1, 7, 4, -10}
Result: 8
Explanation: Subarrays with sum zero:
- {-2, 2}
- {-8, 1, 7}
- {-2, 2, -8, 1, 7}
- {6, -2, 2, -8, 1, 7, 4, -10}
The length of the longest subarray with sum zero is 8.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n), where n is the length of the string. This is because we are using a single pass through

```cpp
#include <bits/stdc++.h>
using namespace std;

// compute length of the longest subarray with
  sum 0
int maxLen(int A[], int n) {
  // map prefix sum -> first index seen
  unordered_map<int, int> mpp;
  // best length so far
  int maxi = 0;
  // running prefix sum
  int sum = 0;

  // iterate over the array
  for (int i = 0; i < n; i++) {
    // update running sum
    sum += A[i];

    // if sum is zero, subarray [0..i] has
      zero sum
    if (sum == 0) {
      // update best length
      maxi = i + 1;
    }
    // otherwise check if this sum was seen
      before
    else {
      // when seen, zero-sum segment between
        previous index + 1 and i
      if (mpp.find(sum) != mpp.end()) {
        // maximize length
        maxi = max(maxi, i - mpp[sum]);
      }
      // first time seeing this sum
      else {
        // record index
        mpp[sum] = i;
      }
    }
  }

  // return best length
  return maxi;
}

// program entry
```

Q22. Two Sum

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Input: N = 5, arr[] = {2,6,5,8,11}, target = 14
Output : YES
Explanation: arr[1] + arr[3] = 14. So, the answer is “YES” for first variant for second variant output will be : [1,3].
Input: N = 5, arr[] = {2,6,5,8,11}, target = 15
Output : NO.
Explanation: There... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N) because we traverse the array only once, and each lookup or ins*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Variant 1: Check if two numbers sum to
      target using hashing
    string twoSumExists(vector<int>& arr, int
      target) {
        unordered_map<int, int> mp;  // Map to
          store element -> index
        int n = arr.size();
        // Iterate over all elements
        for (int i = 0; i < n; i++) {
            int complement = target - arr[i];
              // Needed value to reach target
              sum
            // Check if complement is already
              in map
            if (mp.find(complement) !=
              mp.end()) {
                return "YES";  // Pair found
            }
            // Store current element with its
              index for future reference
            mp[arr[i]] = i;
        }
        // No pair found
        return "NO";
    }

    // Variant 2: Return indices of two
      numbers that sum to target using hashing
    vector<int> twoSumIndices(vector<int>&
      arr, int target) {
        unordered_map<int, int> mp;  // Map to
          store element -> index
        int n = arr.size();
        for (int i = 0; i < n; i++) {
            int complement = target - arr[i];
            // If complement found, return
              indices
            if (mp.find(complement) !=
              mp.end()) {
                return {mp[complement], i};
            }
            // Store current element and index
            mp[arr[i]] = i;
        }
        return {-1, -1};  // No pair found
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N log N) due to sorting the array initially, where N is the number*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Variant 1: Check if two numbers sum to
      target using two-pointer approach
    string twoSumExists(vector<int> arr, int
      target) {
        int n = arr.size();
        // Create a vector of pairs to
          remember original indices after
          sorting
        vector<pair<int, int>> numsWithIndex;
        for (int i = 0; i < n; i++) {
            numsWithIndex.push_back({arr[i],
              i});
        }
        // Sort based on the element values
        sort(numsWithIndex.begin(),
          numsWithIndex.end());
        int left = 0, right = n - 1;
        // Loop until pointers cross
        while (left < right) {
            int sum =
              numsWithIndex[left].first +
              numsWithIndex[right].first;
            if (sum == target) {
                return "YES";  // Pair found
            } else if (sum < target) {
                left++;  // Need bigger sum,
                  move left pointer right
            } else {
                right--; // Need smaller sum,
                  move right pointer left
            }
        }
        return "NO";  // No pair found
    }
    // Variant 2: Return original indices of
      two numbers that sum to target
    vector<int> twoSumIndices(vector<int> arr,
      int target) {
        int n = arr.size();
        vector<pair<int, int>> numsWithIndex;
        for (int i = 0; i < n; i++) {
            numsWithIndex.push_back({arr[i],
              i});
        }
        sort(numsWithIndex.begin(),
          numsWithIndex.end());
        int left = 0, right = n - 1;
        while (left < right) {
            int sum =
              numsWithIndex[left].first +
              numsWithIndex[right].first;
            if (sum == target) {
                // Return original indices
                return
                  {numsWithIndex[left].second,
                  numsWithIndex[right].second}
                  ;
            } else if (sum < target) {
                left++;
            } else {
                right--;
            }
        }
        return {-1, -1};  // No pair found
    }
};
```

Q23. Sort an array of 0's 1's and 2's

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Input: nums = [1, 0, 2, 1, 0]
Output: [0, 0, 1, 1, 2]
Explanation: The nums array in sorted order has 2 zeroes, 2 ones and 1 two
Input: nums = [0, 0, 1, 1, 1]
Output: [0, 0, 1, 1, 1]
Explanation: The nums array in sorted order has 2 zeroes, 3 ones and zero two... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity:,O(n) We make two passes through the array: one for counting and one*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to sort an array containing
      only 0s, 1s, and 2s
    void sortZeroOneTwo(vector<int>& nums) {
        // Count of 0s, 1s, and 2s
        int cnt0 = 0, cnt1 = 0, cnt2 = 0;

        // First pass: Count the number of 0s,
          1s, and 2s
        for (int i = 0; i < nums.size(); i++)
          {
            if (nums[i] == 0) cnt0++;
            else if (nums[i] == 1) cnt1++;
            else cnt2++;
        }

        // Second pass: Fill the array with
          0s, then 1s, then 2s

        // Fill the first 'cnt0' elements with
          0
        for (int i = 0; i < cnt0; i++) {
            nums[i] = 0;
        }

        // Fill the next 'cnt1' elements with
          1
        for (int i = cnt0; i < cnt0 + cnt1;
          i++) {
            nums[i] = 1;
        }

        // Fill the remaining elements with 2
        for (int i = cnt0 + cnt1; i <
          nums.size(); i++) {
            nums[i] = 2;
        }
    }
};

// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n) The array is traversed only once using the `mid` pointer. Each*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to sort array containing 0s,
      1s, and 2s using Dutch National Flag
      Algorithm
    void sortZeroOneTwo(vector<int>& nums) {
        // Initialize three pointers: low, mid
          starting from 0, high from end of
          array
        int low = 0, mid = 0, high =
          nums.size() - 1;

        // Process elements until mid pointer
          crosses high pointer
        while (mid <= high) {
            // If current element is 0, swap
              with low and move both pointers
              forward
            if (nums[mid] == 0) {
                swap(nums[mid], nums[low]);
                mid++;
                low++;
            }
            // If current element is 1, it's
              already in correct place → move
              mid forward
            else if (nums[mid] == 1) {
                mid++;
            }
            // If current element is 2, swap
              with high and move only high
              pointer backward
            else {
                swap(nums[mid], nums[high]);
                high--;
            }
        }
    }
};

// Driver code
```

Q24. Majority Element-I

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Example 1:
Input: nums = [7, 0, 0, 1, 7, 7, 2, 7, 7]
Output: 7
Explanation: The number 7 appears 5 times in the 9-sized array, making it the most frequent element.

Example 2:
Input: nums = [1, 1, 1, 2, 1, 2]
Output: 1
Explanation: The number 1 app... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N), where N is the size of the input array. This is because we are*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the majority element
      in an array
    int majorityElement(vector<int>& nums) {

        // Size of the given array
        int n = nums.size();

        // Hash map to store element counts
        unordered_map<int, int> mp;

        // Count occurrences of each element
        for (int num : nums) {
            mp[num]++;
        }

        /* Iterate through the map to
        find the majority element*/
        for (auto& pair : mp) {
            if (pair.second > n / 2) {
                return pair.first;
            }
        }

        // Return -1 if no majority element is
          found
        return -1;
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N), where N is the size of the input array. This is because we are*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the majority element
      in an array
    int majorityElement(vector<int>& nums) {

        // Size of the given array
        int n = nums.size();

        // Count
        int cnt = 0;

        // Element
        int el;

        // Applying the algorithm
        for (int i = 0; i < n; i++) {
            if (cnt == 0) {
                cnt = 1;
                el = nums[i];
            } else if (el == nums[i]) {
                cnt++;
            } else {
                cnt--;
            }
        }

        /* Checking if the stored element
         is the majority element*/
        int cnt1 = 0;
        for (int i = 0; i < n; i++) {
            if (nums[i] == el) {
                cnt1++;
            }
        }

        //return element if it is a majority
          element
        if (cnt1 > (n / 2)) {
            return el;
        }

        //return -1 if no such element found
        return -1;
    }
};
```

Q25. Kadane's Algorithm

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Example 1:
Input: nums = [2, 3, 5, -2, 7, -4]
Output: 15
Explanation: The subarray from index 0 to index 4 has the largest sum = 15, which is the maximum sum of any contiguous subarray.

Example 2:
Input: nums = [-2, -3, -7, -2, -10, -4]
Output: -2... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N^2), where N is the size of the array. This is because we have tw*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find maximum sum of
      subarrays
    int maxSubArray(vector<int>& nums) {

        /* Initialize maximum sum with
        the smallest possible integer*/
        int maxi = INT_MIN;

        // Iterate over each starting index of
          subarrays
        for (int i = 0; i < nums.size(); i++)
          {

            /* Variable to store the sum
            of the current subarray*/
            int sum = 0;

            /* Iterate over each ending index
            of subarrays starting from i*/
            for (int j = i; j < nums.size();
              j++) {

                /* Add the current element
                  nums[j] to
                the sum i.e. sum of
                  nums[i...j-1]*/
                sum += nums[j];

                /* Update maxi with the
                  maximum of its current
                value and the sum of the
                  current subarray*/
                maxi = max(maxi, sum);
            }
        }

        // Return the maximum subarray sum
          found
        return maxi;
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n), where n is the number of elements in the array. We traverse th*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find maximum sum of
      subarrays and print the subarray having
      maximum sum
    int maxSubArray(vector<int>& nums) {
        // maximum sum
        long long maxi = LLONG_MIN;
        // current sum of subarray
        long long sum = 0;
        // starting index of current subarray
        int start = 0;
        // indices of the maximum sum subarray
        int ansStart = -1, ansEnd = -1;
        // Iterate through the array
        for (int i = 0; i < nums.size(); i++)
          {
            // update starting index if sum is
              reset
            if (sum == 0) {
                start = i;
            }
            // add current element to the sum
            sum += nums[i];
            if (sum > maxi) {
                maxi = sum;
                ansStart = start;
                ansEnd = i;
            }
            // Reset sum to 0 if it becomes
              negative
            if (sum < 0) {
                sum = 0;
            }
        }
        // Printing the subarray
        cout << "The subarray is: [";
        for (int i = ansStart; i <= ansEnd;
          i++) {
            cout << nums[i] << " ";
        }
        cout << "]" << endl;
        // Return the maximum subarray sum
          found
        return maxi;
    }
};
```

Q26. Print subarray with maximum subarray sum (extended version of above problem)

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Example 1:
Input: nums = [2, 3, 5, -2, 7, -4]
Output: 15
Explanation: The subarray from index 0 to index 4 has the largest sum = 15, which is the maximum sum of any contiguous subarray.

Example 2:
Input: nums = [-2, -3, -7, -2, -10, -4]
Output: -2... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N^2), where N is the size of the array. This is because we have tw*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find maximum sum of
      subarrays
    int maxSubArray(vector<int>& nums) {

        /* Initialize maximum sum with
        the smallest possible integer*/
        int maxi = INT_MIN;

        // Iterate over each starting index of
          subarrays
        for (int i = 0; i < nums.size(); i++)
          {

            /* Variable to store the sum
            of the current subarray*/
            int sum = 0;

            /* Iterate over each ending index
            of subarrays starting from i*/
            for (int j = i; j < nums.size();
              j++) {

                /* Add the current element
                  nums[j] to
                the sum i.e. sum of
                  nums[i...j-1]*/
                sum += nums[j];

                /* Update maxi with the
                  maximum of its current
                value and the sum of the
                  current subarray*/
                maxi = max(maxi, sum);
            }
        }

        // Return the maximum subarray sum
          found
        return maxi;
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n), where n is the number of elements in the array. We traverse th*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find maximum sum of
      subarrays and print the subarray having
      maximum sum
    int maxSubArray(vector<int>& nums) {
        // maximum sum
        long long maxi = LLONG_MIN;
        // current sum of subarray
        long long sum = 0;
        // starting index of current subarray
        int start = 0;
        // indices of the maximum sum subarray
        int ansStart = -1, ansEnd = -1;
        // Iterate through the array
        for (int i = 0; i < nums.size(); i++)
          {
            // update starting index if sum is
              reset
            if (sum == 0) {
                start = i;
            }
            // add current element to the sum
            sum += nums[i];
            if (sum > maxi) {
                maxi = sum;
                ansStart = start;
                ansEnd = i;
            }
            // Reset sum to 0 if it becomes
              negative
            if (sum < 0) {
                sum = 0;
            }
        }
        // Printing the subarray
        cout << "The subarray is: [";
        for (int i = ansStart; i <= ansEnd;
          i++) {
            cout << nums[i] << " ";
        }
        cout << "]" << endl;
        // Return the maximum subarray sum
          found
        return maxi;
    }
};
```

Q27. Stock Buy and Sell

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note: That buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n),This is because we are iterating through the array of prices exactly once. There are no

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to calculate maximum profit using single pass
    int stockbuySell(vector<int>& prices) {
        // Initialize the minimum price to a large number
        int minPrice = INT_MAX;

        // Initialize the maximum profit to 0
        int maxProfit = 0;

        // Traverse each price in the array
        for (int price : prices) {
            // If current price is less than minPrice, update minPrice
            if (price < minPrice) {
                minPrice = price;
            }
            // Else calculate profit and update maxProfit if it's greater
            else {
                maxProfit = max(maxProfit, price - minPrice);
            }
        }

        // Return the maximum profit found
        return maxProfit;
    }
};

// Driver code
```

Q28. Rearrange array elements by sign

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Example 1:
Input:
arr[] = {1,2,-4,-5}, N = 4
Output:
1 -4 2 -5
Explanation:
Positive elements = 1,2
Negative elements = -4,-5
To maintain relative ordering, 1 must occur before 2, and -4 must occur before -5.

Example 2:
Input:
arr[] = {1,2,-3,-1,-2,-3}, N = 6
Output:
1 -3 2 -1 3 -2
Explanation:
Positive elements = 1,2,3
Negative elements = -3,-1,-2
To maintain relative ordering, 1 must occur before 2, and 2 must occur before 3.
Also, -3 should come before -1, and -1 should come before -2.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N) { O(N) for traversing the array once and substituting positives and negatives simultaneo

```cpp
#include <bits/stdc++.h>
using namespace std;

// Define a class to handle array manipulation
class ArrayManipulator {
public:
    // Function to rearrange elements by alternating sign
    vector<int> rearrangeBySign(vector<int>& A) {
        int n = A.size();

        // Create a result array of size n initialized with 0
        vector<int> ans(n, 0);

        // posIndex will store index for next positive number (even index)
        // negIndex will store index for next negative number (odd index)
        int posIndex = 0, negIndex = 1;

        // Loop through the original array
        for (int i = 0; i < n; i++) {
            if (A[i] < 0) {
                // Place negative numbers at odd indices
                ans[negIndex] = A[i];
                negIndex += 2;
            } else {
                // Place positive numbers at even indices
                ans[posIndex] = A[i];
                posIndex += 2;
            }
        }

        return ans;
    }
};
```

Q29. Next Permutation

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Input: Arr[] = {1,3,2}
Output: {2,1,3}
Explanation: All permutations of {1,2,3} are {{1,2,3} , {1,3,2}, {2,13} , {2,3,1} , {3,1,2} , {3,2,1}}. So, the next permutation just after {1,3,2} is {2,1,3}.

Input : Arr[] = {3,2,1}
Output: {1,2,3}
Explanation : As we see all permutations of {1,2,3}, we find {3,2,1} at the last position. So, we have to return the lowest permutation.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), we find the breaking point and reverse the subarray in linear time.

```cpp
#include <bits/stdc++.h>
using namespace std;

// Solution class
class Solution {
public:
    // Function to find next permutation
    void nextPermutation(vector<int>& nums) {
        // Set index to -1
        int index = -1;

        // Find the first decreasing element
          from end
        for (int i = nums.size() - 2; i >= 0;
          i--) {
            // If a smaller element found
            if (nums[i] < nums[i + 1]) {
                // Store index
                index = i;
                break;
            }
        }

        // If no such index found
        if (index == -1) {
            // Reverse the entire array
            reverse(nums.begin(), nums.end());
            return;
        }

        // Find element just greater than
          nums[index]
        for (int i = nums.size() - 1; i >
          index; i--) {
            // Swap the two
            if (nums[i] > nums[index]) {
                swap(nums[i], nums[index]);
                break;
            }
        }

        // Reverse the part after index
        reverse(nums.begin() + index + 1,
          nums.end());
    }
};

// Main function
```

Q30. Leaders in an Array

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Example 1:
Input: arr = [4, 7, 1, 0]
Output: 7 1 0
Explanation: The rightmost element (0) is always a leader.
7 and 1 are greater than the elements to their right, making them leaders as well.

Example 2:
Input: arr = [10, 22, 12, 3, 0, 6]
Output: 22 12 6
Explanation: 6 is a leader because there are no elements after it.
12 is greater than all the elements to its right (3, 0, 6), and 22 is greater than 12, 3, 0, 6, making them leaders as well.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the size of the input array. This is because we traverse the array only once

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
public:
    //Function to find the leaders in an array.
    vector<int> leaders(vector<int>& nums) {
        vector<int> ans;

        if(nums.empty()) {
            return ans;
        }

        // Last element of the vector is always a leader
        int max = nums[nums.size() - 1];
        ans.push_back(nums[nums.size() - 1]);

        // Check elements from right to left
        for (int i = nums.size() - 2; i >= 0; i--) {
            if (nums[i] > max) {
                ans.push_back(nums[i]);
                max = nums[i];
            }
        }

        /* Reverse the vector to match
        the required output order*/
        reverse(ans.begin(), ans.end());

        //Return the leaders
        return ans;
    }
};
```

Q31. Longest Consecutive Sequence in an Array

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Example 1:
Input: nums = [100, 4, 200, 1, 3, 2]
Output: 4
Explanation: The longest sequence of consecutive elements in the array is [1, 2, 3, 4], which has a length of 4. This sequence can be formed regardless of the initial order of the elements in the... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(n log n), where n is the number of elements in the array. This is*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int longestConsecutive(vector<int>& nums)
      {
        int n = nums.size();

        // Return 0 if array is empty
        if (n == 0) return 0;

        sort(nums.begin(), nums.end());

        // Track last smaller element
        int lastSmaller = INT_MIN;
        // Count current sequence length
        int cnt = 0;
        // Track longest sequence length
        int longest = 1;

        for (int i = 0; i < n; i++) {
            // If consecutive number exists
            if (nums[i] - 1 == lastSmaller) {
                // Increment sequence count
                cnt += 1;
                // Update last smaller element
                lastSmaller = nums[i];
            }
            // If consecutive number doesn't
              exits
            else if (nums[i] != lastSmaller) {
                // Reset count for new
                  sequence
                cnt = 1;
                // Update last smaller element
                lastSmaller = nums[i];
            }
            // Update longest if needed
            longest = max(longest, cnt);
        }
        return longest;
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n), where n is the number of elements in the array. This is becaus*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int longestConsecutive(vector<int>& a) {
        int n = a.size();
        // If the array is empty
        if (n == 0) return 0;

        // Initialize the longest sequence
          length
        int longest = 1;
        unordered_set<int> st;

        // Put all the array elements into the
          set
        for (int i = 0; i < n; i++) {
            st.insert(a[i]);
        }

        /* Traverse the set to
           find the longest sequence  */
        for (auto it : st) {
            // Check if 'it' is a starting
              number of a sequence
            if (st.find(it - 1) == st.end()) {
                // Initialize the count of the
                  current sequence
                int cnt = 1;
                // Starting element of the
                  sequence
                int x = it;

                // Find consecutive numbers in
                  the set
                while (st.find(x + 1) !=
                  st.end()) {
                    // Move to the next
                      element in the sequence
                    x = x + 1;
                    // Increment the count of
                      the sequence
                    cnt = cnt + 1;
                }
                // Update the longest sequence
                  length
                longest = max(longest, cnt);
            }
        }
        return longest;
    }
};
```

Q32. Set Matrix Zeroes

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Input: matrix=[[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
Explanation: Since matrix[2][2]=0.Therfore the 2nd column and 2nd row wil be set to 0.
Input: matrix=[[0,1,2,0],[3,4,5,2],[1,3,1,5]]
Output:[[0,0,0,0],[0,4,5,0],[0,3,1,0]]
Explanation: Si... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(m × n),We make two passes over the matrix.First pass to identify r*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to set entire row and column
      to 0 if an element in the matrix is 0
    void setZeroes(vector<vector<int>>&
      matrix) {
        // Get number of rows
        int m = matrix.size();
        // Get number of columns
        int n = matrix[0].size();

        // Create row marker array
        vector<int> row(m, 0);
        // Create column marker array
        vector<int> col(n, 0);

        // First pass: mark rows and columns
          that need to be zeroed
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                // If element is zero, mark
                  its row and column
                if (matrix[i][j] == 0) {
                    row[i] = 1;
                    col[j] = 1;
                }
            }
        }

        // Second pass: set cells to zero
          based on markers
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                // If the row or column is
                  marked, set cell to zero
                if (row[i] == 1 || col[j] ==
                  1) {
                    matrix[i][j] = 0;
                }
            }
        }
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(m × n),We iterate over the entire matrix a constant number of time*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    void setZeroes(vector<vector<int>>&
      matrix) {
        int m = matrix.size();
        int n = matrix[0].size();
        bool firstRowZero = false;
        bool firstColZero = false;
        for (int j = 0; j < n; j++) {
            if (matrix[0][j] == 0) {
                firstRowZero = true;
                break;
            }
        }
        for (int i = 0; i < m; i++) {
            if (matrix[i][0] == 0) {
                firstColZero = true;
                break;
            }
        }
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                if (matrix[i][j] == 0) {
                    matrix[i][0] = 0;
                    matrix[0][j] = 0;
                }
            }
        }
        for (int i = 1; i < m; i++) {
            for (int j = 1; j < n; j++) {
                if (matrix[i][0] == 0 ||
                  matrix[0][j] == 0) {
                    matrix[i][j] = 0;
                }
            }
        }
        if (firstRowZero) {
            for (int j = 0; j < n; j++) {
                matrix[0][j] = 0;
            }
        }
        if (firstColZero) {
            for (int i = 0; i < m; i++) {
                matrix[i][0] = 0;
            }
        }
    }
};
```

Q33. Rotate matrix by 90 degrees

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Input :matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

Output :matrix = [[7, 4, 1], [8, 5, 2], [9, 6, 3]]

Explanation :First, we transpose the matrix: rows become columns. Then, we reverse each row to simulate 90° clockwise rotation. So element at (0,0) goes to (0,2), (0,1) goes to (1,2), and so on, achieving the rotated layout.
Input :matrix = [[0, 1, 1, 2], [2, 0, 3, 1], [4, 5, 0, 5], [5, 6, 7, 0]]

Output :matrix = [[5, 4, 2, 0], [6, 5, 0, 1], [7, 0, 3, 1], [0, 5, 1, 2]]

Explanation... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N²),We traverse every element once during transposition and again during reversal of each r

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to rotate matrix 90 degrees clockwise in-place
    void rotateClockwise(vector<vector<int>>& matrix) {
        int n = matrix.size();

        // Step 1: Transpose the matrix
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                // Swap element at (i, j) with (j, i) to transpose
                swap(matrix[i][j], matrix[j][i]);
            }
        }

        // Step 2: Reverse each row
        for (int i = 0; i < n; ++i) {
            // Reverse the current row to complete clockwise rotation
            reverse(matrix[i].begin(), matrix[i].end());
        }
    }
};

// Driver code
```

Q34. Print the matrix in spiral manner

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Input: Matrix[][] = { { 1, 2, 3, 4 },{ 5, 6, 7, 8 },{ 9, 10, 11, 12 },{ 13, 14, 15, 16 } }
Outhput: 1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10.
Explanation: The output of matrix in spiral form.
Input: Matrix[][] = { { 1, 2, 3 }, { 4, 5, 6 },{ 7, 8, 9 } }
Output: 1, 2, 3, 6, 9, 8, 7, 4, 5.
Explanation: The output of matrix in spiral form.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(m × n),Because we visit each element of the matrix exactly once, where `m` is the number of

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to return matrix in spiral
      order
    vector<int>
      spiralOrder(vector<vector<int>>& matrix)
      {
        // Vector to store final spiral order
          result
        vector<int> result;
        // Get number of rows and columns
        int top = 0;
        int bottom = matrix.size() - 1;
        int left = 0;
        int right = matrix[0].size() - 1;
        // Traverse the matrix in spiral order
        while(top <= bottom && left <= right)
          {
            // Traverse from left to right
              across the top row
            for(int i = left; i <= right; i++)
              {
result.push_back(matrix[top][i]);
            }
            top++; // Move top boundary down
            // Traverse from top to bottom on
              the right column
            for(int i = top; i <= bottom; i++)
              {
result.push_back(matrix[i][right]);
            }
            right--; // Move right boundary
              left
            // Check if there are rows
              remaining
            if(top <= bottom) {
                // Traverse from right to left
                  on the bottom row
                for(int i = right; i >= left;
                  i--) {
result.push_back(matrix[bottom][i]);
                }
                bottom--; // Move bottom
                  boundary up
            }
            // Check if there are columns
              remaining
            if(left <= right) {
                // Traverse from bottom to top
                  on the left column
                for(int i = bottom; i >= top;
                  i--) {
result.push_back(matrix[i][left]);
                }
                left++; // Move left boundary
                  right
            }
        }
        // Return the final spiral order
        return result;
    }
};
// Driver code
```

Q35. Count subarrays with given sum

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Medium

Practice Link

Input : N = 4, array[] = {3, 1, 2, 4}, k = 6
Output: 2
Explanation: The subarrays that sum up to 6 are [3, 1, 2] and [2, 4].
Input: N = 3, array[] = {1,2,3}, k = 3
Output: 2
Explanation: The subarrays that sum up to 3 are [1, 2], and [3].

[BETTER] Better Approach

Complexity: *Time Complexity: O(n²),We use two nested loops to check all subarrays, where n is the*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find count of subarrays
      with sum equal to k
    int subarraySum(vector<int>& arr, int k) {
        // Size of the array
        int n = arr.size();

        // Initialize count of subarrays
        int count = 0;

        // Traverse all possible start indices
        for (int i = 0; i < n; i++) {
            // Initialize sum for current
              subarray
            int sum = 0;

            // Traverse all possible end
              indices from start
            for (int j = i; j < n; j++) {
                // Add current element to sum
                sum += arr[j];

                // If sum equals k, increment
                  count
                if (sum == k) {
                    count++;
                }
            }
        }

        // Return total count of subarrays
        return count;
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n) We traverse the array once, where n is the size of the array. E*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find count of subarrays
      with sum equal to k using prefix sums
      and hashmap
    int subarraySum(vector<int>& arr, int k) {
        // Size of the array
        int n = arr.size();

        // Map to store frequency of prefix
          sums
        unordered_map<int, int>
          prefixSumCount;

        // Initialize prefix sum and count of
          subarrays
        int prefixSum = 0;
        int count = 0;

        // Base case: prefix sum 0 has
          occurred once
        prefixSumCount[0] = 1;

        // Traverse through the array
        for (int i = 0; i < n; i++) {
            // Add current element to prefix
              sum
            prefixSum += arr[i];

            // Calculate the prefix sum that
              needs to be removed
            int remove = prefixSum - k;

            // If this prefix sum has been
              seen before,
            // add its count to the result
            if (prefixSumCount.find(remove) !=
              prefixSumCount.end()) {
                count +=
                  prefixSumCount[remove];
            }

            // Update the frequency of the
              current prefix sum
            prefixSumCount[prefixSum]++;
        }

        // Return the total count of subarrays
        return count;
    }
};
```

Q36. Pascal's Triangle I

EASY

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Input: N = 5, r = 5, c = 3
Output: Element at position (r, c): 6
N-th row of Pascal's triangle: 1 4 6 4 1
First n rows of Pascal's triangle:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
Explanation: Pascal triangle for first 5 rows is shown above.

Input: N = 1, r = 1... (cont.)

[BETTER] Approach - 2

Complexity: *Time Complexity: O(N), we iterate N times to compute each element of the row in O(1)*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class containing Pascal's Triangle row
  generation logic
class Solution {
public:
    // Function to generate the Nth row of
      Pascal's Triangle
    vector<long long> getNthRow(int N) {
        // Result vector to store the row
        vector<long long> row;

        // First value of the row is always 1
        long long val = 1;
        row.push_back(val);

        // Compute remaining values using the
          relation:
        // C(n, k) = C(n, k-1) * (n-k) / k
        for (int k = 1; k < N; k++) {
            val = val * (N - k) / k;
            row.push_back(val);
        }

        return row;
    }
};
```

[OPTIMAL] Approach - 1

Complexity: *Time Complexity: O(N^2), we generate all the elements in first N rows sequentially on*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class containing Pascal's Triangle
  generation logic
class Solution {
public:
    // Function to generate Pascal's Triangle
      up to numRows
    vector<vector<int>> generate(int numRows)
      {
        // Result vector to hold all rows
        vector<vector<int>> triangle;

        // Loop for each row
        for (int i = 0; i < numRows; i++) {
            // Create a row with size (i+1)
              and initialize all elements to 1
            vector<int> row(i + 1, 1);

            // Fill elements from index 1 to
              i-1 (middle values)
            for (int j = 1; j < i; j++) {
                // Each element = sum of two
                  elements above it
                row[j] = triangle[i - 1][j -
                  1] + triangle[i - 1][j];
            }

            // Add current row to the triangle
            triangle.push_back(row);
        }
        return triangle;
    }
};
```

Q37. Majority Element-II

HARD

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Example 1:
Input: nums = [1, 2, 1, 1, 3, 2]
Output: [1]
Explanation: Here, n / 3 = 6 / 3 = 2.
Therefore, the elements appearing 3 or more times are: [1].

Example 2:
Input: nums = [1, 2, 1, 1, 3, 2, 2]
Output: [1, 2]
Explanation: Here, n / 3 = 7... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N * logN), where N is the size of the given array. For using a map*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find majority elements in
      an array
    vector<int>
      majorityElementTwo(vector<int>& nums) {

        // size of the array
        int n = nums.size();

        // list of answers
        vector<int> result;

        // declaring a map
        unordered_map<int, int> mpp;

        // least occurrence of the majority
          element
        int mini = int(n / 3) + 1;

        // storing the elements with its
          occurrence
        for (int i = 0; i < n; i++) {
            mpp[nums[i]]++;

            // checking if nums[i] is the
              majority element
            if (mpp[nums[i]] == mini) {
                result.push_back(nums[i]);
            }

            // if result size is equal to 2
              break out of loop
            if (result.size() == 2) {
                break;
            }
        }

        // return the majority elements
        return result;
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N), where N is the size of the input array. We traverse the array*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    vector<int>
      majorityElementTwo(vector<int>& nums) {
        int n = nums.size();
        int cnt1 = 0, cnt2 = 0;
        int el1 = INT_MIN, el2 = INT_MIN;
        for (int i = 0; i < n; i++) {
            if (cnt1 == 0 && el2 != nums[i]) {
                cnt1 = 1;
                el1 = nums[i];
            }
            else if (cnt2 == 0 && el1 !=
              nums[i]) {
                cnt2 = 1;
                el2 = nums[i];
            }
            else if (nums[i] == el1) {
                cnt1++;
            }
            else if (nums[i] == el2) {
                cnt2++;
            }
            else {
                cnt1--;
                cnt2--;
            }
        }
        cnt1 = 0, cnt2 = 0;
        for (int i = 0; i < n; i++) {
            if (nums[i] == el1) {
                cnt1++;
            }
            if (nums[i] == el2) {
                cnt2++;
            }
        }
        int mini = n / 3 + 1;
        vector<int> result;
        if (cnt1 >= mini) {
            result.push_back(el1);
        }
        if (cnt2 >= mini && el1 != el2) {
            result.push_back(el2);
        }
        return result;
    }
};
```

Q38. 3 Sum

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Example 1:
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation: Out of all possible unique triplets possible, [-1,-1,2] and [-1,0,1] satisfy the condition of summing up to zero with i!=j!=k

Example 2:
Input: nums=[-1,0,1,0]
Output: Outpu... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N2 * log(no. of unique triplets)),as we are mainly using 3 nested*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to solve 3-sum problem
class Solution {
public:
    // Function to find triplets with sum zero
    vector<vector<int>> threeSum(vector<int>&
      arr, int n) {
        // Store unique triplets
        set<vector<int>> ans;

        // First loop for first element
        for (int i = 0; i < n; i++) {
            // Set to store elements seen in
              this iteration
            set<int> hashset;

            // Second loop for second element
            for (int j = i + 1; j < n; j++) {
                // Calculate third element
                  needed
                int third = -(arr[i] +
                  arr[j]);

                // If third already in set, we
                  found a triplet
                if (hashset.find(third) !=
                  hashset.end()) {
                    vector<int> temp =
                      {arr[i], arr[j], third};
                    sort(temp.begin(),
                      temp.end());
                    ans.insert(temp);
                }

                // Add current element to set
                hashset.insert(arr[j]);
            }
        }

        // Convert set to vector
        return
          vector<vector<int>>(ans.begin(),
          ans.end());
    }
};

// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(NlogN)+O(N2), as The pointer i, is running for approximately N tim*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to solve 3-sum problem
class Solution {
public:
    // Function to find triplets with sum zero
    vector<vector<int>> threeSum(vector<int>&
      arr, int n) {
        // Sort the array
        sort(arr.begin(), arr.end());
        // Store final result
        vector<vector<int>> ans;

        // First loop for first element
        for (int i = 0; i < n; i++) {
            // Skip duplicates for first
              element
            if (i > 0 && arr[i] == arr[i - 1])
              continue;

            // Two pointers
            int left = i + 1, right = n - 1;

            // Find pairs for current arr[i]
            while (left < right) {
                int sum = arr[i] + arr[left] +
                  arr[right];

                if (sum == 0) {
                    ans.push_back({arr[i],
                      arr[left], arr[right]});
                    left++, right--;

                    // Skip duplicates for
                      left
                    while (left < right &&
                      arr[left] == arr[left -
                      1]) left++;
                    // Skip duplicates for
                      right
                    while (left < right &&
                      arr[right] == arr[right
                      + 1]) right--;
                }
                else if (sum < 0) left++;
                else right--;
            }
        }
        return ans;
    }
};

// Driver code
```

Q39. 4 Sum

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Example 1:
Input Format:arr[] = [1,0,-1,0,-2,2], target = 0
Result: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]
Explanation: We have to find unique quadruplets from the array such that the sum of those elements is equal to the target sum given that is 0. The result obt... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N3*log(M)), as we are mainly using 3 nested loops, and inside the*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find all unique quadruplets
    vector<vector<int>> fourSum(vector<int>&
      arr, int target) {
        int n = arr.size();
        set<vector<int>> st;

        // First loop - pick first number
        for (int i = 0; i < n; i++) {
            // Second loop - pick second
              number
            for (int j = i + 1; j < n; j++) {
                // HashSet to store numbers
                  between j and k
                unordered_set<int> seen;

                // Third loop - pick third
                  number
                for (int k = j + 1; k < n;
                  k++) {
                    // Find required fourth
                      number
                    long long required = (long
                      long)target - arr[i] -
                      arr[j] - arr[k];

                    // If required number is
                      already in set, we found
                      a quadruplet
                    if (seen.count(required))
                      {
                        vector<int> temp =
                          {arr[i], arr[j],
                          arr[k],
                          (int)required};
                        sort(temp.begin(),
                          temp.end());
                        st.insert(temp);
                    }

                    // Add current third
                      number into set
                    seen.insert(arr[k]);
                }
            }
        }

        // Convert set into vector and return
        return vector<vector<int>>(st.begin(),
          st.end());
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N3), as Each of the pointers i and j, is running for approximately*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find all unique quadruplets
    vector<vector<int>> fourSum(vector<int>&
      arr, int target) {
        int n = arr.size();
        vector<vector<int>> ans;
        // Step 1: Sort array
        sort(arr.begin(), arr.end());
        // Step 2: First loop for first number
        for (int i = 0; i < n; i++) {
            // Skip duplicates for first
              number
            if (i > 0 && arr[i] == arr[i - 1])
              continue;
            // Step 3: Second loop for second
              number
            for (int j = i + 1; j < n; j++) {
                // Skip duplicates for second
                  number
                if (j > i + 1 && arr[j] ==
                  arr[j - 1]) continue;
                // Step 4: Two pointers for
                  remaining two numbers
                int left = j + 1, right = n -
                  1;
                while (left < right) {
                    long long sum = (long
                      long)arr[i] + arr[j] +
                                    arr[left]
                                      + arr[ri
                                      ght];
                    if (sum == target) {
                        ans.push_back({arr[i],
                          arr[j],
arr[left], arr[right]});
                        // Move left pointer
                          skipping duplicates
                        while (left < right &&
                          arr[left] ==
                          arr[left + 1])
                            left++;
                        // Move right pointer
                          skipping duplicates
                        while (left < right &&
                          arr[right] ==
                          arr[right - 1])
                            right--;
                        left++;
                        right--;
                    }
                    else if (sum < target)
                      left++;
                    else right--;
                }
            }
        }
        return ans;
    }
};
```

Q40. Largest Subarray with Sum 0

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Example 1:
Input: N = 6, array[] = {9, -3, 3, -1, 6, -5}
Result: 5
Explanation: The following subarrays sum to zero:
- {-3, 3}
- {-1, 6, -5}
- {-3, 3, -1, 6, -5}
The length of the longest subarray with sum zero is 5.

Example 2:
Input: N = 8, array[] = {6, -2, 2, -8, 1, 7, 4, -10}
Result: 8
Explanation: Subarrays with sum zero:
- {-2, 2}
- {-8, 1, 7}
- {-2, 2, -8, 1, 7}
- {6, -2, 2, -8, 1, 7, 4, -10}
The length of the longest subarray with sum zero is 8.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n), where n is the length of the string. This is because we are using a single pass through

```cpp
#include <bits/stdc++.h>
using namespace std;

// compute length of the longest subarray with
  sum 0
int maxLen(int A[], int n) {
  // map prefix sum -> first index seen
  unordered_map<int, int> mpp;
  // best length so far
  int maxi = 0;
  // running prefix sum
  int sum = 0;

  // iterate over the array
  for (int i = 0; i < n; i++) {
    // update running sum
    sum += A[i];

    // if sum is zero, subarray [0..i] has
      zero sum
    if (sum == 0) {
      // update best length
      maxi = i + 1;
    }
    // otherwise check if this sum was seen
      before
    else {
      // when seen, zero-sum segment between
        previous index + 1 and i
      if (mpp.find(sum) != mpp.end()) {
        // maximize length
        maxi = max(maxi, i - mpp[sum]);
      }
      // first time seeing this sum
      else {
        // record index
        mpp[sum] = i;
      }
    }
  }

  // return best length
  return maxi;
}

// program entry
```

Q41. Count subarrays with given xor K

HARD

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Input: A = [4, 2, 2, 6, 4] , k = 6
Output: 4
Explanation: The subarrays having XOR of their elements as 6 are [4, 2], [4, 2, 2, 6, 4], [2, 2, 6], [6]

Input: A = [5, 6, 7, 8, 9], k = 5
Output: 2
Explanation: The subarrays having XOR of their elements as 5 are [5] and [5, 6, 7, 8, 9]

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), we traverse the entire array once to calculate prefix XOR and subarrays with given XOR.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to count subarrays with given XOR
    int countSubarrays(vector<int>& A, int k) {
        // Store frequency of prefix XORs
        unordered_map<int, int> freq;
        // Initialize with prefix XOR 0
        freq[0] = 1;

        // Current prefix XOR
        int prefixXor = 0;
        // Answer count
        int count = 0;

        // Traverse array
        for (int num : A) {
            // Update prefix XOR
            prefixXor ^= num;

            // Compute required XOR
            int target = prefixXor ^ k;

            // If target exists in map, add its frequency
            if (freq.find(target) != freq.end()) {
                count += freq[target];
            }

            // Store current prefix XOR in map
            freq[prefixXor]++;
        }
        return count;
    }
};
```

Q42. Merge Overlapping Subintervals

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Input : intervals=[[1,3],[2,6],[8,10],[15,18]]
Output : [[1,6],[8,10],[15,18]]
Explanation : Since intervals [1,3] and [2,6] are overlapping we can merge them to form [1,6] intervals.

Input : [[1,4],[4,5]]
Output : [[1,5]]
Explanation : Since intervals [1,4] and [4,5] are overlapping we can merge them to form [1,5].

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N*logN) + O(N), we sort the entire array and then merge them in a single pass.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to merge overlapping intervals
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        // Sort intervals based on starting time
        sort(intervals.begin(), intervals.end());

        // Vector to store final merged intervals
        vector<vector<int>> merged;

        // Traverse each interval
        for (auto interval : intervals) {
            // If merged is empty or current interval does not overlap
            if (merged.empty() || merged.back()[1] < interval[0]) {
                // Add current interval as a new non-overlapping block
                merged.push_back(interval);
            } else {
                // Overlapping: merge by extending the end time
                merged.back()[1] = max(
                    merged.back()[1],
                    interval[1]
                );
            }
        }

        return merged;
    }
};
```

Q43. Merge two sorted arrays without extra space

MEDIUM

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Input : nums1 = [-5, -2, 4, 5, 0, 0, 0], nums2 = [-3, 1, 8]
Output : [-5, -3, -2, 1, 4, 5, 8]
Explanation : The merged array is: [-5, -3, -2, 1, 4, 5, 8], where [-5, -2, 4, 5] are from nums1 and [-3, 1, 8] are from nums2

Input : nums1 = [0, 2, 7, 8, 0, 0, 0], nums2 = [-7, -3, -1]
Output : [-7, -3, -1, 0, 2, 7, 8]
Explanation : The merged array is: [-7, -3, -1, 0, 2, 7, 8], where [0, 2, 7, 8] are from nums1 and [-7, -3, -1] are from nums2

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N+M), we traverse both the arrays exactly once. Space Complexit

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Merges nums2 into nums1 in-place in sorted order.
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        // i points to last valid element in nums1
        int i = m - 1;

        // j points to last element in nums2
        int j = n - 1;

        // k is the last index of nums1 (including 0 placeholders)
        int k = m + n - 1;

        // Fill nums1 from the end by comparing nums1[i] and nums2[j]
        while (i >= 0 && j >= 0) {
            // Place larger of the two at nums1[k]
            if (nums1[i] > nums2[j]) {
                nums1[k--] = nums1[i--];
            } else {
                nums1[k--] = nums2[j--];
            }
        }

        // If nums2 has remaining elements, copy them
        while (j >= 0) {
            nums1[k--] = nums2[j--];
        }

        // No need to copy remaining nums1 elements, as they are already in
          place
    }
};
```

Q44. Find the repeating and missing number

HARD

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Example 1:
Input: nums = [3, 5, 4, 1, 1]
Output: [1, 2]
Explanation: 1 appears twice in the array, and 2 is missing from the array. So the output is [1, 2].

Example 2:
Input: nums = [1, 2, 3, 6, 7, 5, 7]
Output: [7, 4]
Explanation: 7 appears twice... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(2*N), where N is the size of the array. This is because we are ite*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find repeating and missing
      numbers
    vector<int>
      findMissingRepeatingNumbers(vector<int>&
      nums) {

        // Size of the array
        int n = nums.size();

         // Hash array to count occurrences
        int hash[n + 1] = {0};

        // Update the hash array:
        for (int i = 0; i < n; i++) {
            hash[nums[i]]++;
        }

        int repeating = -1, missing = -1;

        // Find the repeating and missing
          number:
        for (int i = 1; i <= n; i++) {
            if (hash[i] == 2) {
                repeating = i;
            } else if (hash[i] == 0) {
                missing = i;
            }

            /* If both repeating and missing
            are found, break out of loop*/
            if (repeating != -1 && missing !=
              -1) {
                break;
            }
        }

        // Return {repeating, missing}
        return {repeating, missing};
    }
};
```

[OPTIMAL] Optimal Approach 1

Complexity: *Time Complexity: O(N), where N is the size of the array. This is because we are itera*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find repeating and missing
      numbers
    vector<int>
      findMissingRepeatingNumbers(vector<int>&
      nums) {

        // Size of the array
        long long n = nums.size();

        // Sum of first n natural numbers
        long long SN = (n * (n + 1)) / 2;

        // Sum of squares of first n natural
          numbers
        long long S2N = (n * (n + 1) * (2 * n
          + 1)) / 6;

        /*Calculate actual sum (S) and sum
        of squares (S2) of array elements*/
        long long S = 0, S2 = 0;
        for (int i = 0; i < n; i++) {
            S += nums[i];
            S2 += (long long)nums[i] * (long
              long)nums[i];
        }

        //Compute the difference values
        long long val1 = S - SN;

        // S2 - S2n = X^2 - Y^2
        long long val2 = S2 - S2N;

        //Calculate X + Y using X + Y = (X^2 -
          Y^2) / (X - Y)
        val2 = val2 / val1;

        /* Calculate X and Y from X + Y and X
          - Y
         X = ((X + Y) + (X - Y)) / 2
         Y = X - (X - Y)*/
        long long x = (val1 + val2) / 2;
        long long y = x - val1;

        // Return the results as {repeating,
          missing}
        return {(int)x, (int)y};
    }
};
```

Q45. Count Inversions

HARD

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Example 1:
Input Format: N = 5, array[] = {1,2,3,4,5}
Result: 0
Explanation: we have a sorted array and the sorted array has 0 inversions as for i < j you will never find a pair such that A[j] < A[i]. More clear example: 2 has index 1 and 5 has index 4 now 1 < 5 but 2 < 5 so this is not an inversion.

Example 2:
Input Format: N = 5, array[] = {5,4,3,2,1}
Result: 10
Explanation: we have a reverse sorted array and we will get the maximum inversions as for i < j we will always find a pair such that... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N log N), since it is based on merge sort. Space Complexity: O(N), for the

```cpp
#include <bits/stdc++.h>
using namespace std;
int merge(vector<int> &arr, int low, int mid,
  int high) {
    vector<int> temp;
    int left = low;
    int right = mid + 1;
    int cnt = 0;
    while (left <= mid && right <= high) {
        if (arr[left] <= arr[right]) {
            temp.push_back(arr[left]);
            left++;
        } else {
            temp.push_back(arr[right]);
            cnt += (mid - left + 1);
            right++;
        }
    }
    while (left <= mid) {
        temp.push_back(arr[left]);
        left++;
    }
    while (right <= high) {
        temp.push_back(arr[right]);
        right++;
    }
    for (int i = low; i <= high; i++) {
        arr[i] = temp[i - low];
    }
    return cnt;
}
int mergeSort(vector<int> &arr, int low, int
  high) {
    int cnt = 0;
    if (low >= high) return cnt;
    int mid = (low + high) / 2;
    cnt += mergeSort(arr, low, mid);
    cnt += mergeSort(arr, mid + 1, high);
    cnt += merge(arr, low, mid, high);
    return cnt;
}
int numberOfInversions(vector<int>& a, int n)
  {
    return mergeSort(a, 0, n - 1);
}
```

Q46. Reverse Pairs

HARD

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Example 1:
Input: N = 5, array[] = {1,3,2,3,1)
Output: 2
Explanation: The pairs are (3, 1) and (3, 1) as from both the pairs the condition arr[i] > 2*arr[j] is satisfied.

Example 2:
Input: N = 4, array[] = {3,2,1,4}
Output: 1
Explaination: There is only 1 pair (3 , 1) that satisfy the condition arr[i] > 2*arr[j]

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(2N*logN), Inside the mergeSort() we call merge() and countPairs() except mergeSort() itself

```cpp
#include <bits/stdc++.h>
using namespace std;
void merge(vector<int> &arr, int low, int mid,
  int high) {
    vector<int> temp; // temporary array
    int left = low;      // starting index of
      left half of arr
    int right = mid + 1;   // starting index
      of right half of arr
    //storing elements in the temporary array
      in a sorted manner//
    while (left <= mid && right <= high) {
        if (arr[left] <= arr[right]) {
            temp.push_back(arr[left]);
            left++;
        }
        else {
            temp.push_back(arr[right]);
            right++;
        }
    }
    // if elements on the left half are still
      left //
    while (left <= mid) {
        temp.push_back(arr[left]);
        left++;
    }
    //  if elements on the right half are
      still left //
    while (right <= high) {
        temp.push_back(arr[right]);
        right++;
    }
    // transfering all elements from temporary
      to arr //
    for (int i = low; i <= high; i++) {
        arr[i] = temp[i - low];
    }
}
int countPairs(vector<int> &arr, int low, int
  mid, int high) {
    int right = mid + 1;
    int cnt = 0;
    for (int i = low; i <= mid; i++) {
        while (right <= high && arr[i] > 2 *
          arr[right]) right++;
        cnt += (right - (mid + 1));
    }
    return cnt;
}
int mergeSort(vector<int> &arr, int low, int
  high) {
    int cnt = 0;
    if (low >= high) return cnt;
    int mid = (low + high) / 2 ;
    cnt += mergeSort(arr, low, mid);  // left
      half
    cnt += mergeSort(arr, mid + 1, high); //
      right half
    cnt += countPairs(arr, low, mid, high);
      //Modification
    merge(arr, low, mid, high);  // merging
      sorted halves
    return cnt;
}
int team(vector <int> & skill, int n)
{
    return mergeSort(skill, 0, n - 1);
}
```

Q47. Maximum Product Subarray in an Array

HARD

Topic: Solve Problems on Arrays [Easy -> Medium -> Hard] > Hard

Practice Link

Input: Nums = [1,2,3,4,5,0]
Output: 120
Explanation:
In the given array, 1×2×3×4×5 gives maximum product value.

Input: Nums = [1,2,-3,0,-4,-5]
Output: 20
Explanation:
In the given array, (-4)×(-5) gives maximum product value.

[OPTIMAL SOLUTION] Optimal Approach - 1 — Time Complexity: O(N), every element of array is visited once. Space Complexity: O(1), constant number of

```cpp
#include <bits/stdc++.h>
using namespace std;

// This function returns the maximum product
  subarray
// using prefix and suffix traversal
class Solution {
public:
    int maxProductSubArray(vector<int>& arr) {
        // Store size of array
        int n = arr.size();

        // Initialize prefix and suffix
          product
        int pre = 1, suff = 1;

        // Initialize answer to negative
          infinity
        int ans = INT_MIN;

        // Traverse from both left and right
        for (int i = 0; i < n; i++) {
            // Reset prefix if zero
            if (pre == 0) pre = 1;

            // Reset suffix if zero
            if (suff == 0) suff = 1;

            // Multiply prefix with current
              element from front
            pre *= arr[i];

            // Multiply suffix with current
              element from back
            suff *= arr[n - i - 1];

            // Update the maximum of all
              products seen so far
            ans = max(ans, max(pre, suff));
        }

        // Return the final answer
        return ans;
    }
};
```

Q48. Search X in sorted array

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Refer to the code implementation and logic below.

[BETTER] Recursive Approach

Complexity: *Time Complexity: In the algorithm, in every step, we are basically dividing the searc*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Recursive Binary Search function
    int binarySearch(vector& nums, int low,
      int high, int target) {
        if (low > high) return -1; // Base
          case: target not found

        // Find middle index
        int mid = (low + high) / 2;

        // If target is found at mid
        if (nums[mid] == target) return mid;
        // If target is greater, search right
          half
        else if (target > nums[mid])
            return binarySearch(nums, mid + 1,
              high, target);
        // Otherwise, search left half
        return binarySearch(nums, low, mid -
          1, target);
    }

    // Public function to initiate search
    int search(vector& nums, int target) {
        return binarySearch(nums, 0,
          nums.size() - 1, target);
    }
};
```

[OPTIMAL] General

Complexity: *Time Complexity: In the algorithm, in every step, we are basically dividing the searc*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to perform Binary Search on
      sorted array
    int binarySearch(vector& nums, int target)
      {
        int n = nums.size(); // size of the
          array
        int low = 0, high = n - 1;

        // Keep searching until low crosses
          high
        while (low <= high) {
            int mid = (low + high) / 2; //
              Find the middle index
            if (nums[mid] == target) return
              mid;       // Target found
            else if (target > nums[mid]) low =
              mid + 1; // Search in right half
            else high = mid - 1;
              // Search in left half
        }
        return -1; // Target not found
    }
};
```

Q49. Lower Bound

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Example 1:
Input Format: N = 4, arr[] = {1,2,2,3}, x = 2
Result: 1
Explanation: Index 1 is the smallest index such that arr[1] >= x.

Example 2:
Input Format: N = 5, arr[] = {3,5,8,15,19}, x = 9
Result: 3
Explanation: Index 3 is the smallest index such that arr[3] >= x.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(logn), used for typical binary search Space Complexity: O(1), no extra

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to find the lower bound index in an array
class LowerBoundFinder {
public:
    // Function to find the lower bound using binary search
    int lowerBound(vector<int> arr, int n, int x) {
        int low = 0;           // Start of search range
        int high = n - 1;      // End of search range
        int ans = n;           // Default to n (not found)

        // Binary search loop
        while (low <= high) {
            int mid = (low + high) / 2;  // Middle index

            if (arr[mid] >= x) {
                ans = mid;           // Store possible answer
                high = mid - 1;      // Try to find smaller index on left side
            } else {
                low = mid + 1;       // Move right if current element is less
                  than x
            }
        }
        return ans;  // Return the index of the lower bound
    }
};
```

Q50. Upper Bound

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Example 1:
Input Format: N = 4, arr[] = {1,2,2,3}, x = 2
Result: 3
Explanation: Index 3 is the smallest index such that arr[3] > x.

Example 2:
Input Format: N = 6, arr[] = {3,5,8,9,15,19}, x = 9
Result: 4
Explanation: Index 4 is the smallest index such that arr[4] > x.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(logn), used for typical binary search Space Complexity: O(1), no extra

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to compute upper bound
class UpperBoundFinder {
public:
    // Function to find the upper bound using binary search
    int upperBound(vector<int> &arr, int x, int n) {
        int low = 0, high = n - 1;
        int ans = n;  // Default answer if x is >= all elements

        while (low <= high) {
            int mid = (low + high) / 2;

            if (arr[mid] > x) {
                ans = mid;        // Potential answer found
                high = mid - 1;   // Try to find smaller valid index on the
                  left
            } else {
                low = mid + 1;    // Move right if mid is <= x
            }
        }
        return ans;  // Index of the first element > x
    }
};
```

Q51. Search insert position

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Example 1:
Input Format: arr[] = {1,2,4,7}, x = 6
Result: 3
Explanation: 6 is not present in the array. So, if we will insert 6 in the 3rd index(0-based indexing), the array will still be sorted. {1,2,4,6,7}.

Example 2:
Input Format: arr[] = {1,2,4,7}, x = 2
Result: 1
Explanation: 2 is present in the array and so we will return its index i.e. 1.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(logN), where N = size of the given array. Space Complexity: O(1) as we are using no extra s

```java
import java.util.*;

public class BinarySearchInsert {

    // Function to find the insert position of x in sorted array
    public int searchInsert(int[] arr, int x) {
        int n = arr.length;
        int low = 0, high = n - 1;
        int ans = n; // Default to end if x is greater than all elements

        while (low <= high) {
            int mid = (low + high) / 2;

            if (arr[mid] >= x) {
                // Potential answer found, try to go left
                ans = mid;
                high = mid - 1;
            } else {
                // Go right
                low = mid + 1;
            }
        }

        return ans;
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 7};
        int x = 6;

        BinarySearchInsert obj = new BinarySearchInsert();
        int index = obj.searchInsert(arr, x);

        System.out.println("The index is: " + index);
    }
}
```

Q52. Floor and Ceil in Sorted Array

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Example 1:
Input Format: n = 6, arr[] ={3, 4, 4, 7, 8, 10}, x= 5
Result: 4 7
Explanation: The floor of 5 in the array is 4, and the ceiling of 5 in the array is 7.

Example 2:
Input Format: n = 6, arr[] ={3, 4, 4, 7, 8, 10}, x= 8
Result: 8 8
Explanation: The floor of 8 in the array is 8, and the ceiling of 8 in the array is also 8.

[OPTIMAL SOLUTION] Approach — Time Complexity:O(logN), where N = size of the given array. We are using the Binary Search algorithm Space Com

```cpp
#include <bits/stdc++.h>
using namespace std;
class FloorCeilFinder {
public:
    // Function to find the floor of x
    int findFloor(int arr[], int n, int x) {
        int low = 0, high = n - 1;
        int ans = -1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] <= x) {
                ans = arr[mid];     //
                  Potential floor
                low = mid + 1;      // Search
                  right side
            } else {
                high = mid - 1;     // Search
                  left side
            }
        }
        return ans;
    }
    // Function to find the ceiling of x
    int findCeil(int arr[], int n, int x) {
        int low = 0, high = n - 1;
        int ans = -1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] >= x) {
                ans = arr[mid];     //
                  Potential ceil
                high = mid - 1;     // Search
                  left side
            } else {
                low = mid + 1;      // Search
                  right side
            }
        }
        return ans;
    }
    // Function to return both floor and ceil
    pair<int, int> getFloorAndCeil(int arr[],
      int n, int x) {
        int f = findFloor(arr, n, x);
        int c = findCeil(arr, n, x);
        return make_pair(f, c);
    }
};
```

Q53. First and last occurrence

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Example 1:
Input: N = 7, target = 13, array[] = {3, 4, 13, 13, 13, 20, 40}
Output: 4
Explanation: The target value 13 appears for the first time at index number 2 in the array.

Example 2:
Input: N = 7, target = 60, array[] = {3, 4, 13, 13, 13, 20, 40}
Output: -1
Explanation: Target value 60 is not present in the array, so the output is -1.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(log N), where N is the size of the array. This is because we are using binary search, which

```cpp
#include <bits/stdc++.h>
using namespace std;

// find last index of key using binary search
int solve(int n, int key, vector<int>& v) {
  // initialize search bounds and result
  int start = 0;
  int end = n - 1;
  int res = -1;

  // binary search loop
  while (start <= end) {
    // compute mid safely
    int mid = start + (end - start) / 2;
    // when match found, store index and move right
    if (v[mid] == key) {
      res = mid;
      start = mid + 1;
    }
    // when key is smaller, move left
    else if (key < v[mid]) {
      end = mid - 1;
    }
    // otherwise move right
    else {
      start = mid + 1;
    }
  }
  // return last occurrence or -1
  return res;
}

// program entry
```

Q54. Count Occurrences in a Sorted Array

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Example 1:
Input: N = 7, X = 3 , array[] = {2, 2 , 3 , 3 , 3 , 3 , 4}
Output: 4
Explanation: 3 is occurring 4 times in the given array so it is our answer.

Example 2:
Input: N = 8, X = 2 , array[] = {1, 1, 2, 2, 2, 2, 2, 3}
Output: 5
Explanation: 2 is occurring 5 times in the given array so it is our answer.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(2*logN), We are basically using the binary search algorithm twice. Space Complexity: O(1),

```cpp
#include <bits/stdc++.h>
using namespace std;
int firstOccurrence(vector<int> &arr, int n,
  int k) {
    int low = 0, high = n - 1;
    int first = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        // maybe an answer
        if (arr[mid] == k) {
            first = mid;
            //look for smaller index on the
              left
            high = mid - 1;
        }
        else if (arr[mid] < k) {
            low = mid + 1; // look on the
              right
        }
        else {
            high = mid - 1; // look on the
              left
        }
    }
    return first;
}
int lastOccurrence(vector<int> &arr, int n,
  int k) {
    int low = 0, high = n - 1;
    int last = -1;
    while (low <= high) {
        int mid = (low + high) / 2;
        // maybe an answer
        if (arr[mid] == k) {
            last = mid;
            //look for larger index on the
              right
            low = mid + 1;
        }
        else if (arr[mid] < k) {
            low = mid + 1; // look on the
              right
        }
        else {
            high = mid - 1; // look on the
              left
        }
    }
    return last;
}
pair<int, int>
  firstAndLastPosition(vector<int>& arr, int
  n, int k) {
    int first = firstOccurrence(arr, n, k);
    if (first == -1) return { -1, -1};
    int last = lastOccurrence(arr, n, k);
    return {first, last};
}
int count(vector<int>& arr, int n, int x) {
    pair<int, int> ans =
      firstAndLastPosition(arr, n, x);
    if (ans.first == -1) return 0;
    return (ans.second - ans.first + 1);
}
```

Q55. Search in rotated sorted array-I

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Input:nums = [4, 5, 6, 7, 0, 1, 2], k = 0
Output :4
Explanation : Here, the target is 0. We can see that 0 is present in the given rotated sorted array, nums. Thus, we get output as 4, which is the index at which 0 is present in the array.
Input: nums = [4, 5, 6, 7, 0, 1, 2], k = 3
Output :-1
Explanation :Here, the target is 3. Since 3 is not present in the given rotated sorted array. Thus, we get the output as -1.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(log N),We eliminate half of the search space in each iteration using binary search. Space C

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to search for target using
      binary search in rotated sorted array
    int search(vector<int>& nums, int target)
      {
        // Set the search space to entire
          array
        int low = 0;
        int high = nums.size() - 1;
        // Continue until the search space
          becomes invalid
        while (low <= high) {
            // Find the middle index
            int mid = (low + high) / 2;
            // If the target is found at mid,
              return mid
            if (nums[mid] == target)
                return mid;
            // Check if the left half is
              sorted
            if (nums[low] <= nums[mid]) {
                // If target lies in the
                  sorted left half, search
                  there
                if (nums[low] <= target &&
                  target < nums[mid]) {
                    high = mid - 1;
                }
                // Else search in the right
                  half
                else {
                    low = mid + 1;
                }
            }
            // Otherwise, right half is sorted
            else {
                // If target lies in the
                  sorted right half, search
                  there
                if (nums[mid] < target &&
                  target <= nums[high]) {
                    low = mid + 1;
                }
                // Else search in the left
                  half
                else {
                    high = mid - 1;
                }
            }
        }
        // If not found, return -1
        return -1;
    }
};
// Driver code
```

Q56. Search in rotated sorted array-II

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Example 1:
Input Format: arr = [7, 8, 1, 2, 3, 3, 3, 4, 5, 6], k = 3
Result: True
Explanation: The element 3 is present in the array. So, the answer is True.

Example 2:
Input Format: arr = [7, 8, 1, 2, 3, 3, 3, 4, 5, 6], k = 10
Result: False
Explanation: The element 10 is not present in the array. So, the answer is False.

[OPTIMAL SOLUTION] Optimised Approach — Time Complexity: O(logN) for the best and average case. O(N/2) for the worst case. Here, N = size of the given

```cpp
#include <bits/stdc++.h>
using namespace std;

bool
  searchInARotatedSortedArrayII(vector<int>&
  arr, int k) {
    int n = arr.size();
    int low = 0, high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        // If mid points to the target
        if (arr[mid] == k) return true;

        // Edge case: all three are equal, we
          cannot determine which side is
          sorted
        if (arr[low] == arr[mid] && arr[mid]
          == arr[high]) {
            low++;
            high--;
            continue;
        }

        // If the left half is sorted
        if (arr[low] <= arr[mid]) {
            if (arr[low] <= k && k <=
              arr[mid]) {
                high = mid - 1; // Target lies
                  in left
            } else {
                low = mid + 1;  // Target lies
                  in right
            }
        } else {
            // Right half is sorted
            if (arr[mid] <= k && k <=
              arr[high]) {
                low = mid + 1;  // Target lies
                  in right
            } else {
                high = mid - 1; // Target lies
                  in left
            }
        }
    }

    return false;
}
```

Q57. Find minimum in Rotated Sorted Array

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Input: arr = [4,5,6,7,0,1,2,3]
Output: 0
Explanation: The minimum element in the array is 0.

Input : arr = [3,4,5,1,2]
Output: 1
Explanation : The minimum element in the array is 1.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(logN), at every step the search space is reduced to half using binary search.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the minimum element using binary search
    int findMin(vector<int>& nums) {

        // Initialize low and high pointers
        int low = 0, high = nums.size() - 1;

        // Binary search loop
        while (low < high) {

            // Calculate mid index
            int mid = low + (high - low) / 2;

            // Check which half to discard
            if (nums[mid] > nums[high]) {

                // Minimum lies in right half
                low = mid + 1;

            } else {

                // Minimum lies in left half (including mid)
                high = mid;
            }
        }

        // Return the minimum element
        return nums[low];
    }
};
```

Q58. Find out how many times the array is rotated

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Input : arr = [4,5,6,7,0,1,2,3]
Result: 4
Explanation: The original array should be [0,1,2,3,4,5,6,7]. So, we can notice that the array has been rotated 4 times.
Input : arr = [3,4,5,1,2]
Output : 3
Explanation: The original array should be [1,2,3,4,5]. So, we... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(n),We traverse the array once to find the rotation point, where n*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to find rotation count using
  one-pass scan
int findRotationCount(vector<int> &arr) {
    // Get the size of the array
    int n = arr.size();
    // Traverse the array till second-last
      element
    for (int i = 0; i < n - 1; i++) {
        // If current element is greater than
          the next, break point found
        if (arr[i] > arr[i + 1]) {
            // Rotation count is index of next
              element
            return i + 1;
        }
    }
    // If no break point found, array not
      rotated
    return 0;
}
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(log n),The binary search halves the search space each iteration, w*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find rotation count using
      binary search
    int findRotations(vector<int>& arr) {
        // Initialize low and high pointers
        int low = 0;
        int high = arr.size() - 1;

        // Loop until low meets high
        while (low < high) {
            // Find mid index
            int mid = low + (high - low) / 2;

            // If mid element is greater than
              element at high,
            // smallest element lies to the
              right of mid
            if (arr[mid] > arr[high]) {
                low = mid + 1;
            } else {
                // Else smallest element is at
                  mid or to the left
                high = mid;
            }
        }

        // When low == high, we found the
          smallest element
        return low;
    }
};
```

Q59. Single element in a Sorted Array

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Input : arr[] = {1,1,2,2,3,3,4,5,5,6,6}
Output: 4
Explanation: Only the number 4 appears once in the array.
Input: arr[] = {1,1,3,5,5}
Output : 3
Explanation: Only the number 3 appears once in the array.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(logN), N = size of the given array ,as we are basically using the Binary Search algorithm.

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
// Function to find the single non-duplicate
  element using binary search
int singleNonDuplicate(vector<int>& arr) {
    // Get the size of the array
    int n = arr.size();
    // Edge case: only one element in the
      array
    if (n == 1) return arr[0];
    // Edge case: first element is the unique
      one
    if (arr[0] != arr[1]) return arr[0];
    // Edge case: last element is the unique
      one
    if (arr[n - 1] != arr[n - 2]) return arr[n
      - 1];
    // Initialize binary search bounds
      (exclude first and last index)
    int low = 1, high = n - 2;
    // Perform binary search
    while (low <= high) {
        // Calculate middle index
        int mid = (low + high) / 2;
        // Check if middle element is the
          unique one
        if (arr[mid] != arr[mid + 1] &&
          arr[mid] != arr[mid - 1]) {
            return arr[mid];
        }
        // If mid is in the left half (pairing
          is valid)
        if ((mid % 2 == 1 && arr[mid] ==
          arr[mid - 1]) ||
            (mid % 2 == 0 && arr[mid] ==
              arr[mid + 1])) {
            // Move to the right half
            low = mid + 1;
        }
        // If mid is in the right half
          (pairing broken earlier)
        else {
            // Move to the left half
            high = mid - 1;
        }
    }
    // Dummy return (not reachable if input is
      valid)
    return -1;
   }
};
// Driver code
```

Q60. Find peak element

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 1D Arrays

Practice Link

Input: arr[] = {1,2,3,4,5,6,7,8,5,1}
Output: 7
Explanation: There is only 1 peak element, 8, that is at index 7.

Input: arr[] = {1,2,1,3,5,6,4}
Output: 1
Explanation : There are 2 peak numbers that are at indices 1 and 5. We can return any of them.

[OPTIMAL SOLUTION] Optimal Approach

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find a peak element using binary search
    int findPeakElement(vector<int>& nums) {
        // Set left and right bounds
        int low = 0, high = nums.size() - 1;

        // Binary search loop
        while (low < high) {
            // Find mid point
            int mid = (low + high) / 2;

            // If mid element is greater than next
            if (nums[mid] > nums[mid + 1]) {
                // Move to left half
                high = mid;
            } else {
                // Move to right half
                low = mid + 1;
            }
        }

        // Return peak index
        return low;
    }
};
```

Q61. Find square root of a number

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Input: N = 36
Output: 6
Explanation: Square root of 36 is 6.

Input: N = 28
Output: 5
Explanation: Square root of 28 is approximately 5.292. So, the floor value will be 5.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(log(N)), we apply binary search on our search space to reduce it into half at every step.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // This function returns the floor value of the square root of a number
    int mySqrt(int x) {
        // Handle small numbers directly
        if (x < 2) return x;

        // Initialize binary search range
        int left = 1, right = x / 2, ans = 0;

        // Perform binary search
        while (left <= right) {
            // Find middle point
            long long mid = left + (right - left) / 2;

            // Check if mid*mid is less than or equal to x
            if (mid * mid <= x) {
                // Store mid as potential answer
                ans = mid;
                // Move to right half
                left = mid + 1;
            } else {
                // Move to left half
                right = mid - 1;
            }
        }

        // Return final answer
        return ans;
    }
};
```

Q62. Find Nth root of a number

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Input: N = 3, M = 27
Output: 3
Explanation: The cube root of 27 is equal to 3.

Input : N = 4, M = 69
Output: -1
Explanation : The 4th root of 69 does not exist. So, the answer is -1.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(logM), we search for every possible number from 1 to M to check if it is the Nth root.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find N-th root of M using binary search
    int nthRoot(int n, int m) {
        // Set low and high for binary search
        int low = 1, high = m;

        // Start binary search
        while (low <= high) {
            // Calculate mid
            int mid = (low + high) / 2;

            // Store result of mid^n
            long long ans = 1;
            for (int i = 0; i < n; i++) {
                ans *= mid;
                if (ans > m) break;
            }

            // If mid^n equals m
            if (ans == m) return mid;

            // If mid^n is less than m
            if (ans < m) low = mid + 1;

            // If mid^n is more than m
            else high = mid - 1;
        }

        // Return -1 if not found
        return -1;
    }
};

// Main function
```

Q63. Koko eating bananas

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Input: N = 4, a[] = {7, 15, 6, 3}, h = 8
Output: 5
Explanation: If Koko eats 5 bananas/hr, he will take 2, 3, 2, and 1 hour to eat the piles accordingly. So, he will take 8 hours to complete all the piles.

Input: N = 5, a[] = {25, 12, 8, 14, 19}, h = 5
Output: 25
Explanation: If Koko eats 25 bananas/hr, he will take 1, 1, 1, 1, and 1 hour to eat the piles accordingly.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N*log(max(a[]))), we apply binary search on our search space to reduce it into half at ever

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to calculate total hours at
      given speed
    int calculateTotalHours(vector<int>&
      piles, int speed) {
        int totalH = 0;
        for (int bananas : piles) {
            totalH += ceil((double)bananas /
              speed);
        }
        return totalH;
    }

    // Function to find minimum eating speed
    int minEatingSpeed(vector<int>& piles, int
      h) {
        // Find maximum element
        int maxPile =
          *max_element(piles.begin(),
          piles.end());

        // Initialize low and high pointers
        int low = 1, high = maxPile;
        int ans = maxPile;

        // Binary search on answer space
        while (low <= high) {
            int mid = (low + high) / 2;
            int totalH =
              calculateTotalHours(piles, mid);

            // If possible, try smaller speed
            if (totalH <= h) {
                ans = mid;
                high = mid - 1;
            }
            // Otherwise, try larger speed
            else {
                low = mid + 1;
            }
        }
        return ans;
    }
};
```

Q64. Minimum days to make M bouquets

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Example 1:
Input Format: N = 8, arr[] = {7, 7, 7, 7, 13, 11, 12, 7}, m = 2, k = 3
Result: 12
Explanation: On the 12th the first 4 flowers and the last 3 flowers would have already bloomed. So, we can easily make 2 bouquets, one with the first 3 and another with the last 3 flowers.

Example 2:
Input Format: N = 5, arr[] = {1, 10, 3, 10, 2}, m = 3, k = 2
Result: -1
Explanation: If we want to make 3 bouquets of 2 flowers each, we need at least 6 flowers. But we are given only 5 flowers, so, we cann... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(1) O(log(max(arr[])-min(arr[])+1) * N), where {max(arr[]) -> maximum element of the array,

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Helper function to check if it's
      possible to make m bouquets on or before
      a given day
    bool possible(vector<int>& arr, int day,
      int m, int k) {
        int n = arr.size();         // Total
          number of flowers
        int cnt = 0;                // Counter
          for consecutive bloomed flowers
        int bouquets = 0;           // Count
          of bouquets made
        for (int i = 0; i < n; i++) {
            if (arr[i] <= day) {
                // Flower bloomed, increment
                  consecutive count
                cnt++;
                if (cnt == k) {
                    // We have k consecutive
                      bloomed flowers — make 1
                      bouquet
                    bouquets++;
                    cnt = 0; // reset for next
                      bouquet
                }
            } else {
                // Flower not bloomed, reset
                  consecutive count
                cnt = 0;
            }
        }
        // Check if at least m bouquets can be
          made
        return bouquets >= m;
    }
    // Main function to find the minimum day
      to make m bouquets
    int roseGarden(vector<int>& arr, int k,
      int m) {
        long long total = 1LL * k * m; //
          Total flowers required
        // If total required flowers >
          available flowers, it's impossible
        if (total > arr.size()) return -1;
        // Find minimum and maximum bloom days
          from array
        int mini = *min_element(arr.begin(),
          arr.end());
        int maxi = *max_element(arr.begin(),
          arr.end());
        // Apply binary search on number of
          days
        int low = mini, high = maxi;
        int result = -1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (possible(arr, mid, m, k)) {
                // If it's possible to make
                  bouquets on this day, try to
                  find an earlier day
                result = mid;
                high = mid - 1;
            } else {
                // Otherwise, try with a later
                  day
                low = mid + 1;
            }
        }
        return result;
    }
};
```

Q65. Find the smallest divisor

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Example 1:
Input Format: N = 5, arr[] = {1,2,3,4,5}, limit = 8
Result: 3
Explanation: We can get a sum of 15(1 + 2 + 3 + 4 + 5) if we choose 1 as a divisor.
The sum is 9(1 + 1 + 2 + 2 + 3) if we choose 2 as a divisor. Upon dividing all the elements of the array by 3, we get 1,1,1,2,2 respectively. Now, their sum is equal to 7 <= 8 i.e. the threshold value. So, 3 is the minimum possible answer.

Example 2:
Input Format: N = 4, arr[] = {8,4,2,3}, limit = 10
Result: 2
Explanation: If we choose 1,... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity:O(log(max(arr[]))*N), where max(arr[]) = maximum element in the array, N = size of the array.

```cpp
#include <bits/stdc++.h>
using namespace std;

class SmallestDivisorFinder {
public:
    // Helper function to calculate sum of division results
    int sumByD(vector<int>& arr, int div) {
        int sum = 0;
        for (int num : arr) {
            sum += ceil((double)num / div);
        }
        return sum;
    }

    // Function to find the smallest divisor
    int smallestDivisor(vector<int>& arr, int limit) {
        if (arr.size() > limit) return -1;

        int low = 1;
        int high = *max_element(arr.begin(), arr.end());

        // Binary search to find smallest divisor
        while (low <= high) {
            int mid = (low + high) / 2;
            if (sumByD(arr, mid) <= limit) {
                high = mid - 1;  // Try smaller divisor
            } else {
                low = mid + 1;   // Try larger divisor
            }
        }

        return low;
    }
};
```

Q66. Capacity to Ship Packages Within D Days

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Input: N = 5, weights = [5, 4, 5, 2, 3, 4, 5, 6], d = 5
Output: 9
Explanation: The minimum ship capacity needed to ship all packages within 5 days is 9.
Input: N = 3, weights = [1, 2, 3, 4, 5], d = 2
Output: 9
Explanation: With a capacity of 9, the packages can be shipped in 2 days as [1,2,3,4] and [5].

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N * log(S)), where N is number of packages and S is the search space (sum_weights - max_wei

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to calculate how many days are
      needed to ship packages with given
      capacity
    int daysNeeded(vector<int>& weights, int
      capacity) {
        // Initialize count of days to 1
          (start with day one)
        int days = 1;
        // Variable to track the current load
          on the ship for a day
        int currentLoad = 0;
        // Loop through each package weight in
          the array
        for (int w : weights) {
            // Check if adding current package
              exceeds ship capacity for the
              day
            if (currentLoad + w > capacity) {
                // Need a new day to ship this
                  package, so increment day
                  count
                days++;
                // Start new day's load with
                  current package weight
                currentLoad = w;
            } else {
                // If capacity not exceeded,
                  add current package weight
                  to the day's load
                currentLoad += w;
            }
        }
        // Return total number of days needed
          with this capacity
        return days;
    }

    // Main function to find minimum capacity
      to ship all packages within d days
    int shipWithinDays(vector<int>& weights,
      int d) {
        // Find the maximum package weight as
          the minimum capacity (can't be less
          than this)
        int left =
          *max_element(weights.begin(),
          weights.end());
        // Calculate the sum of all package
          weights as maximum capacity (ship
          all at once)
        int right =
          accumulate(weights.begin(),
          weights.end(), 0);
        // Binary search loop to find optimal
          capacity between left and right
        while (left < right) {
            // Calculate mid value as
              potential capacity to test
            int mid = left + (right - left) /
              2;
            // Calculate how many days needed
              if ship capacity is mid
            int needed = daysNeeded(weights,
              mid);
            // If days needed is less or equal
              to allowed days,
            // try to reduce capacity by
              moving right boundary
            if (needed <= d) {
                right = mid;
            } else {
                // If days needed is more than
                  allowed, increase capacity
                  by moving left boundary
                left = mid + 1;
            }
        }
        // Left is now the minimum capacity
          that can ship packages within d days
        return left;
    }
};
```

Q67. Kth Missing Positive Number

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Example 1:
Input Format: vec[]={4,7,9,10}, k = 1
Result: 1
Explanation: The missing numbers are 1, 2, 3, 5, 6, 8, 11, 12, ……, and so on. Since 'k' is 1, the first missing element is 1.

Example 2:
Input Format: vec[]={4,7,9,10}, k = 4
Result: 5
Explanation: The missing numbers are 1, 2, 3, 5, 6, 8, 11, 12, ……, and so on. Since 'k' is 4, the fourth missing element is 5.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(logn), used for typical binary search Space Complexity: O(1), no extra

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to find the k-th missing number using binary search
class MissingKFinder {
public:
    // Function to return the k-th missing number
    int missingK(vector<int> vec, int n, int k) {
        int low = 0, high = n - 1;

        // Perform binary search
        while (low <= high) {
            int mid = (low + high) / 2;

            // Calculate how many numbers are missing till vec[mid]
            int missing = vec[mid] - (mid + 1);

            if (missing < k) {
                low = mid + 1;  // Move right to find more missing numbers
            } else {
                high = mid - 1; // Move left to find a smaller valid index
            }
        }

        // After loop, 'high' points to the largest index such that
        // number of missing elements till there < k
        return k + high + 1;
    }
};
```

Q68. Aggressive Cows

HARD

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Example 1:
Input Format: N = 6, k = 4, arr[] = {0,3,4,7,10,9}
Result: 3
Explanation: The maximum possible minimum distance between any two cows will be 3 when 4 cows are placed at positions {0, 3, 7, 10}. Here the distances between cows are 3, 4, and 3 respectively. We cannot make the minimum distance greater than 3 in any ways.

Example 2:
Input Format: N = 5, k = 2, arr[] = {4,2,1,3,6}
Result: 5
Explanation: The maximum possible minimum distance between any two cows will be 5 when 2 cows are p... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(NlogN) + O(N * log(max(stalls[])-min(stalls[]))), where N = size of the array, max(stalls[]

```cpp
#include <bits/stdc++.h>
using namespace std;
// Class to solve Aggressive Cows problem
class Solution {
public:
    // Function to check if we can place all
      cows with at least 'd' distance
    bool canPlace(vector<int>& stalls, int
      cows, int d) {
        // Place first cow at the first stall
        int count = 1;
        int lastPos = stalls[0];
        // Loop through all stalls
        for (int i = 1; i < stalls.size();
          i++) {
            // Check if this stall is at least
              'd' away from last placed cow
            if (stalls[i] - lastPos >= d) {
                // Place the cow here
                count++;
                // Update last placed cow
                  position
                lastPos = stalls[i];
            }
            // If all cows are placed
              successfully
            if (count >= cows) return true;
        }
        // Return false if we could not place
          all cows
        return false;
    }
    // Function to maximize minimum distance
      between cows
    int aggressiveCows(vector<int>& stalls,
      int cows) {
        // Sort the stalls
        sort(stalls.begin(), stalls.end());
        // Define search space
        int low = 1;
        int high = stalls.back() -
          stalls.front();
        int ans = 0;
        // Apply binary search
        while (low <= high) {
            // Find mid distance
            int mid = low + (high - low) / 2;
            // If placing cows is possible
              with mid distance
            if (canPlace(stalls, cows, mid)) {
                // Store this as potential
                  answer
                ans = mid;
                // Try to find larger minimum
                  distance
                low = mid + 1;
            }
            else {
                // Otherwise try smaller
                  distance
                high = mid - 1;
            }
        }
        // Return the largest minimum distance
        return ans;
    }
};
// Driver function
```

Q69. Book Allocation Problem

HARD

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Example 1:
Input Format: n = 4, m = 2, arr[] = {12, 34, 67, 90}
Result: 113
Explanation: The allocation of books will be 12, 34, 67 | 90. One student will get the first 3 books and the other will get the last one.

Example 2:
Input Format: n = 5, m = 4, arr[] = {25, 46, 28, 49, 24}
Result: 71
Explanation: The allocation of books will be 25, 46 | 28 | 49 | 24.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N * log(sum(arr[])-max(arr[])+1)), where N = size of the array, sum(arr[]) = sum of all arr

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution{
int countStudents(vector<int> &arr, int pages)
  {
    int n = arr.size(); //size of array.
    int students = 1;
    long long pagesStudent = 0;
    for (int i = 0; i < n; i++) {
        if (pagesStudent + arr[i] <= pages) {
            //add pages to current student
            pagesStudent += arr[i];
        }
        else {
            //add pages to next student
            students++;
            pagesStudent = arr[i];
        }
    }
    return students;
}
```

```cpp
int findPages(vector<int>& arr, int n, int m)
  {
    //book allocation impossible:
    if (m > n) return -1;

    int low = *max_element(arr.begin(),
      arr.end());
    int high = accumulate(arr.begin(),
      arr.end(), 0);
    while (low <= high) {
        int mid = (low + high) / 2;
        int students = countStudents(arr,
          mid);
        if (students > m) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    return low;
}
```

Q70. Split array - largest sum

HARD

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Example 1:
Input Format: N = 5, a[] = {1,2,3,4,5}, k = 3
Result: 6
Explanation: There are many ways to split the array a[] into k consecutive subarrays. The best way to do this is to split the array a[] into [1, 2, 3], [4], and [5], where the largest sum among the three subarrays is only 6.

Example 2:
Input Format: N = 3, a[] = {3,5,1}, k = 3
Result: 5
Explanation: There is only one way to split the array a[] into 3 subarrays, i.e., [3], [5], and [1]. The largest sum among these subarrays is 5.

[OPTIMAL SOLUTION] Optimised Approach — Time Complexity: O(N * log(sum(arr[])-max(arr[])+1)), where N = size of the array, sum(arr[]) = sum of all arr

```java
import java.util.*;

class SubarrayPartitioner {
    // Counts how many partitions are needed
      for a given maxSum
    public int countPartitions(int[] a, int
      maxSum) {
        int partitions = 1; // at least one
          partition
        long subarraySum = 0; // sum of
          current subarray

        for (int num : a) {
            if (subarraySum + num <= maxSum) {
                subarraySum += num;
            } else {
                partitions++;
                subarraySum = num;
            }
        }
        return partitions;
    }

    // Finds the minimum largest subarray sum
      possible for at most k partitions
    public int
      largestSubarraySumMinimized(int[] a, int
      k) {
        int low =
          Arrays.stream(a).max().getAsInt();
          // largest element
        int high = Arrays.stream(a).sum(); //
          sum of all elements
        // Binary search for the smallest
          maxSum
        while (low <= high) {
            int mid = (low + high) / 2;
            int partitions =
              countPartitions(a, mid);

            if (partitions > k) {
                low = mid + 1; // too many
                  partitions → increase maxSum
            } else {
                high = mid - 1; // valid but
                  try smaller maxSum
            }
        }
        return low;
    }

    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int k = 2;
        SubarrayPartitioner sp = new
          SubarrayPartitioner();
        int ans =
          sp.largestSubarraySumMinimized(a,
          k);
        System.out.println("The answer is: " +
          ans);
    }
}
```

Q71. Painter's Partition

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Example 1:
Input Format: N = 4, boards[] = {5, 5, 5, 5}, k = 2
Result: 10
Explanation: We can divide the boards into 2 equal-sized partitions, so each painter gets 10 units of the board and the total time taken is 10.

Example 2:
Input Format: N = 4, boards[] = {10, 20, 30, 40}, k = 2
Result: 60
Explanation: We can divide the first 3 boards for one painter and the last board for the second painter.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N * log(sum(arr[])-max(arr[])+1)), where N = size of the array, sum(arr[]) = sum of all arr

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to solve the Painter Partition
  Problem using Binary Search
class PainterPartition {
public:
    // Helper function: counts how many
      painters are needed if each can paint at
      most 'time' units
    int countPainters(vector<int> &boards, int
      time) {
        int painters = 1;
        long long boardsPainter = 0;

        for (int i = 0; i < boards.size();
          i++) {
            if (boardsPainter + boards[i] <=
              time) {
                // Allocate board to current
                  painter
                boardsPainter += boards[i];
            } else {
                // Allocate board to new
                  painter
                painters++;
                boardsPainter = boards[i];
            }
        }

        return painters;
    }

    // Main function: finds the minimum max
      time using binary search
    int findLargestMinDistance(vector<int>
      &boards, int k) {
        int low = *max_element(boards.begin(),
          boards.end());      // At least the
          largest board
        int high = accumulate(boards.begin(),
          boards.end(), 0);    // At most the
          total sum

        int result = high;

        while (low <= high) {
            int mid = (low + high) / 2;  //
              Mid time to test
            int painters =
              countPainters(boards, mid);

            if (painters > k) {
                low = mid + 1; // Too few
                  painters → increase time
            } else {
                result = mid;  // Valid
                  solution → try to minimize
                high = mid - 1;
            }
        }

        return result;
    }
};
```

Q72. Minimize Max Distance to Gas Station

HARD

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Example 1:
Input Format: N = 5, arr[] = {1,2,3,4,5}, k = 4
Result: 0.5
Explanation: One of the possible ways to place 4 gas stations is {1,1.5,2,2.5,3,3.5,4,4.5,5}. Thus the maximum difference between adjacent gas stations is 0.5. Hence, the value of 'dist' is... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(nlogn + klogn), n = size of the given array, k = no. of gas stat*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to solve the gas station placement
  problem
class Solution {
public:
    // Function to minimize the maximum
      distance between gas stations
    long double
      minimiseMaxDistance(vector<int> &arr,
      int k) {
        int n = arr.size();
        vector<int> howMany(n - 1, 0); //
          Tracks how many stations in each
          segment
        priority_queue<pair<long double, int>>
          pq; // Max-heap

        // Initially insert all segments into
          the heap with their lengths
        for (int i = 0; i < n - 1; i++) {
            long double length = arr[i + 1] -
              arr[i];
            pq.push({length, i});
        }

        // Place k additional gas stations
        for (int gasStations = 1; gasStations
          <= k; gasStations++) {
            auto top = pq.top();
            pq.pop();
            int segmentIndex = top.second;

            // Add a station to the segment
              and recompute its length
            howMany[segmentIndex]++;
            long double totalDist =
              arr[segmentIndex + 1] -
              arr[segmentIndex];
            long double newLen = totalDist /
              (howMany[segmentIndex] + 1);
            pq.push({newLen, segmentIndex});
        }

        // Final answer is the max segment
          length at the top of the heap
        return pq.top().first;
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n*log(Len)) + O(n), n = size of the given array, Len = length of t*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class that contains methods for optimizing
  gas station placement
class GasStationOptimizer {
public:
    // Function to calculate number of gas
      stations required such that
    // no segment exceeds the max allowed
      distance `dist`
    int numberOfGasStationsRequired(long
      double dist, vector<int> &arr) {
        int n = arr.size();
        int cnt = 0;

        for (int i = 1; i < n; i++) {
            // Number of stations needed
              between arr[i-1] and arr[i]
            int numberInBetween = (arr[i] -
              arr[i - 1]) / dist;

            // If perfectly divisible, reduce
              one to avoid extra placement
            if ((arr[i] - arr[i - 1]) == (dist
              * numberInBetween)) {
                numberInBetween--;
            }
            cnt += numberInBetween;
        }
        return cnt;
    }

    // Function to minimize the maximum
      distance between gas stations
    long double
      minimiseMaxDistance(vector<int> &arr,
      int k) {
        int n = arr.size();
        long double low = 0, high = 0;

        // Determine max initial distance
          between stations
        for (int i = 0; i < n - 1; i++) {
            high = max(high, (long
              double)(arr[i + 1] - arr[i]));
        }

        long double diff = 1e-6;

        // Binary search to find minimum
          possible maximum distance
        while (high - low > diff) {
            long double mid = (low + high) /
              2.0;
            int cnt =
              numberOfGasStationsRequired(mid,
              arr);
            if (cnt > k) low = mid;
            else high = mid;
        }

        return high;
    }
};
```

Q73. Kth element of 2 sorted arrays

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on Answers

Practice Link

Example 1:
Input: a = [2, 3, 6, 7, 9], b = [1, 4, 8, 10], k = 5
Output: 6
Explanation: The final sorted array would be [1, 2, 3, 4, 6, 7, 8, 9, 10]. The 5th element of this array is 6.

Example 2:
Input: a = [100, 112, 256, 349, 770], b = [72, 86, 113, 119, 265, 445, 892], k = 7
Output: 256
Explanation: The final sorted array is [72, 86, 100, 112, 113, 119, 256, 265, 349, 445, 770, 892]. The 7th element of this array is 256.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(log(min(M, N))), where M and N are the sizes of the two given arrays. As binary search is b

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    int kthElement(vector<int>& a,
      vector<int>& b, int k) {
        int m = a.size();
        int n = b.size();
        // Ensure a is smaller array for
          optimization
        if (m > n) {
            // Swap a and b
            return kthElement(b, a, k);
        }
        // Length of the left half
        int left = k;
        // Apply binary search
        int low = max(0, k - n), high = min(k,
          m);
        while (low <= high) {
            int mid1 = (low + high) >> 1;
            int mid2 = left - mid1;
            // Initialize l1, l2, r1, r2
            int l1 = (mid1 > 0) ? a[mid1 - 1]
              : INT_MIN;
            int l2 = (mid2 > 0) ? b[mid2 - 1]
              : INT_MIN;
            int r1 = (mid1 < m) ? a[mid1] :
              INT_MAX;
            int r2 = (mid2 < n) ? b[mid2] :
              INT_MAX;
            // Check if we have found the
              answer
            if (l1 <= r2 && l2 <= r1) {
                return max(l1, l2);
            }
            else if (l1 > r2) {
                // Eliminate the right half
                high = mid1 - 1;
            }
            else {
                // Eliminate the left half
                low = mid1 + 1;
            }
        }
        // Dummy return statement
        return -1;
    }
};
```

Q74. Find row with maximum 1's

EASY

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 2D Arrays

Practice Link

Example 1:
Input Format: n = 3, m = 3,
mat[] =
1 1 1
0 0 1
0 0 0
Result: 0
Explanation: The row with the maximum number of ones is 0 (0 - indexed).

Example 2:
Input Format: n = 2, m = 2 ,
mat[] =
0 0
0 0
Result: -1
Explanation: The matrix does not contain any 1. So, -1 is the answer.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity:O(n X logm), where n = given row number, m = given column number. We are using a loop running

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Binary search to find the first index where value >= x
    int lowerBound(vector<int> &arr, int n, int x) {
        int low = 0, high = n - 1;
        int ans = n;  // Default if x not found

        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] >= x) {
                ans = mid;       // Possible answer
                high = mid - 1;  // Look for smaller index
            } else {
                low = mid + 1;   // Search right half
            }
        }
        return ans;
    }

    // Find row with max number of 1s
    int rowWithMax1s(vector<vector<int>> &matrix, int n, int m) {
        int cnt_max = 0;
        int index = -1;

        for (int i = 0; i < n; i++) {
            int cnt_ones = m - lowerBound(matrix[i], m, 1);  // 1s = total -
              index of first 1
            if (cnt_ones > cnt_max) {
                cnt_max = cnt_ones;
                index = i;
            }
        }
        return index;
    }
};
```

Q75. Search in a 2D matrix

HARD

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 2D Arrays

Practice Link

Input :mat = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]], target = 8
Output : True.
Explanation :The target = 8 exists in the 'mat' at index (1, 3).
Input :mat = [[1, 2, 4], [6, 7, 8], [9, 10, 34]], target = 78
Output :false.
Explanation :The target = 78... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(n × log m), We go through each of the `n` rows once. For any valid*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to perform binary search on a
      1D vector
    bool binarySearch(vector<int>& nums, int
      target) {
        // Get the size of the array
        int n = nums.size();
        // Initialize low and high pointers
          for binary search
        int low = 0, high = n - 1;
        // Binary search loop
        while (low <= high) {
            // Calculate the middle index
            int mid = (low + high) / 2;
            // If the middle element is the
              target, return true
            if (nums[mid] == target)
                return true;
            // If the target is greater, move
              to the right half
            else if (target > nums[mid])
                low = mid + 1;
            // Otherwise, move to the left
              half
            else
                high = mid - 1;
        }
        // Return false if target not found
        return false;
    }
    // Function to search for target in the 2D
      matrix
    bool searchMatrix(vector<vector<int>>&
      matrix, int target) {
        // Get the number of rows
        int n = matrix.size();
        // Get the number of columns
        int m = matrix[0].size();
        // Traverse each row
        for (int i = 0; i < n; i++) {
            // Check if target could be within
              this row
            if (matrix[i][0] <= target &&
              target <= matrix[i][m - 1]) {
                // Apply binary search on the
                  current row
                return binarySearch(matrix[i],
                  target);
            }
        }
        // Return false if target is not found
          in any row
        return false;
    }
};
// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(log(NxM)), where N = given row number, M = given column number.We*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to search target in a 2D
      matrix using binary search over a
      virtual 1D array
    bool searchMatrix(vector<vector<int>>&
      matrix, int target) {
        // Get the number of rows
        int n = matrix.size();

        // Get the number of columns
        int m = matrix[0].size();

        // Set initial binary search range
          over the imaginary 1D array
        int low = 0, high = n * m - 1;

        // Perform binary search
        while (low <= high) {
            // Calculate the middle index of
              the virtual 1D array
            int mid = (low + high) / 2;

            // Convert mid index to
              corresponding 2D indices
            int row = mid / m;
            int col = mid % m;

            // Check if the target is found
            if (matrix[row][col] == target)
                return true;

            // If the target is greater,
              discard the left half
            else if (matrix[row][col] <
              target)
                low = mid + 1;

            // If the target is smaller,
              discard the right half
            else
                high = mid - 1;
        }

        // If not found, return false
        return false;
    }
};

// Driver code
```

Q76. Search in 2D matrix - II

HARD

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 2D Arrays

Practice Link

Example 1:
Matrix=
1 4 7 11
2 5 8 12
3 6 9 16
10 13 14 17
Target: 9
Output: Found at (2,2) (0-indexed)

Example 2:
Matrix=
5 10 15
6 12 18
8 16 20
Target: 7
Output: Not Found

[BETTER] Better Approach

Complexity: *Time Complexity: O(N*logM), where N = given row number, M = given column number. We a*

```cpp
#include <bits/stdc++.h>
using namespace std;
// Class to handle binary search operations on
  a matrix
class MatrixSearch {
private:
    vector<vector<int>> matrix; // Store the
      2D matrix
    // Helper function: Perform binary search
      on a 1D vector
    bool binarySearch(vector<int>& nums, int
      target) {
        int n = nums.size();
        int low = 0, high = n - 1;
        // Binary search loop
        while (low <= high) {
            int mid = (low + high) / 2; //
              Middle index
            if (nums[mid] == target) {
                return true; // Target found
            }
            else if (target > nums[mid]) {
                low = mid + 1; // Search in
                  the right half
            }
            else {
                high = mid - 1; // Search in
                  the left half
            }
        }
        return false; // Target not found
    }
public:
    // Constructor to initialize the matrix
    MatrixSearch(vector<vector<int>>& mat) {
        matrix = mat;
    }
    // Function to search the target in the
      entire matrix
    bool searchElement(int target) {
        int n = matrix.size(); // Number of
          rows
        // Loop through each row and apply
          binary search
        for (int i = 0; i < n; i++) {
            if (binarySearch(matrix[i],
              target)) {
                return true; // Target found
                  in current row
            }
        }
        return false; // Target not found in
          any row
    }
};
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N+M), where N = given row number, M = given column number. We are*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to perform staircase search in a 2D
  sorted matrix
class MatrixSearch {
private:
    vector<vector<int>> matrix; // 2D matrix

public:
    // Constructor to initialize the matrix
    MatrixSearch(vector<vector<int>>& mat) {
        matrix = mat;
    }

    // Function to search for target in matrix
      using staircase search
    bool searchElement(int target) {
        int n = matrix.size();       // Number
          of rows
        int m = matrix[0].size();    // Number
          of columns

        int row = 0;        // Start from
          first row
        int col = m - 1;    // Start from last
          column (top-right corner)

        // Traverse the matrix
        while (row < n && col >= 0) {
            if (matrix[row][col] == target) {
                return true; // Found target
            }
            else if (matrix[row][col] <
              target) {
                row++; // Move down to next
                  row
            }
            else {
                col--; // Move left to
                  previous column
            }
        }

        return false; // Target not found
    }
};
```

Q77. Find Peak Element - II

MEDIUM

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 2D Arrays

Practice Link

Example 1:
Input: mat = [[5, 10, 8], [4, 25, 7], [3, 9, 6]]
Output: [1, 1]
Explanation: The value at index [1, 1] is 25, which is a peak because all its neighbors (10, 7, 4, 9) are smaller.

Example 2:
Input: mat = [[1, 2, 3], [6, 5, 4], [7, 8, 9]]
Output: [2, 2]
Explanation: The value at index [2, 2] is 9, which is a peak as it is greater than its neighbors (8, 4).

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N * logM), where N is the number of rows in the matrix, M is the number of columns in each

```cpp
#include <bits/stdc++.h>
using namespace std;
  class Solution {
  public:
      // Function to find the index of the row
      // with the maximum element in a given
        column
      int maxElement(vector<vector<int>>& arr,
        int col) {
          int n = arr.size();
          int max_val = INT_MIN;
          int index = -1;
          // Iterate through each row to find
            the maximum element
          // in the particular column
          for (int i = 0; i < n; i++) {
              if (arr[i][col] > max_val) {
                  max_val = arr[i][col];
                  index = i;
              }
          }
          return index;
      }
      // Function to find a peak element in
        the 2D matrix
      // using binary search
      vector<int>
        findPeakGrid(vector<vector<int>>& arr)
        {
          int n = arr.size();
          int m = arr[0].size();
          // Initialize the lower and upper
            bounds for binary search
          int low = 0;
          int high = m - 1;
          // Perform binary search on columns
          while (low <= high) {
              int mid = (low + high) / 2;
              // Find the index of the row
                with the maximum element
              // in the middle column
              int row = maxElement(arr, mid);
              // Determine the elements to the
                left and right of
              // the middle element in the
                found row
              int left = mid - 1 >= 0 ?
                arr[row][mid - 1] : INT_MIN;
              int right = mid + 1 < m ?
                arr[row][mid + 1] : INT_MIN;
              // Check if the middle element
                is greater than its neighbors
              if (arr[row][mid] > left &&
                arr[row][mid] > right) {
                  return {row, mid};
              }
              else if (left > arr[row][mid]) {
                  high = mid - 1;
              }
              else {
                  low = mid + 1;
              }
          }
          // Return {-1, -1} if no peak
            element is found
          return {-1, -1};
      }
  };
```

Q78. Matrix Median

HARD

Topic: Binary Search [1D, 2D Arrays, Search Space] > BS on 2D Arrays

Practice Link

Input: M = 3, N = 3, matrix[][] =
1 4 9
2 5 6
3 8 7
Output: 5
Explanation:
If we find the linear sorted array, the array becomes 1 2 3 4 5 6 7 8 9. Therefore, median = 5

Input: M = 3, N = 3, matrix[][] =
1 3 8
2 3 4
1 2 5
Output: 3
Explanation:
If we find the linear sorted array, the array becomes 1 1 2 2 3 3 4 5 7 8. Therefore, median = 3.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(rows × log(max - min) × log(cols)), Binary search runs on the value space from min to max o

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to count elements less than or
      equal to mid in a row
    int countLessEqual(vector<int>& row, int
      mid) {
        // Using upper_bound to find count
          efficiently
        return upper_bound(row.begin(),
          row.end(), mid) - row.begin();
    }

    // Function to find median
    int findMedian(vector<vector<int>>&
      matrix) {
        // Number of rows and columns
        int rows = matrix.size();
        int cols = matrix[0].size();

        // Minimum possible element in matrix
        int low = matrix[0][0];

        // Maximum possible element in matrix
        int high = matrix[0][cols - 1];
        for (int i = 1; i < rows; i++) {
            low = min(low, matrix[i][0]);
            high = max(high, matrix[i][cols -
              1]);
        }

        // Binary search over the value range
        while (low < high) {
            int mid = (low + high) / 2;

            // Count elements ≤ mid
            int count = 0;
            for (int i = 0; i < rows; i++) {
                count +=
                  countLessEqual(matrix[i],
                  mid);
            }

            // If count is less than half,
              median is greater
            if (count < (rows * cols + 1) / 2)
                low = mid + 1;
            else
                high = mid;
        }

        // Final low is the median
        return low;
    }
};
```

Q79. Remove Outermost Parentheses

MEDIUM

Topic: Strings [Basic and Medium] > Basic and Easy String Problems

Practice Link

Example 1:
Input: s = "((()))"
Output: "(())"
Explanation: The input string is a single primitive: "((()))".
Removing the outermost layer yields: "(())".

Example 2:
Input: s = "()(()())(())"
Output: "(()())()"
Explanation: Primitive decomposition: "()" + "(()())" + "(())"
After removing outermost parentheses: "" + "()()" + "()"
Final result: "(()())()".

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), since we are performing a single traversal of the string. Spac

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to remove outer parentheses
    string removeOuterParentheses(string s) {
        // Initialize result string
        string result = "";
        // Initialize nesting level counter
        int level = 0;

        // Traverse the string
        for (char ch : s) {
            // If we encounter '(', increase the level
            if (ch == '(') {
                // If we're inside a primitive, add '(' to result
                if (level > 0) result += ch;
                // Increase the nesting level for '('
                level++;
            }
            // If we encounter ')', decrease the level
            else if (ch == ')') {
                // Decrease the nesting level for ')'
                level--;
                // If we're inside a primitive, add ')' to result
                if (level > 0) result += ch;
            }
        }

        // Return the final result after removing the outer parentheses
        return result;
    }
};
```

Q80. Reverse words in a given string / Palindrome Check

MEDIUM

Topic: Strings [Basic and Medium] > Basic and Easy String Problems

Practice Link

Input: s = "welcome to the jungle"
Output: "jungle the to welcome"
Explanation: The words in the input string are "welcome", "to", "the", and "jungle". Reversing the order of these words gives "jungle", "the", "to", and "welcome". The output string should have exactly one space between each word.
Input: s = " amazing coding skills "
Output: "skills coding amazing"
Explanation: The input string has leading and trailing spaces, as well as multiple spaces between the words "amazing", "coding", and... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), We traverse the string once from right to left and construct the result directly withou

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to reverse the order of words
    string reverseWords(string s) {
        // Result string to store final output
        string result = "";

        // Pointer starting from the last
          character
        int i = s.size() - 1;

        // Traverse string from right to left
        while (i >= 0) {
            // Skip spaces at the current
              position
            while (i >= 0 && s[i] == ' ') {
                i--;
            }

            // If pointer is out of bounds,
              break
            if (i < 0) break;

            // Mark the end of the current
              word
            int end = i;

            // Move left until a space or
              start of string is found
            while (i >= 0 && s[i] != ' ') {
                i--;
            }

            // Extract the current word
            string word = s.substr(i + 1, end
              - i);

            // Add space before appending next
              word if result is not empty
            if (!result.empty()) {
                result += " ";
            }

            // Append the word to the result
            result += word;
        }

        return result;
    }
};

// Driver code
```

Q81. Largest Odd Number in a String

EASY

Topic: Strings [Basic and Medium] > Basic and Easy String Problems

Practice Link

Example 1
Input: s = "5347"
Output: "5347"
Explanation: The odd numbers formed by the given string are → 5, 3, 53, 347, 5347. The largest odd number without leading zeroes is 5347.

Example 2
Input: s = "0214638"
Output: "21463"
Explanation: The odd numbers formed by the string are → 1, 3, 21, 63, 463, 1463, 21463. We can't use numbers starting with 0, so the largest valid odd number is 21463.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), since the loop runs once through the string of length N. Space Com

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Returns the largest odd number substring from the given string
    string largeOddNum(string& s) {
        int ind = -1;

        // Find the last odd digit in the string
        int i;
        for (i = s.length() - 1; i >= 0; i--) {
            if ((s[i] - '0') % 2 == 1) {
                ind = i;
                break;
            }
        }

        // Skip leading zeroes up to the odd digit
        i = 0;
        while (i <= ind && s[i] == '0') i++;

        // Return substring from first non-zero to odd digit
        return s.substr(i, ind - i + 1);
    }
};
```

Q82. Longest Common Prefix

EASY

Topic: Strings [Basic and Medium] > Basic and Easy String Problems

Practice Link

Example 1
Input: str = ["flower", "flow", "flight"]
Output: "fl"
Explanation: All strings in the array begin with the common prefix "fl".

Example 2
Input: str = ["apple", "banana", "grape", "mango"]
Output: ""
Explanation: None of the strings share a common starting sequence, so the result is an empty string.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N * log N + M), where N is the number of strings and M is the minimum length of a string. T

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Returns the longest common prefix in a list of strings
    string longestCommonPrefix(vector<string>& str) {
        // Handle the case where input is empty
        if (str.empty()) return "";

        // Sort the strings lexicographically
        sort(str.begin(), str.end());

        // First string in sorted order
        string first = str[0];

        // Last string in sorted order
        string last = str[str.size() - 1];

        // To store the common prefix
        string ans = "";

        // Compare up to the length of the shorter string
        int minLength = min(first.size(), last.size());

        // Compare characters of first and last string
        for (int i = 0; i < minLength; i++) {
            // Stop if characters differ
            if (first[i] != last[i]) break;

            // Add matching character to answer
            ans += first[i];
        }

        // Return the longest common prefix
        return ans;
    }
};
```

Q83. Isomorphic String

EASY

Topic: Strings [Basic and Medium] > Basic and Easy String Problems

Practice Link

Example 1
Input: s = "paper", t = "title"
Output: true
Explanation: The characters in "s" can be mapped one-to-one to characters in "t":
'p' → 't', 'a' → 'i', 'e' → 'l', 'r' → 'e'
Since the mapping is consistent and unique for each character, the strings are isomorphic.

Example 2
Input: s = "foo", t = "bar"
Output: false
Explanation: 'f' → 'b' is fine, 'o' → 'a' for the first 'o', But the second 'o' in "s" would need to map to 'r' in "t", which conflicts with the earlier mapping of 'o' → 'a'
T... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N) where N is the length of the input strings, due to the single loop iterating through eac

```cpp
#include <bits/stdc++.h>
  using namespace std;

  class Solution {
  public:
      // Method to check if two strings are isomorphic
      bool isomorphicString(string s, string t) {
          // Arrays to store last seen positions of characters in s and t
          int m1[256] = {0}, m2[256] = {0};

          // Get the length of the strings
          int n = s.size();

          // Traverse each character of the strings
          for (int i = 0; i < n; ++i) {
              // If previous positions of current characters differ, return
                false
              if (m1[s[i]] != m2[t[i]]) return false;

              // Update the position with current index + 1
              m1[s[i]] = i + 1;
              m2[t[i]] = i + 1;
          }

          // If no mismatch is found, return true
          return true;
      }
  };

  // Main function to test the method
```

Q84. Rotate String

EASY

Topic: Strings [Basic and Medium] > Basic and Easy String Problems

Practice Link

Example 1:
Input: s = "rotation", goal = "tionrota"
Output: true
Explanation: After multiple left shifts on "rotation", we get:
1st shift → "otationr"
2nd shift → "tationro"
3rd shift → "ationrot"
4th shift → "tionrota"
So the goal string can be obtained by rotating the original string.

Example 2:
Input: s = "hello", goal = "lohelx"
Output: false
Explanation:
Even after all possible rotations of "hello", we cannot form "lohelx" due to the presence of an extra character 'x'.... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), because checking for a substring in s + s is linear in time. S

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Check if goal is a rotation of s using string doubling method
    bool rotateString(string& s, string& goal) {
        // Strings must be the same length to be rotations of each other
        if (s.length() != goal.length()) return false;

        // Concatenate s with itself and check if goal exists in it
        string doubledS = s + s;
        return doubledS.find(goal) != string::npos;
    }
};
```

Q85. Check if two strings are anagram of each other

EASY

Topic: Strings [Basic and Medium] > Basic and Easy String Problems

Practice Link

Example 1:
Input: CAT, ACT
Output: true
Explanation: Since the count of every letter of both strings are equal.

Example 2:
Input: RULES, LESRT
Output: false
Explanation: Since the count of U and T is not equal in both strings.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the length of the strings. Each string is traversed

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to check if two strings are anagrams
bool CheckAnagrams(string str1, string str2) {
    // Case: when both of the strings have different lengths
    if (str1.length() != str2.length())
        return false;

    // Initialize a frequency array to store character counts
    int freq[26] = {0};

    // Count frequency of each character in str1
    for (int i = 0; i < str1.length(); i++) {
        freq[str1[i] - 'A']++;  // Increment frequency for each character in
          str1
    }

    // Decrement frequency for each character in str2
    for (int i = 0; i < str2.length(); i++) {
        freq[str2[i] - 'A']--;  // Decrement frequency for each character in
          str2
    }

    // Check if all frequencies are zero, meaning both strings have the same
      characters
    for (int i = 0; i < 26; i++) {
        if (freq[i] != 0)  // If any frequency is non-zero, they are not
          anagrams
            return false;
    }

    return true;  // The strings are anagrams
}
```

Q86. Sort Characters by Frequency

EASY

Topic: Strings [Basic and Medium] > Medium String Problems

Practice Link

Example 1:
Input: s = "tree"
Output: ['e', 'r', 't']
Explanation:
e → 2
r → 1
t → 1
Since 'r' and 't' have the same frequency, they are sorted alphabetically → 'r' comes before 't'.

Example 2:
Input: s = "raaaajj"
Output: ['a', 'j', 'r']
Explanation:
a → 4
j → 2
r → 1
Characters are sorted by decreasing frequency. In case of ties, alphabetically.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n + k log k), where n is the length of the string and k is the constant 26 for the alphabet

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
private:
    // Custom comparator: sort by frequency
      descending, then by character ascending
    static bool comparator(pair<int, char> p1,
      pair<int, char> p2) {
        if (p1.first > p2.first) return true;
        if (p1.first < p2.first) return false;
        return p1.second < p2.second;
    }

public:
    vector<char> frequencySort(string& s) {
        // Array to store frequency and
          character
        pair<int, char> freq[26];

        // Initialize frequency array with
          character labels
        for (int i = 0; i < 26; i++) {
            freq[i] = {0, i + 'a'};
        }

        // Count frequency of each character
          in string
        for (char ch : s) {
            freq[ch - 'a'].first++;
        }

        // Sort the frequency array using the
          custom comparator
        sort(freq, freq + 26, comparator);

        // Store characters with non-zero
          frequency
        vector<char> ans;
        for (int i = 0; i < 26; i++) {
            if (freq[i].first > 0)
              ans.push_back(freq[i].second);
        }

        // Return the result
        return ans;
    }
};

// Main function to test the logic
```

Q87. Maximum Nesting Depth of the Parentheses

MEDIUM

Topic: Strings [Basic and Medium] > Medium String Problems

Practice Link

Example 1:
Input: s = "(1+(2*3)+((8)/4))+1"
Output: 3
Explanation: Digit 8 is inside of 3 nested parentheses in the string.

Example 2:
Input: s = "(1)+((2))+(((3)))"
Output: 3
Explanation: Digit 3 is inside of 3 nested parentheses in the string.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), where n is the length of the string. Space Complexity: O(1), as on

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Method to compute maximum depth of parentheses
    int maxDepth(string s) {
        int p = 0;
        int ans = 0;
        for (char x : s) {
            // Increase depth on open parenthesis
            if (x == '(') p++;
            // Decrease depth on close parenthesis
            else if (x == ')') p--;
            // Update maximum depth encountered
            ans = max(ans, p);
        }
        return ans;
    }
};

// Main function to test the method
```

Q88. Roman to Integer

MEDIUM

Topic: Strings [Basic and Medium] > Medium String Problems

Practice Link

Example 1:
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.

Example 2:
Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), where n is the length of the input string since we traverse the string once.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int romanToInt(string s) {
        int res = 0;

        // Map of Roman numerals to their integer values
        unordered_map<char, int> roman = {
            {'I', 1}, {'V', 5}, {'X', 10},
            {'L', 50}, {'C', 100}, {'D', 500}, {'M', 1000}
        };

        // Iterate through the string, except the last character
        for (int i = 0; i < s.size() - 1; i++) {
            // Subtract if current numeral is less than the next
            if (roman[s[i]] < roman[s[i + 1]]) {
                res -= roman[s[i]];
            } else {
                // Otherwise, add the current value
                res += roman[s[i]];
            }
        }

        // Add the value of the last character
        return res + roman[s.back()];
    }
};
```

Q89. String to Integer (atoi)

MEDIUM

Topic: Strings [Basic and Medium] > Medium String Problems

Practice Link

Example 1:
Input: s = " -12345"
Output: -12345
Explanation:
Ignore leading whitespaces.
The sign '-' is encountered, indicating the number is negative.
Digits 12345 are read and converted to -12345.

Example 2:
Input: s = "4193 with words"
Output: 4193
Explanation:
Read the digits 4193 and stop when encountering the first non-digit character (w).

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n) since each character is processed once. Space Complexity: O(n) since the recursion stack

```cpp
#include <bits/stdc++.h>
using namespace std;

const int INT_MIN_VAL = -2147483648;
const int INT_MAX_VAL = 2147483647;

// Recursive helper
int helper(const string &s, int i, long long num, int sign) {
    // If we are out of bounds or at a non-digit, return
    if (i >= s.size() || !isdigit(s[i]))
        return (int)(sign * num);

    // Update num with current digit
    num = num * 10 + (s[i] - '0');

    // Clamp if overflow
    if (sign * num <= INT_MIN_VAL) return INT_MIN_VAL;
    if (sign * num >= INT_MAX_VAL) return INT_MAX_VAL;

    // Recurse for next character
    return helper(s, i + 1, num, sign);
}

int myAtoi(string s, int i = 0) {
    // Skip leading whitespaces
    while (i < s.size() && s[i] == ' ') i++;

    // Handle sign
    int sign = 1;
    if (i < s.size() && (s[i] == '+' || s[i] == '-')) {
        sign = (s[i] == '-') ? -1 : 1;
        i++;
    }

    // Call recursive helper
    return helper(s, i, 0, sign);
}
```

Q90. Count Number of Substrings

EASY

Topic: Strings [Basic and Medium] > Medium String Problems

Practice Link

Example 1:
Input: s = "pqpqs", k = 2
Output: 7
Explanation: All substrings with exactly 2 distinct characters:
"pq", "pqp", "pqpq", "qp", "qpq", "pqs", "qs"
Total = 7.

Example 2:
Input: s = "abcbaa", k = 3
Output: 5
Explanation: All substrings with exactly 3 distinct characters:
"abc", "abcb", "abcba", "bcba", "cbaa"
Total = 5.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n) for each call to atMostKDistinct. Space Complexity: O(1) map size bounded by 26 characte

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to count substrings with at most k distinct characters
int atMostKDistinct(string s, int k) {
    // Left pointer and result
    int left = 0, res = 0;
    // Frequency map
    unordered_map<char, int> freq;

    // Iterate through string with right pointer
    for (int right = 0; right < s.size(); right++) {
        // Add current character
        freq[s[right]]++;

        // Shrink window if distinct characters exceed k
        while (freq.size() > k) {
            freq[s[left]]--;
            if (freq[s[left]] == 0) freq.erase(s[left]);
            left++;
        }

        // Count substrings in current window
        res += (right - left + 1);
    }
    return res;
}

// Function to count substrings with exactly k distinct characters
int countSubstrings(string s, int k) {
    // Exactly k = atMost(k) - atMost(k-1)
    return atMostKDistinct(s, k) - atMostKDistinct(s, k - 1);
}
```

Q91. Sum of Beauty of All Substrings

MEDIUM

Topic: Strings [Basic and Medium] > Medium String Problems

Practice Link

Example 1:
Input: s = "xyx"
Output: 1
Explanation: The substrings with non-zero beauty are:
"xyx" → frequencies: x:2, y:1 → beauty = 2 - 1 = 1
"xy" → x:1, y:1 → beauty = 0
"yx" → y:1, x:1 → beauty = 0
"x" or "y" → beauty = 0
Total sum = 1 (from "xyx") = 1

Example 2:
Input: s = "aabcbaa"
Output: 17
Explanation: Various substrings such as "aabc", "bcba", etc., have non-zero beauty values. Summing all gives 17.

[OPTIMAL SOLUTION] Approach — Time Complexity: Outer loop: O(n) (for each starting index) Inner loop: O(n) (for each ending index) Com

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int beautySum(string s) {
        int n = s.length();
        int sum = 0;

        // Loop over all substrings
        for (int i = 0; i < n; i++) {
            unordered_map<char, int> freq;

            for (int j = i; j < n; j++) {
                // Increase frequency of current character
                freq[s[j]]++;

                int maxi = INT_MIN;
                int mini = INT_MAX;

                // Find max and min frequency
                for (auto it : freq) {
                    mini = min(mini, it.second);
                    maxi = max(maxi, it.second);
                }

                // Add difference to sum
                sum += (maxi - mini);
            }
        }

        return sum;
    }
};
```

Q92. Reverse every word in a string

MEDIUM

Topic: Strings [Basic and Medium] > Medium String Problems

Practice Link

Input: s = "welcome to the jungle"
Output: "jungle the to welcome"
Explanation: The words in the input string are "welcome", "to", "the", and "jungle". Reversing the order of these words gives "jungle", "the", "to", and "welcome". The output string should have exactly one space between each word.
Input: s = " amazing coding skills "
Output: "skills coding amazing"
Explanation: The input string has leading and trailing spaces, as well as multiple spaces between the words "amazing", "coding", and... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), We traverse the string once from right to left and construct the result directly withou

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to reverse the order of words
    string reverseWords(string s) {
        // Result string to store final output
        string result = "";

        // Pointer starting from the last
          character
        int i = s.size() - 1;

        // Traverse string from right to left
        while (i >= 0) {
            // Skip spaces at the current
              position
            while (i >= 0 && s[i] == ' ') {
                i--;
            }

            // If pointer is out of bounds,
              break
            if (i < 0) break;

            // Mark the end of the current
              word
            int end = i;

            // Move left until a space or
              start of string is found
            while (i >= 0 && s[i] != ' ') {
                i--;
            }

            // Extract the current word
            string word = s.substr(i + 1, end
              - i);

            // Add space before appending next
              word if result is not empty
            if (!result.empty()) {
                result += " ";
            }

            // Append the word to the result
            result += word;
        }

        return result;
    }
};

// Driver code
```

Q93. Introduction to Singly LinkedList

EASY

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Learn 1D LinkedList

Practice Link

Refer to the code implementation and logic below.

[OPTIMAL SOLUTION] General

```cpp
#include <bits/stdc++.h>
using namespace std;

// Node class represents a node in the linked list
class Node {
public:
    int data;     // Data value
    Node* next;   // Pointer to next node

    // Constructor with data and next
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }

    // Constructor with only data
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
```

Q94. Insertion at the head of Linked List

EASY

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Learn 1D LinkedList

Practice Link

Input: 0->1->2, val = 5
Output: 5->0->1->2
Explanation: We need to insert the value 5 before the head of the given Linked List.

Input: 12->5->8->7, val = 4
Output: 4->12->5->8->7
Explanation: We need to insert the value 4 before the head of the given Linked List.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1), creating a new node and updating the head takes constant time. Space Complexity: O(

```cpp
#include <bits/stdc++.h>
using namespace std;

// Node class to represent each node in the
  linked list
class Node {
public:
    // Data stored in the node
    int data;
    // Pointer to the next node
    Node* next;

    // Constructor with data and next pointer
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }

    // Constructor with only data
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
```

```cpp
// Solution class to handle linked list
  operations
class Solution {
public:
    // Function to insert a new node at the
      head
    Node* insertAtHead(Node* head, int
      newData) {
        // Create a new node whose next points
          to current head
        Node* newNode = new Node(newData,
          head);
        // Return the new node as the head
        return newNode;
    }

    // Function to print the linked list
    void printList(Node* head) {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};
```

Q95. Deletion of the head of LL

EASY

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Learn 1D LinkedList

Practice Link

Input: 0->1->2
Output: 0->1

Explanation: Last node of the Linked List is 2.

Input: 12->5->8->7
Output: 12->5->8
Explanation: Last node of the Linked List is 7.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), we traverse the entire linked list once to delete the tail of the list. Space Compl

```cpp
#include <bits/stdc++.h>
using namespace std;

// Definition for singly linked list
struct Node {
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};

class Solution {
public:
    // Function to delete tail node of linked
      list
    Node* deleteTail(Node* head) {
        // If list is empty or has one node
        if (head == NULL || head->next ==
          NULL) {
            delete head;
            return NULL;
        }

        // Traverse to the second last node
        Node* curr = head;
        while (curr->next->next != NULL) {
            curr = curr->next;
        }

        // Delete tail node
        delete curr->next;
        curr->next = NULL;

        // Return updated head
        return head;
    }
};

// Driver code
```

Q96. Find the length of the Linked List

EASY

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Learn 1D LinkedList

Practice Link

Input: 0->1->2
Output: 3
Explanation: The list has a total of 3 nodes, thus the length of the list is 3.

Input: 12->5->8->7
Output: 4
Explanation: The list has a total of 4 nodes, thus the length of the list is 4.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), we traverse the entire linked list once to find the total number of nodes. Space Co

```cpp
#include <bits/stdc++.h>
using namespace std;

// Node class to represent each element in the
  linked list
class Node {
public:
    int data;
    Node* next;

    // Constructor to initialize data and next
      pointer
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};

// Solution class containing the method to
  find length
class Solution {
public:
    // Function to find the length of the
      linked list
    int lengthOfLinkedList(Node* head) {
        // Initialize counter to 0
        int count = 0;

        // Initialize a temporary pointer to
          head
        Node* temp = head;

        // Traverse the linked list
        while (temp != nullptr) {
            // Increment count for each node
            count++;

            // Move to the next node
            temp = temp->next;
        }

        // Return the total count
        return count;
    }
};
```

Q97. Search in Linked List

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Learn 1D LinkedList

Practice Link

Input: 0->1->2, val = 2
Output: True
Explanation: Since element 2 is present in the list, return true.

Input: 12->5->8->7, val = 6
Output: False
Explanation: The list does not contain element 6. Therefore, return false.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), we traverse the entire linked list once in worst case to search for the required value.

```cpp
#include <bits/stdc++.h>
using namespace std;

// Node structure for Linked List
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(NULL) {}
};

// Solution class containing search function
class Solution {
public:
    // Function to search for a value in LL
    bool searchValue(Node* head, int key) {
        // Pointer to traverse the list
        Node* current = head;

        // Traverse until end
        while (current != NULL) {
            // Check if current node matches key
            if (current->data == key) {
                // Return true if found
                return true;
            }
            // Move to next node
            current = current->next;
        }

        // Return false if not found
        return false;
    }
};

// Driver code
```

Q98. Introduction to Doubly LL

EASY

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Learn Doubly LinkedList

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```java
// Node class represents a node in the linked
  list
class Node {
    int data;      // Data value
    Node next;     // Pointer to next node

    // Constructor with data and next
    Node(int data1, Node next1) {
        data = data1;
        next = next1;
    }

    // Constructor with only data
    Node(int data1) {
        data = data1;
        next = null;
    }
}

public class Main {
    public static void main(String[] args) {
        // Create an array
        int[] arr = {2, 5, 8, 7};

        // Create first node
        Node y = new Node(arr[0]);

        // Print memory reference of node
        System.out.println(y);

        // Print data stored in node
        System.out.println(y.data);
    }
}
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Node class represents a node in the linked
  list
class Node {
public:
    int data;     // Data value
    Node* next;   // Pointer to next node

    // Constructor with data and next
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }

    // Constructor with only data
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
```

Q99. Insert node before head in Doubly Linked List

EASY

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Learn Doubly LinkedList

Practice Link

Example 1:
Input Format:
DLL: 1 <-> 2 <-> 3 <-> 4
Value to be Inserted: 6
Result: DLL: 1 <-> 2 <-> 3 <-> 4 <-> 6
Explanation: A new node with value 6 has been inserted at the end of the doubly linked list after the tail node.

Example 2:
Input Format:
DLL: 10 <-> 20 <-> 30
Value to be Inserted: 40
Result: DLL: 10 <-> 20 <-> 30 <-> 40
Explanation: In this case, a new node with value 40 is inserted after 30, which is at the end of the doubly linked list.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), where n is the number of nodes in the doubly linked list. This is because we traverse t

```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node* back;
    Node(int data1, Node* next1, Node* back1)
      {
        data = data1;
        next = next1;
        back = back1;
    }
    Node(int data1) {
        data = data1;
        next = nullptr;
        back = nullptr;
    }
};
Node* convertArr2DLL(vector<int> arr) {
    Node* head = new Node(arr[0]);
    Node* prev = head;
    for (int i = 1; i < arr.size(); i++) {
        Node* temp = new Node(arr[i], nullptr,
          prev);
        prev->next = temp;
        prev = temp;
    }
    return head;
}
void print(Node* head) {
    while (head != nullptr) {
        cout << head->data << " ";
        head = head->next;
    }
}
Node* insertAtTail(Node* head, int k) {
    Node* newNode = new Node(k);
    if (head == nullptr) {
        return newNode;
    }
    Node* tail = head;
    while (tail->next != nullptr) {
        tail = tail->next;
    }
    tail->next = newNode;
    newNode->back = tail;
    return head;
}
```

Q100. Delete head of Doubly Linked List

EASY

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Learn Doubly LinkedList

Practice Link

Input: DLL: 1 <-> 3 <-> 4 <-> 1

Output: DLL: 1 <-> 3 <-> 4

Explanation: Last node of the Doubly Linked List to be deleted is 1.

Input: DLL: 7 <-> 5

Output: DLL: 7

Explanation: Last node of the Doubly Linked List to be deleted is 5.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), we traverse the entire linked list once to delete the tail of the list. Space Compl

```cpp
#include <bits/stdc++.h>
using namespace std;

// Node structure for DLL
struct Node {
    int data;
    Node* prev;
    Node* next;
    Node(int val) {
        data = val;
        prev = NULL;
        next = NULL;
    }
};

class Solution {
public:
    // Function to delete tail of DLL
    Node* deleteTail(Node* head) {
        // If list is empty
        if (head == NULL) return NULL;

        // If only one node present
        if (head->next == NULL) {
            delete head;
            return NULL;
        }

        // Traverse to the last node
        Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }

        // Update second last node's next to
          NULL
        temp->prev->next = NULL;

        // Delete tail node
        delete temp;

        // Return head
        return head;
    }
};
```

Q101. Reverse a Doubly Linked List

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Learn Doubly LinkedList

Practice Link

Input : DLL - 1 <-> 2 <-> 3 <-> 4

Output: DLL - 4 <-> 3 <-> 2 <-> 1

Explanation: The doubly linked list is reversed and its last node is returned at the new head pointer.
Input: DLL-10 <-> 20 <-> 30

Output: DLL - 30 <-> 20 <-> 10

Explanation: In this case, the doubly linked list is reversed and its former tail is returned as its new head.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity : O(N) We only have to traverse the doubly linked list once, hence our time complexity is O(N)

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node* back;
    Node(int data1, Node* next1, Node* back1)
      {
        data = data1;
        next = next1;
        back = back1;
    }
    Node(int data1) {
        data = data1;
        next = nullptr;
        back = nullptr;
    }
};
Node* convertArr2DLL(vector<int> arr) {
    Node* head = new Node(arr[0]);
    Node* prev = head;
    for (int i = 1; i < arr.size(); i++) {
        Node* temp = new Node(arr[i], nullptr,
          prev);
        prev->next = temp;
        prev = temp;
    }
    return head;
}
void print(Node* head) {
    while (head != nullptr) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
Node* reverseDLL(Node* head) {
    if (head == nullptr || head->next ==
      nullptr) return head;
    Node* curr = head;
    while (curr != nullptr) {
        Node* temp = curr->next;
        curr->next = curr->back;
        curr->back = temp;
        head = curr;
        curr = temp;
    }
    return head;
}
```

Q102. Middle of a LinkedList [TortoiseHare Method]

EASY

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Example 1:
Input: LL: 1 2 3 4 5

Output: 3

Explanation: Node with value 3 is the middle node of this linked list.

Example 2:
Input: LL: 1 2 3 4 5 6

Output: 4

Explanation: In this example, the linked list has an even number of nodes hence we return the second middle node which is 4.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N/2) The algorithm requires the 'fast' pointer to reach the end of the list which it does a

```cpp
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
// Node class represents a node in a linked
  list
class Node {
public:
    // Data stored in the node
    int data;
     // Pointer to the next node in the list
    Node* next;
    // Constructor with both data
    // and next node as parameters
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }
    // Constructor with only data as
    // a parameter, sets next to nullptr
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
// Function to find the middle
// node of a linked list
Node *findMiddle(Node *head) {
     // Initialize the slow pointer to the
       head.
    Node *slow = head;
     // Initialize the fast pointer to the
       head.
    Node *fast = head;
    // Traverse the linked list using the
    // Tortoise and Hare algorithm.
    while (fast != NULL && fast->next != NULL)
      {
        // Move slow one step.
        slow = slow->next;
         // Move fast two steps.
        fast = fast->next->next;
    }
     // Return the slow pointer,
     // which is now at the middle node.
    return slow;
}
```

Q103. Reverse a LinkedList [Iterative]

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input- LL: 1 3 2 4
Output: 3
Explanation: After reversing the linked list, the new head will point to the tail of the old linked list.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N) Because we are traversing each node of the linked list exactly once. Each pointer revers

```cpp
#include <bits/stdc++.h>
using namespace std;

class ListNode {
public:
    int val;
    ListNode* next;

    // Constructor to initialize node value
    ListNode(int val) {
        this->val = val;
        this->next = NULL;
    }
};

class Solution {
public:
    // Function to reverse a linked list
      iteratively
    ListNode* reverseList(ListNode* head) {
        // Initialize previous pointer to NULL
        ListNode* prev = NULL;

        // Start from the head of the list
        ListNode* temp = head;

        // Traverse the list
        while (temp != NULL) {
            // Save the next node
            ListNode* front = temp->next;

            // Reverse the current node's
              pointer
            temp->next = prev;

            // Move prev to current node
            prev = temp;

            // Move to the next node
            temp = front;
        }

        // Return new head (last node becomes
          first)
        return prev;
    }
};

// Driver code
```

Q104. Reverse a LL

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input- LL: 1 3 2 4
Output: 3
Explanation: After reversing the linked list, the new head will point to the tail of the old linked list.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N) Because we are traversing each node of the linked list exactly once. Each pointer revers

```cpp
#include <bits/stdc++.h>
using namespace std;

class ListNode {
public:
    int val;
    ListNode* next;

    // Constructor to initialize node value
    ListNode(int val) {
        this->val = val;
        this->next = NULL;
    }
};

class Solution {
public:
    // Function to reverse a linked list
      iteratively
    ListNode* reverseList(ListNode* head) {
        // Initialize previous pointer to NULL
        ListNode* prev = NULL;

        // Start from the head of the list
        ListNode* temp = head;

        // Traverse the list
        while (temp != NULL) {
            // Save the next node
            ListNode* front = temp->next;

            // Reverse the current node's
              pointer
            temp->next = prev;

            // Move prev to current node
            prev = temp;

            // Move to the next node
            temp = front;
        }

        // Return new head (last node becomes
          first)
        return prev;
    }
};

// Driver code
```

Q105. Detect a loop in LL

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input: LL: 1 2 3 4 5

Output: True
Explanation: The last node with the value of 5 has its 'next' pointer pointing back to a previous node with the value of 3. This has resulted in a loop, hence we return true.

Input: LL: 1 2 3 4 9 9

Output: False
Explanation: In this example, the linked list does not have a loop hence returns false.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), we traverse the entire linked list once. The fast pointer either reaches the end of the

```cpp
#include <bits/stdc++.h>
using namespace std;

// Definition of singly linked list
struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(NULL) {}
    ListNode(int data1) : val(data1),
      next(NULL) {}
    ListNode(int data1, ListNode *next1) :
      val(data1), next(next1) {}
};

class Solution {
public:
    // Function to detect a loop in a linked
    // list using the Tortoise and Hare
      Algorithm
    bool hasCycle(ListNode *head) {
        // Initialize two pointers, slow and
          fast,
        // to the head of the linked list
        ListNode *slow = head;
        ListNode *fast = head;

        // Step 2: Traverse the linked list
          with
        // the slow and fast pointers
        while (fast != nullptr && fast->next
          != nullptr) {
            // Move slow one step
            slow = slow->next;
            // Move fast two steps
            fast = fast->next->next;

            // Check if slow and fast pointers
              meet
            if (slow == fast) {
                return true;  // Loop detected
            }
        }

        // If fast reaches the end of the
          list,
        // there is no loop
        return false;
    }
};

// Main function to test the Solution
```

Q106. Find the starting point in LL

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input: LL: 1 2 3 4 5
Output: 3
Explanation: This linked list contains a loop of size 3 starting at node with value 3.

Input: LL: 1 -> 2 -> 3 -> 4 -> 9 -> 9
Output:NULL
Explanation: This linked list does not contain a loop hence has no starting point

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N) where N is the number of nodes in the linked list. In the worst case, we traverse the en

```cpp
#include <bits/stdc++.h>
using namespace std;
class ListNode {
public:
    int val;
    ListNode* next;
    ListNode(int x) {
        val = x;
        next = NULL;
    }
};
class Solution {
public:
    // Function to detect the starting point
      of the loop using Floyd's algorithm
    ListNode* detectCycle(ListNode* head) {
        // Initialize slow and fast pointers
        ListNode* slow = head;
        ListNode* fast = head;
        // Traverse until fast and fast->next
          are not null
        while (fast != NULL && fast->next !=
          NULL) {
            // Move slow by one step
            slow = slow->next;
            // Move fast by two steps
            fast = fast->next->next;
            // If they meet, loop is detected
            if (slow == fast) {
                // Reset slow to head
                slow = head;
                // Move both one step at a
                  time to find starting point
                while (slow != fast) {
                    slow = slow->next;
                    fast = fast->next;
                }
                // Return the start node of
                  the loop
                return slow;
            }
        }
        // If no loop found, return NULL
        return NULL;
    }
};
// Driver code
```

Q107. Length of loop in LL

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input:

Output: 3
Explanation: A cycle exists in the linked list starting at node 3 -> 4 -> 5 and then back to 3. There are 3 nodes present in this cycle.

Input:

Output: 0
Explanation: The linked list is linear and does not have a loop hence return 0.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), we traverse the entire linked list atleast once to find the length of the loop. Spa

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
class Solution {
public:
    int lengthOfLoop(Node* head) {
        Node* slow = head;
        Node* fast = head;
        while (fast != NULL && fast->next !=
          NULL) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast) {
                return countLoopLength(slow);
            }
        }
        return 0;
    }
    int countLoopLength(Node* meetingPoint) {
        Node* temp = meetingPoint;
        int length = 1;
        while (temp->next != meetingPoint) {
            temp = temp->next;
            length++;
        }
        return length;
    }
};
```

Q108. Check if LL is palindrome or not

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Example 1:
Input: head -> 3 -> 7 -> 5 -> 7 -> 3
Output: true
Explanation: 37573 is a palindrome.

Example 2:
Input: head -> 1 -> 1 -> 2 -> 1
Output: false
Explanation: 1121 is not a palindrome.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), we traverse the list twice, once to reverse half of it and once

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
Node* reverseLinkedList(Node* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }
    Node* newHead =
      reverseLinkedList(head->next);
    Node* front = head->next;
    front->next = head;
    head->next = NULL;
    return newHead;
}
bool isPalindrome(Node* head) {
    if (head == NULL || head->next == NULL) {
        return true;
    }
    Node* slow = head;
    Node* fast = head;
    while (fast->next != NULL &&
      fast->next->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }
    Node* newHead =
      reverseLinkedList(slow->next);
    Node* first = head;
    Node* second = newHead;
    while (second != NULL) {
        if (first->data != second->data) {
            reverseLinkedList(newHead);
            return false;
        }
        first = first->next;
        second = second->next;
    }
    reverseLinkedList(newHead);
    return true;
}
void printLinkedList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
```

Q109. Segregate odd and even nodes in Linked List

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input: 1→2→3→4→5→6→Null
Output: 2→4→6→1→3→5→Null
Explanation : Odd Nodes in LinkedList are 1,3,5 and Even Nodes in LinkedList are 2,4,6
In Modified LinkedList all even Nodes comes before all Odd Nodes. So Modified LinkedList looks like 2→4→6→1→3→5→Null. Order of even and odd Nodes is maintained in modified LinkedList.
Input: 1→3→5→Null
Output: 1→3→5→Null
Explanation: As there are no Even Nodes in LinkedList, The Modified LinkedList is same as Original LinkedList.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), We traverse the entire linked list only once to rearrange the nodes. Each node is visi

```
#include <bits/stdc++.h>
using namespace std;
class ListNode {
public:
    int val;
    ListNode* next;
    ListNode(int x) {
        val = x;
        next = nullptr;
    }
};
ListNode* head, *tail;
void PrintList(ListNode* head) {
    ListNode* curr = head;
    for (; curr != nullptr; curr = curr->next)
        cout << curr->val << "-->";
    cout << "null" << endl;
}
void InsertatLast(int value) {
    ListNode* newnode = new ListNode(value);
    if (head == nullptr)
        head = newnode, tail = newnode;
    else
        tail = tail->next = newnode;
}
ListNode* SegregatetoOddEVen() {
    ListNode* oddHead = new ListNode(-1),
      *oddTail = oddHead;
    ListNode* evenHead = new ListNode(-1),
      *evenTail = evenHead;
    ListNode* curr = head, *temp;
    while (curr) {
        temp = curr;
        curr = curr->next;
        temp->next = nullptr;
        if (temp->val & 1) {
            oddTail->next = temp;
            oddTail = temp;
        }
        else {
            evenTail->next = temp;
            evenTail = temp;
        }
    }
    evenTail->next = oddHead->next;
    return evenHead->next;
}
```

Q110. Remove Nth node from the back of the LL

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input: 5->1->2, N=2
Output: 5->2
Explanation: The 2nd node from the end of the linked list is 1. Therefore, we get this result after removing 1 from the linked list.

Input: 1->2->3->4->5, N=3
Output: 1->2->4->5
Explanation: The 3rd node from the end is 3, therefore, we remove 3 from the linked list.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), since the fast pointer will traverse the entire linked list, where N is the length of t

```cpp
#include <bits/stdc++.h>
using namespace std;
// Class representing a node in the linked
  list
class Node {
public:
    int data;
    Node* next;
    // Constructor for Node with data and next
      node
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }
    // Constructor for Node with only data
      (next = nullptr)
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
// Class to hold the solution logic
class Solution {
public:
    // Function to print the linked list
    void printLL(Node* head) {
        while (head != NULL) {
            cout << head->data << " ";
            head = head->next;
        }
    }
    // Function to delete the Nth node from
      the end
    // using the optimized two-pointer method
    Node* deleteNthNodeFromEnd(Node* head, int
      N) {
        // Create a dummy node before head to
          handle edge cases
        Node* dummy = new Node(0, head);
        // Initialize slow and fast pointers
          at dummy
        Node* slow = dummy;
        Node* fast = dummy;
        // Move fast pointer N+1 steps ahead
          to create a gap
        for (int i = 0; i <= N; i++) {
            fast = fast->next;
        }
        // Move both pointers until fast
          reaches the end
        while (fast != NULL) {
            slow = slow->next;
            fast = fast->next;
        }
        // Slow is now at node before target →
          delete target node
        slow->next = slow->next->next;
        // Return updated head
        return dummy->next;
    }
};
// Main driver code
```

Q111. Delete the middle node in LL

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input: 1->2->3->4->5

Output: 1->2->4->5

Explanation: Node with value 3 is at the middle node and deleted.

Input: 1->2->3->4

Output: 1->2->4

Explanation: The linked list has an even number of nodes hence we delete the second middle node which is 3.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N/2), we traverse the entire linked list using slow and fast pointers, effectively covering

```cpp
#include <bits/stdc++.h>
using namespace std;
// Node class represents a node in the linked
  list
class Node {
public:
    // Data stored in the node
    int data;
    // Pointer to the next node
    Node* next;
    // Constructor with data and next pointer
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }
    // Constructor with only data
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
// Function to delete the middle node using
  slow and fast pointer
Node* deleteMiddle(Node* head) {
    // If list has only one node, delete it
    if (head == nullptr || head->next ==
      nullptr) {
        delete head;
        return nullptr;
    }
    // Initialize slow pointer to head
    Node* slow = head;
    // Initialize fast pointer two steps ahead
    Node* fast = head->next->next;
    // Traverse until fast reaches end
    while (fast != nullptr && fast->next !=
      nullptr) {
        slow = slow->next;
        fast = fast->next->next;
    }
    // Pointer to middle node
    Node* middle = slow->next;
    // Bypass the middle node
    slow->next = slow->next->next;
    // Delete middle node
    delete middle;
    // Return head of updated list
    return head;
}
// Function to print linked list
void printLL(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
// Driver function
```

Q112. Sort LL

HARD

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input: 3->4->2->1->5

Output: 1->2->3->4->5

Explanation: The input linked list when sorted from [3, 4, 2, 1, 5] results in a linked list with values [1, 2, 3, 4, 5].

Input: 40->20->60->10->50->30

Output: 10->20->30->40->50->60

Explanation: The input li... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N*LogN), we recursively divide the linked list into two halves and then merge two sorted ha

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
class Solution {
public:
    Node* mergeTwoSortedLinkedLists(Node*
      list1, Node* list2) {
        Node* dummyNode = new Node(-1);
        Node* temp = dummyNode;
        while (list1 != nullptr && list2 !=
          nullptr) {
            if (list1->data <= list2->data) {
                temp->next = list1;
                list1 = list1->next;
            } else {
                temp->next = list2;
                list2 = list2->next;
            }
            temp = temp->next;
        }
        if (list1 != nullptr) {
            temp->next = list1;
        } else {
            temp->next = list2;
        }
        return dummyNode->next;
    }
    Node* findMiddle(Node* head) {
        if (head == nullptr || head->next ==
          nullptr) {
            return head;
        }
        Node* slow = head;
        Node* fast = head->next;
        while (fast != nullptr && fast->next
          != nullptr) {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
    Node* sortLL(Node* head) {
        if (head == nullptr || head->next ==
          nullptr) {
            return head;
        }
        Node* middle = findMiddle(head);
        Node* right = middle->next;
        middle->next = nullptr;
        Node* left = head;
        left = sortLL(left);
        right = sortLL(right);
        return mergeTwoSortedLinkedLists(left,
          right);
    }
};
void printLinkedList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
```

Q113. Sort a Linked List of 0's 1's and 2's

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input: 1 -> 2 -> 0 -> 1 -> 0 -> 2 -> NULL
Output: 0 -> 0 -> 1 -> 1 -> 2 -> 2 -> NULL

Input: 2 -> 1 -> 2 -> 0 -> 0 -> 1 -> NULL
Output: 0 -> 0 -> 1 -> 1 -> 2 -> 2 -> NULL

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), We traverse the entire list once. Space Complexity: O(1), Only dumm

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = nullptr;
    }
};
class LinkedList {
public:
    Node* head;
    LinkedList() {
        head = nullptr;
    }
    void insert(int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next)
            temp = temp->next;
        temp->next = newNode;
    }
    void print() {
        Node* temp = head;
        while (temp) {
            cout << temp->data;
            if (temp->next) cout << " -> ";
            temp = temp->next;
        }
        cout << " -> NULL\n";
    }
};
class Solution {
public:
    void sortZeroOneTwo(LinkedList& ll) {
        Node* zeroDummy = new Node(-1);
        Node* oneDummy = new Node(-1);
        Node* twoDummy = new Node(-1);
        Node* zeroTail = zeroDummy;
        Node* oneTail = oneDummy;
        Node* twoTail = twoDummy;
        Node* curr = ll.head;
        while (curr) {
            if (curr->data == 0) {
                zeroTail->next = curr;
                zeroTail = zeroTail->next;
            } else if (curr->data == 1) {
                oneTail->next = curr;
                oneTail = oneTail->next;
            } else {
                twoTail->next = curr;
                twoTail = twoTail->next;
            }
            curr = curr->next;
        }
        zeroTail->next = oneDummy->next ?
          oneDummy->next : twoDummy->next;
        oneTail->next = twoDummy->next;
        twoTail->next = nullptr;
        ll.head = zeroDummy->next;
        delete zeroDummy;
        delete oneDummy;
        delete twoDummy;
    }
};
```

Q114. Find the intersection point of Y LL

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Example 1:
Input:
List 1 = [1,3,1,2,4], List 2 = [3,2,4]
Output:
2
Explanation: Here, both lists intersecting nodes start from node 2.

Example 2:
Input:
List1 = [1,2,7], List 2 = [2,8,1]
Output:
Null
Explanation: Here,... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(n + m), Iterating through list 1 first takes O(n), then*

```cpp
#include<bits/stdc++.h>
using namespace std;
class node {
public:
    int num;
    node* next;
    node(int val) {
        num = val;
        next = NULL;
    }
};
// Utility function to insert node at the end
  of the linked list
void insertNode(node* &head, int val) {
    node* newNode = new node(val);
    if (head == NULL) {
        head = newNode;
        return;
    }
    node* temp = head;
    while (temp->next != NULL) temp =
      temp->next;
    temp->next = newNode;
}
// Utility function to check presence of
  intersection
node* intersectionPresent(node* head1, node*
  head2) {
    unordered_set<node*> st;  // Set to store
      visited nodes from the first list
    while (head1 != NULL) {
        st.insert(head1);  // Insert nodes of
          the first list into the set
        head1 = head1->next;
    }
    while (head2 != NULL) {
        if (st.find(head2) != st.end()) return
          head2;  // If node is found in set,
          it's the intersection point
        head2 = head2->next;
    }
    return NULL;  // Return NULL if no
      intersection is found
}
// Utility function to print linked list
void printList(node* head) {
    while (head->next != NULL) {
        cout << head->num << "->";
        head = head->next;
    }
    cout << head->num << endl;
}
```

[OPTIMAL] Optimal Approach 1

Complexity: *Time Complexity: O(2 × max(length of list1, length of list2)) + O(abs(length of list1*

```cpp
#include<bits/stdc++.h>
using namespace std;
class node {
public:
    int num;
    node* next;
    node(int val) {
        num = val;
        next = NULL;
    }
};
void insertNode(node* &head, int val) {
    node* newNode = new node(val);
    if (head == NULL) {
        head = newNode;
        return;
    }
    node* temp = head;
    while (temp->next != NULL) temp =
      temp->next;
    temp->next = newNode;
}
int getDifference(node* head1, node* head2) {
    int len1 = 0, len2 = 0;
    while (head1 != NULL || head2 != NULL) {
        if (head1 != NULL) {
            ++len1;
            head1 = head1->next;
        }
        if (head2 != NULL) {
            ++len2;
            head2 = head2->next;
        }
    }
    return len1 - len2;
}
node* intersectionPresent(node* head1, node*
  head2) {
    int diff = getDifference(head1, head2);
    if (diff < 0)
        while (diff++ != 0) head2 =
          head2->next;
    else
        while (diff-- != 0) head1 =
          head1->next;
    while (head1 != NULL) {
        if (head1 == head2) return head1;
        head2 = head2->next;
        head1 = head1->next;
    }
    return head1;
}
void printList(node* head) {
    while (head->next != NULL) {
        cout << head->num << "->";
        head = head->next;
    }
    cout << head->num << endl;
}
```

Q115. Add one to a number represented by LL

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Input: 4->5->6
Output: 4->5->7
Explanation: 456 + 1 = 457

Input: 9->9->9
Output: 1->0->0->0
Explanation: 999 + 1 = 1000

[BETTER] Recursive approach

Complexity: *Time Complexity: O(n), One pass for addition. Space Complexit*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
class LinkedList {
public:
    Node* append(Node* head, int value) {
        Node* newNode = new Node(value);
        if (!head) {
            return newNode;
        }
        Node* current = head;
        while (current->next)
            current = current->next;
        current->next = newNode;
        return head;
    }
    void printList(Node* head) {
        Node* current = head;
        while (current) {
            cout << current->data;
            current = current->next;
        }
        cout << endl;
    }
};
class Solution {
public:
    int addOneUtil(Node* node) {
        if (!node) return 1;
        int carry = addOneUtil(node->next);
        int sum = node->data + carry;
        node->data = sum % 10;
        return sum / 10;
    }
    Node* addOne(Node* head) {
        int carry = addOneUtil(head);
        if (carry) {
            Node* newHead = new Node(carry);
            newHead->next = head;
            head = newHead;
        }
        return head;
    }
};
```

[OPTIMAL] Iterative approach

Complexity: *Time Complexity: O(n), Two reversals + one pass for addition.*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int value) {
        data = value;
        next = nullptr;
    }
};
class LinkedList {
public:
    Node* append(Node* head, int value) {
        Node* newNode = new Node(value);
        if (!head) {
            return newNode;
        }
        Node* current = head;
        while (current->next)
            current = current->next;
        current->next = newNode;
        return head;
    }
    void printList(Node* head) {
        Node* current = head;
        while (current) {
            cout << current->data;
            current = current->next;
        }
        cout << endl;
    }
};
class Solution {
public:
    Node* reverseList(Node* node) {
        Node* prev = nullptr;
        Node* current = node;
        while (current) {
            Node* nextNode = current->next;
            current->next = prev;
            prev = current;
            current = nextNode;
        }
        return prev;
    }
    Node* addOne(Node* head) {
        head = reverseList(head);
        Node* current = head;
        int carry = 1;
        while (current && carry) {
            int sum = current->data + carry;
            current->data = sum % 10;
            carry = sum / 10;
            if (!current->next && carry) {
                current->next = new
                  Node(carry);
                carry = 0;
            }
            current = current->next;
        }
        head = reverseList(head);
        return head;
    }
};
```

Q116. Add two numbers in Linked List

MEDIUM

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of LL

Practice Link

Example 1:
Input: num1 = 243, num2 = 564
Output:sum = 807; L = [7,0,8]

Explanation: Since the digits are stored in reverse order, reverse the numbers first to get the or original number and then add them as → 342 + 465 = 807.

Example 2:
Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
Output: Result: [8,9,9,9,0,0,0,1]

Explanation: Since the digits are stored in reverse order, reverse the numbers first to get the original number and then add them as → 9999999 + 9999 = 8999001. Refer to the image b... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(max(m,n)). Assume that m and n represent the length of l1 and l2 respectively, the algorith

```
#include <bits/stdc++.h>
using namespace std;
struct ListNode {
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr) {}
    ListNode(int x) : val(x), next(nullptr) {}
    ListNode(int x, ListNode *next) : val(x),
      next(next) {}
};
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1,
      ListNode* l2) {
        ListNode *dummy = new ListNode();
        ListNode *temp = dummy;
        int carry = 0;
        while( (l1 != NULL || l2 != NULL) ||
          carry) {
            int sum = 0;
            if(l1 != NULL) {
                sum += l1->val;
                l1 = l1 -> next;
            }
            if(l2 != NULL) {
                sum += l2 -> val;
                l2 = l2 -> next;
            }
            sum += carry;
            carry = sum / 10;
            ListNode *node = new ListNode(sum
              % 10);
            temp -> next = node;
            temp = temp -> next;
        }
        return dummy -> next;
    }
};
ListNode* createList(vector arr) {
    ListNode *head = new ListNode(arr[0]);
    ListNode *temp = head;
    for (int i = 1; i < arr.size(); i++) {
        temp->next = new ListNode(arr[i]);
        temp = temp->next;
    }
    return head;
}
void printList(ListNode* head) {
    while (head) {
        cout << head->val;
        if (head->next) cout << " -> ";
        head = head->next;
    }
    cout << endl;
}
```

Q117. Delete all occurrences of a key in DLL

HARD

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of DLL

Practice Link

Input: head -> 1 <-> 2 <-> 3 <-> 1 <-> 4, target = 1
Output: head -> 2 <-> 3 <-> 4

Input: head -> 2 <-> 3 <-> -1 <-> 4 <-> 2, target = 2
Output: head -> 3 <-> -1 <-> 4

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), Every node is visited once. Space Complexity: O(1), No ext

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* prev;
    Node* next;
    Node(int val) {
        data = val;
        prev = nullptr;
        next = nullptr;
    }
};
class Solution {
public:
    void insertAtEnd(Node*& head, int val) {
        Node* newNode = new Node(val);
        if (!head) {
            head = newNode;
            return;
        }
        Node* temp = head;
        while (temp->next)
            temp = temp->next;
        temp->next = newNode;
        newNode->prev = temp;
    }
    void printList(Node* head) {
        Node* temp = head;
        while (temp) {
            cout << temp->data;
            if (temp->next) cout << " <-> ";
            temp = temp->next;
        }
        cout << endl;
    }
    Node* deleteTargetNodes(Node* head, int
      target) {
        Node* current = head;
        while (current != nullptr) {
            Node* nextNode = current->next;
            if (current->data == target) {
                if (current->prev)
                    current->prev->next =
                      current->next;
                else
                    head = current->next;
                if (current->next)
                    current->next->prev =
                      current->prev;
                delete current;
            }
            current = nextNode;
        }
        return head;
    }
};
```

Q118. Remove duplicates from sorted DLL

HARD

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Medium Problems of DLL

Practice Link

Input: head -> 1 <-> 1 <-> 3 <-> 3 <-> 4 <-> 5
Output: head -> 1 <-> 3 <-> 4 <-> 5
Explanation: Duplicate occurences of 1 and 3 are deleted

Input: head -> 1 <-> 1 <-> 1 <-> 1 <-> 1 <-> 2
Output: head -> 1 <-> 2
Explanation: All duplicate occurences of 1 is de... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), Every node is visited exactly once. Space Complexity: O(1), No

```
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    int data;
    Node* prev;
    Node* next;
    Node(int value) {
        data = value;
        prev = nullptr;
        next = nullptr;
    }
};
class Solution {
public:
    Node* head = nullptr;
    void insertAtEnd(int value) {
        Node* newNode = new Node(value);
        if (!head) {
            head = newNode;
            return;
        }
        Node* current = head;
        while (current->next) {
            current = current->next;
        }
        current->next = newNode;
        newNode->prev = current;
    }
    Node* removeDuplicates() {
        if (!head) return nullptr;
        Node* current = head;
        while (current != nullptr &&
          current->next != nullptr) {
            Node* nextDistinct =
              current->next;
            while (nextDistinct != nullptr &&
              nextDistinct->data ==
              current->data) {
                Node* duplicateNode =
                  nextDistinct;
                nextDistinct =
                  nextDistinct->next;
                delete duplicateNode;
            }
            current->next = nextDistinct;
            if (nextDistinct != nullptr) {
                nextDistinct->prev = current;
            }
            current = current->next;
        }
        return head;
    }
    void printList() {
        Node* current = head;
        while (current) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};
```

Q119. Reverse LL in group of given size K

HARD

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Hard Problems of LL

Practice Link

Input : head -> 1 -> 2 -> 3 -> 4 -> 5, k = 2
Output :head -> 2 -> 1 -> 4 -> 3 -> 5
Explanation :The groups 1 -> 2 and 3 -> 4 were reversed as 2 -> 1 and 4 -> 3.
Input :head -> 1 -> 2 -> 3 -> 4 -> 5, k = 3
Output :head -> 3 -> 2 -> 1 -> 4 -> 5
Explanation :The groups 1 -> 2 -> 3 were reversed as 3 -> 2 -> 1.
Note that 4 -> 5 was not reversed.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N),We visit each node exactly once during reversal and during group detection (getKthNode).

```
#include <bits/stdc++.h>
using namespace std;
// Definition of singly-linked list node
class ListNode {
public:
    int val;
    ListNode* next;
    ListNode(int x) {
        val = x;
        next = NULL;
    }
};
class Solution {
public:
    // Function to reverse nodes in groups of
      k
    ListNode* reverseKGroup(ListNode* head,
      int k) {
        // Creating a dummy node to handle
          edge cases easily
        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        // Pointer to keep track of the
          previous group's tail
        ListNode* groupPrev = dummy;
        while (true) {
            // Finding the k-th node from the
              groupPrev
            ListNode* kth =
              getKthNode(groupPrev, k);
            if (!kth) break;
            // Store the next group's head
            ListNode* groupNext = kth->next;
            // Break the chain to reverse
              current k-group cleanly
            ListNode* prev = groupNext;
            ListNode* curr = groupPrev->next;
            // Reversing k nodes
            for (int i = 0; i < k; i++) {
                ListNode* temp = curr->next;
                curr->next = prev;
                prev = curr;
                curr = temp;
            }
            // Connecting previous group to
              the reversed group
            ListNode* temp = groupPrev->next;
            groupPrev->next = kth;
            groupPrev = temp;
        }
        // Returning the new head
        return dummy->next;
    }
    // Helper function to find the k-th node
      from the current node
    ListNode* getKthNode(ListNode* curr, int
      k) {
        while (curr && k > 0) {
            curr = curr->next;
            k--;
        }
        return curr;
    }
};
// Driver code
```

Q120. Rotate a LL

HARD

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Hard Problems of LL

Practice Link

Input : head -> 1 -> 2 -> 3 -> 4 -> 5, k = 2
Output : head -> 4 -> 5 -> 1 -> 2 -> 3
Explanation :List after 1 shift to right: head -> 5 -> 1 -> 2 -> 3 -> 4.
List after 2 shift to right: head -> 4 -> 5 -> 1 -> 2 -> 3.
Input : head -> 1 -> 2 -> 3 -> 4 -> 5, k = 4
Output :head -> 2 -> 3 -> 4 -> 5 -> 1
Explanation :List after 1 shift to right: head -> 5 -> 1 -> 2 -> 3 -> 4.
List after 2 shift to right: head -> 4 -> 5 -> 1 -> 2 -> 3.
List after 3 shift to right: head -> 3 -> 4 -> 5 -> 1 -> 2.
List af... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), We perform a single traversal to calculate the length, another to find the new tail, an

```cpp
#include <bits/stdc++.h>
using namespace std;
// Definition of a singly linked list node
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) {
        val = x;
        next = NULL;
    }
};
class Solution {
public:
    // Function to rotate the linked list to
      the right by k places
    ListNode* rotateRight(ListNode* head, int
      k) {
        // If list is empty or has only one
          node, or no rotation is needed
        if (!head || !head->next || k == 0)
            return head;
        // Initialize length and tail pointer
        int length = 1;
        ListNode* tail = head;
        // Traverse to find the tail and
          length
        while (tail->next) {
            tail = tail->next;
            length++;
        }
        // Make it a circular linked list
        tail->next = head;
        // Effective rotations needed
        k = k % length;
        // Traverse to the new tail (length -
          k - 1 steps from head)
        int stepsToNewTail = length - k;
        ListNode* newTail = head;
        for (int i = 1; i < stepsToNewTail;
          i++) {
            newTail = newTail->next;
        }
        // Set the new head
        ListNode* newHead = newTail->next;
        // Break the circle
        newTail->next = NULL;
        return newHead;
    }
};
// Driver code
```

Q121. Flattening of LL

HARD

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Hard Problems of LL

Practice Link

Example 1:
Input:
Output: 1 2 3 4 5 6 7 8 9 10 11 12
Explanation: The linked list is flattened in sorted order, resulting in a single layer with all nodes appearing in ascending order.

Example 2:
Input:
Output: 1 2 3 4 5 6 7 8 9 10 11 12
Explanation: The... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N x (2M)) ~ O(2N x M), where N is the length of the linked list along the next pointer and

```
#include <bits/stdc++.h>
using namespace std;
struct ListNode {
    int val;
    ListNode *next;
    ListNode *child;
    ListNode() {
        val = 0;
        next = NULL;
        child = NULL;
    }
    ListNode(int data1) {
        val = data1;
        next = NULL;
        child = NULL;
    }
    ListNode(int data1, ListNode
      *next1, ListNode* next2) {
        val = data1;
        next = next1;
        child = next1;
    }
};
class Solution {
private:
    ListNode* merge(ListNode* list1,
      ListNode* list2){
        ListNode* dummyNode = new
          ListNode(-1);
        ListNode* res = dummyNode;
        while(list1 != NULL && list2
          != NULL){
            if(list1->val <
              list2->val){
                res->child = list1;
                res = list1;
                list1 =
                  list1->child;
            }
            else{
                res->child = list2;
                res = list2;
                list2 =
                  list2->child;
            }
            res->next = NULL;
        }
        if(list1){
            res->child = list1;
        } else {
            res->child = list2;
        }
        if(dummyNode->child){
            dummyNode->child->next =
              NULL;
        }
        return dummyNode->child;
    }
public:
    ListNode*
      flattenLinkedList(ListNode*
      head) {
        if(head == NULL ||
          head->next == NULL){
            return head;
        }
        ListNode* mergedHead =
          flattenLinkedList(head->ne
          xt);
        head = merge(head,
          mergedHead);
        return head;
    }
};
void printLinkedList(ListNode* head)
  {
    while (head != nullptr) {
        cout << head->val << " ";
        head = head->child;
    }
    cout << endl;
}
void
  printOriginalLinkedList(ListNode*
  head, int depth) {
    while (head != nullptr) {
        cout << head->val;
        if (head->child) {
            cout << " -> ";
printOriginalLinkedList(head->child,
              depth + 1);
        }
        if (head->next) {
            cout << endl;
            for (int i = 0; i <
              depth; ++i) {
                cout << "| ";
            }
        }
        head = head->next;
    }
}
```

Q122. Clone a LL with random and next pointer

HARD

Topic: Learn LinkedList [Single LL, Double LL, Medium, Hard Problems] > Hard Problems of LL

Practice Link

Example 1:
Input: [[1, -1], [2, 0], [3, 4], [4, 1], [5, 2]]
Output: 1 2 3 4 5, true
Explanation: All the nodes in the new list have same corresponding values as original nodes.
All the random pointers point to their corresponding nodes in the n... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(3N), where N is the number of nodes in the linked list. The algorit

```
#include <iostream>
using namespace std;
class Node {
public:
    int data;
    Node *next;
    Node *random;
    Node() : data(0), next(nullptr),
      random(nullptr){};
    Node(int x) : data(x), next(nullptr),
      random(nullptr) {}
    Node(int x, Node *nextNode, Node
      *randomNode) :
            data(x), next(nextNode),
              random(randomNode) {}
};
void insertCopyInBetween(Node* head){
    Node* temp = head;
    while(temp != NULL){
        Node* nextElement = temp->next;
        Node* copy = new Node(temp->data);
        copy->next = nextElement;
        temp->next = copy;
        temp = nextElement;
    }
}
void connectRandomPointers(Node* head){
    Node* temp = head;
    while(temp != NULL){
        Node* copyNode = temp->next;
        if(temp->random){
            copyNode->random =
              temp->random->next;
        }
        else{
            copyNode->random = NULL;
        }
        temp = temp->next->next;
    }
}
Node* getDeepCopyList(Node* head){
    Node* temp = head;
    Node* dummyNode = new Node(-1);
    Node* res = dummyNode;
    while(temp != NULL){
        res->next = temp->next;
        res = res->next;
        temp->next = temp->next->next;
        temp = temp->next;
    }
    return dummyNode->next;
}
Node *cloneLL(Node *head){
    if(!head) return nullptr;
    insertCopyInBetween(head);
    connectRandomPointers(head);
    return getDeepCopyList(head);
}
void printClonedLinkedList(Node *head) {
    while (head != nullptr) {
        cout << "Data: " << head->data;
        if (head->random != nullptr) {
            cout << ", Random: " <<
              head->random->data;
        } else {
            cout << ", Random: nullptr";
        }
        cout << endl;
        head = head->next;
    }
}
```

Q123. Recursive Implementation of atoi()

MEDIUM

Topic: Recursion [PatternWise] > Get a Strong Hold

Practice Link

Example 1:
Input: s = " -12345"
Output: -12345
Explanation:
Ignore leading whitespaces.
The sign '-' is encountered, indicating the number is negative.
Digits 12345 are read and converted to -12345.

Example 2:
Input: s = "4193 with words"
Output: 4193
Explanation:
Read the digits 4193 and stop when encountering the first non-digit character (w).

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n) since each character is processed once. Space Complexity: O(n) since the recursion stack

```cpp
#include <bits/stdc++.h>
using namespace std;

const int INT_MIN_VAL = -2147483648;
const int INT_MAX_VAL = 2147483647;

// Recursive helper
int helper(const string &s, int i, long long num, int sign) {
    // If we are out of bounds or at a non-digit, return
    if (i >= s.size() || !isdigit(s[i]))
        return (int)(sign * num);

    // Update num with current digit
    num = num * 10 + (s[i] - '0');

    // Clamp if overflow
    if (sign * num <= INT_MIN_VAL) return INT_MIN_VAL;
    if (sign * num >= INT_MAX_VAL) return INT_MAX_VAL;

    // Recurse for next character
    return helper(s, i + 1, num, sign);
}

int myAtoi(string s, int i = 0) {
    // Skip leading whitespaces
    while (i < s.size() && s[i] == ' ') i++;

    // Handle sign
    int sign = 1;
    if (i < s.size() && (s[i] == '+' || s[i] == '-')) {
        sign = (s[i] == '-') ? -1 : 1;
        i++;
    }

    // Call recursive helper
    return helper(s, i, 0, sign);
}
```

Q124. Pow(x, n)

EASY

Topic: Recursion [PatternWise] > Get a Strong Hold

Practice Link

Example 1:
Input: x = 2.0000, n = 10
Output: 1024.0000
Explanation: The answer is calculated as 2^10, which equals 1024.

Example 2:
Input: x = 2.0000, n = -2
Output: 0.2500
Explanation: The answer is calculated as 2^(-2), which is equal to 1/4 = 0.25.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(log n), where n is the absolute value of the exponent. This is because we reduce the proble

```cpp
#include<bits/stdc++.h>
using namespace std;

// Class definition for the solution
class Solution {
private:
    // Function to calculate power
    // of 'x' raised to 'n'
    double power(double x, long n) {
        // Base case: anything raised to 0 is
          1
        if (n == 0) return 1.0;

        // Base case: anything raised to 1 is
          itself
        if (n == 1) return x;

        // If 'n' is even
        if (n % 2 == 0) {
            // Recursive call: x * x, n / 2
            return power(x * x, n / 2);
        }
        // If 'n' is odd
        // Recursive call: x * power(x, n-1)
        return x * power(x, n - 1);
    }

public:
    // Function to calculate x raised to n
    double myPow(double x, int n) {
        // Store the value of n in a separate
          variable
        int num = n;

        // If n is negative
        if (num < 0) {
            // Calculate the power of -n and
              take reciprocal
            return (1.0 / power(x, -1 * num));
        }
        // If n is non-negative
        return power(x, num);
    }
};
```

Q125. Count Good Numbers

MEDIUM

Topic: Recursion [PatternWise] > Get a Strong Hold

Practice Link

Example 1:
Input: n = 1
Output: 5
Explanation: Only one index (0) → must be even.
Valid strings: "0", "2", "4", "6", "8"

Example 2:
Input: n = 2
Output: 20
Explanation: Index 0: 5 options (even digits)
Index 1: 4 options (prime digits)
Total: 5 * 4 = 20

[OPTIMAL SOLUTION] Approach — Time Complexity (TC): O(2^n) because at each index, we check 5 even digits for even indices and 4 prime digits

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MOD = 1e9 + 7;

// Recursive function to count good numbers
int countGoodNumbers(int index, int n) {
    // Base case: if we've reached the end of the string
    if (index == n) {
        // Return 1 as we've formed a valid string
        return 1;
    }

    int result = 0;
    // Even index: Use even digits
    if (index % 2 == 0) {
        // Even digits: 0, 2, 4, 6, 8
        for (int digit : {0, 2, 4, 6, 8}) {
            result = (result + countGoodNumbers(index + 1, n)) % MOD;
        }
    }
    // Odd index: Use prime digits
    else {
        // Prime digits: 2, 3, 5, 7
        for (int digit : {2, 3, 5, 7}) {
            result = (result + countGoodNumbers(index + 1, n)) % MOD;
        }
    }
    return result;
}
```

Q126. Sort a stack using recursion

MEDIUM

Topic: Recursion [PatternWise] > Get a Strong Hold

Practice Link

Example 1:
Input: stack = [4, 1, 3, 2]
Output: [4, 3, 2, 1]
Explanation: After sorting, the largest element (4) is at the top, and the smallest (1) is at the bottom.

Example 2:
Input: stack = [1]
Output: [1]
Explanation: A single-element stack is already sorted.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n2), where n is the number of elements in the stack. Space Complexity: O(n), due to the rec

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to insert element in sorted order
void insert(stack<int>& s, int temp) {
    // Base case: if the stack is empty or temp is larger than the top element
    if (s.empty() || s.top() <= temp) {
        s.push(temp);
        return;
    }

    // Otherwise, pop the top element and recursively insert
    int val = s.top();
    s.pop();
    insert(s, temp);

    // Push the popped element back
    s.push(val);
}

// Function to sort the stack
void sortStack(stack<int>& s) {
    if (!s.empty()) {
        int temp = s.top();
        s.pop();

        sortStack(s);

        insert(s, temp);
    }
}

// Main function
```

Q127. Reverse a Stack

MEDIUM

Topic: Recursion [PatternWise] > Get a Strong Hold

Practice Link

Example 1:
Input: stack = [4, 1, 3, 2]
Output: [2, 3, 1, 4]

Example 2:
Input: stack = [10, 20, -5, 7, 15]
Output: [15, 7, -5, 20, 10]

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n²), as each element is popped and inserted at the bottom (O(n) per element). Space Complex

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to insert an element at the bottom of the stack
void insertAtBottom(stack<int> &st, int val) {
    // If stack is empty, push the value
    if (st.empty()) {
        st.push(val);
        return;
    }

    // Pop the top element
    int topVal = st.top();
    st.pop();

    // Recurse for the rest of the stack
    insertAtBottom(st, val);

    // Push the popped element back
    st.push(topVal);
}

// Function to reverse the stack
void reverseStack(stack<int> &st) {
    // Base case: If stack is empty, return
    if (st.empty()) return;

    // Pop the top element
    int topVal = st.top();
    st.pop();

    // Recursively reverse the remaining stack
    reverseStack(st);

    // Insert the popped element at the bottom
    insertAtBottom(st, topVal);
}
```

Q128. Generate Binary Strings Without Consecutive 1s

MEDIUM

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Example 1:
Input: n = 3
Output: ["000", "001", "010", "100", "101"]
Explanation: All binary strings of length 3 that do not contain consecutive 1s.

Example 2:
Input: n = 2
Output: ["00", "01", "10"]
Explanation: All binary strings of length 2 that do not contain consecutive 1s.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(2^n), since each position has 2 choices. Space Complexity: O(n) per recursive path (due to

```cpp
#include <bits/stdc++.h>
using namespace std;

void generate(int n, string curr, vector<string>& result) {
    // Base case: if length is n, add to result
    if (curr.length() == n) {
        result.push_back(curr);
        return;
    }

    // Always try adding '0'
    generate(n, curr + "0", result);

    // Add '1' only if previous char is not '1'
    if (curr.empty() || curr.back() != '1') {
        generate(n, curr + "1", result);
    }
}
```

Q129. Generate Parentheses

MEDIUM

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Example 1:
Input: n = 3
Output: ["((()))", "(()())", "(())()", "()(())", "()()()"]

Example 2:
Input: n = 1
Output: ["()"]

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(2^n) (Catalan number): C(n) = (2n)! / (n!(n+1)!) is the number of valid sequences. Each seq

```cpp
#include <bits/stdc++.h>
using namespace std;

void backtrack(string curr, int open, int close, int n, vector<string>& res) {
    if (curr.length() == 2 * n) {
        res.push_back(curr);
        return;
    }
    if (open < n) backtrack(curr + '(', open + 1, close, n, res);
    if (close < open) backtrack(curr + ')', open, close + 1, n, res);
}

vector<string> generateParenthesis(int n) {
    vector<string> res;
    backtrack("", 0, 0, n, res);
    return res;
}
```

Q130. Power Set

MEDIUM

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Input: str = "abc"
Output: [a, ab, abc, ac, b, bc, c]
Explanation: Given string has 7 subsequences.

Input: str = "aa"
Output: [a, a, aa]
Explanation: Given string has 3 subsequences.

[BETTER] Approach - 2

Complexity: *Time Complexity: O(n * 2^n), for each subsequence, we construct and print the entire*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Solution class to generate all subsequences
  using recursion
class Solution {
public:
    // Helper recursive function to generate
      subsequences
    void helper(string &s, int index, string
      &t, vector<string> &result) {
        // Base case: If index reaches string
          length, add current subsequence to
          result
        if (index == s.size()) {
            result.push_back(current);
            return;
        }

        // Exclude current character and
          recurse
        helper(s, index + 1, current, result);

        // Include current character and
          recurse
        current.push_back(s[index]);
        helper(s, index + 1, current, result);

        // Backtrack: remove last character
          before returning to previous call
        current.pop_back();
    }

    // Function to return all subsequences of
      string s
    vector<string> getSubsequences(string s) {
        // Vector to store all subsequences
        vector<string> result;
        // Current subsequence being built
        string current = "";
        helper(s, 0, current, result);
        return result;
    }
};
```

[OPTIMAL] Approach - 1

Complexity: *Time Complexity: O(n * 2^n), for each subsequence, we may check up to n bits to decid*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Solution class to generate all subsequences
  using bit manipulation
class Solution {
public:
    // Function to return all subsequences of
      string s
    vector<string> getSubsequences(string s) {
        // Length of input string
        int n = s.size();

        // Total subsequences = 2^n
        int total = 1 << n;

        // Vector to store all subsequences
        vector<string> subsequences;

        // Iterate over all bit masks from 0
          to 2^n - 1
        for (int mask = 0; mask < total;
          mask++) {
            // Temporary subsequence string
            string subseq = "";

            // Check each bit position in mask
            for (int i = 0; i < n; i++) {
                // If i-th bit of mask is set,
                  include s[i]
                if (mask & (1 << i)) {
                    subseq += s[i];
                }
            }

            // Store the formed subsequence
            subsequences.push_back(subseq);
        }

        // Return all generated subsequences
        return subsequences;
    }
};
```

Q131. Learn All Patterns of Subsequences (Theory)

EASY

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Refer to the code implementation and logic below.

[OPTIMAL SOLUTION] Snippet 1 — N/A (Conceptual Tutorial)

```
function recurse(index, current):
    if index == n:
        print current
        return
    // Pick the current element
    recurse(index + 1, current + arr[index])
    // Do not pick the current element
    recurse(index + 1, current)
```

Q132. Count all subsequences with sum K

EASY

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Example 1:
Input : nums = [4, 9, 2, 5, 1] , k = 10
Output : 2
Explanation : The possible subsets with sum k are [9, 1] , [4, 5, 1].

Example 2:
Input : nums = [4, 2, 10, 5, 1, 3] , k = 5
Output : 3
Explanation : The possible subsets with sum k are [4, 1] , [2, 3] , [5].

[OPTIMAL SOLUTION] Approach — Time Complexity: O(2^n), where n is the number of elements in the array. This is because each element can eith

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
private:
    // Helper function to count subsequences
    // with the target sum
    int func(int ind, int sum, vector<int> &nums) {
        // Base case: if sum is 0, one valid
        // subsequence is found
        if (sum == 0) return 1;
        // Base case: if sum is negative or
        // index exceeds array size
        if (sum < 0 || ind == nums.size()) return 0;
        // Recurse by including current number
        // or excluding it from the sum
        return func(ind + 1, sum - nums[ind], nums) + func(ind + 1, sum,
          nums);
    }

public:
    // Function to start counting subsequences
    int countSubsequenceWithTargetSum(vector<int>& nums, int target) {
        return func(0, target, nums);
    }
};

// Main function to test the solution
```

Q133. Check if there exists a subsequence with sum K

EASY

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Example 1:
Input : nums = [1, 2, 3, 4, 5] , k = 8
Output : Yes
Explanation : The subsequences like [1, 2, 5] , [1, 3, 4] , [3, 5] sum up to 8.

Example 2:
Input : nums = [4, 3, 9, 2] , k = 10
Output : No
Explanation : No subsequence can sum up to 10.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(2^n), where n is the number of elements in the input array. This is because each element ca

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
private:
    // This method recursively checks for the subsequence with the given sum
    bool func(int ind, int sum, std::vector<int> &nums) {
        // Base case: if all elements are processed, check if sum is 0
        if (ind == nums.size()) {
            return sum == 0;
        }
        // Recursive call: include the current element in the subsequence
        // or exclude the current element from the subsequence
        return func(ind + 1, sum - nums[ind], nums) | func(ind + 1, sum,
          nums);
    }

public:
    // This method initiates the recursive process
    bool checkSubsequenceSum(std::vector<int>& nums, int target) {
        return func(0, target, nums); // Start the recursive process
    }
};

// Main function to test the solution
```

Q134. Combination Sum

MEDIUM

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Example 1:
Input: array = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation: 2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.

Example 2:
Input: array = [2], target = 1
Output: []
Explaination: No combination is possible.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(2t * k) due to exploring all combinations up to the target with copying

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
  public:
    // Function to find all combinations of numbers that sum up to the target
    void findCombination(int ind, int target, vector<int>& arr,
      vector<vector<int>>& ans, vector<int>& ds) {
        // Base case: if we have considered all elements in the array
        if (ind == arr.size()) {
            // If the target is zero, we have found a valid combination
            if (target == 0) {
                ans.push_back(ds);  // Add the current combination to the
                  result
            }
            return;
        }

        // Recursive case: pick the element if it's less than or equal to the
          target
        if (arr[ind] <= target) {
            ds.push_back(arr[ind]);  // Add the current element to the
              combination
            findCombination(ind, target - arr[ind], arr, ans, ds);  //
              Continue with the same index to allow repeated elements
            ds.pop_back();  // Backtrack by removing the last added element
        }

        // Skip the current element and move to the next index
        findCombination(ind + 1, target, arr, ans, ds);
    }

  public:
    // Main function to get all combinations
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<vector<int>> ans;  // To store the result
        vector<int> ds;  // To store a current combination
        findCombination(0, target, candidates, ans, ds);  // Start the
          recursive search
        return ans;  // Return all valid combinations
    }
};
```

Q135. Combination Sum II

MEDIUM

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Example 1:
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]]
Explanation: These are the unique combinations whose sum is equal to target.

Example 2:
Input: candidates = [2,5,2,1,2], target = 5
Output: [[1,2,2],[5]]
Explanation: These are the unique combinations whose sum is equal to target.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(2n * k), For each of the 2n subsequences, storing takes O(k) time w

```cpp
#include<bits/stdc++.h>
using namespace std;

// Function to find all combinations of numbers that sum up to the target
void findCombination(int ind, int target, vector<int>& arr,
  vector<vector<int>>& ans, vector<int>& ds) {
    // Base case: If the target becomes 0, we found a valid combination
    if (target == 0) {
        ans.push_back(ds);  // Add the current combination to the result
        return;
    }

    // Loop through the elements starting from index 'ind'
    for (int i = ind; i < arr.size(); i++) {
        // Skip duplicates to avoid repeating combinations
        if (i > ind && arr[i] == arr[i - 1]) continue;

        // If the current element is greater than the remaining target, break
          the loop
        if (arr[i] > target) break;

        // Include the current element in the combination
        ds.push_back(arr[i]);

        // Recur with the updated target and next index (i + 1 to avoid
          repetition)
        findCombination(i + 1, target - arr[i], arr, ans, ds);

        // Backtrack by removing the last added element
        ds.pop_back();
    }
}

// Function to calculate all unique combinations that sum up to the target
vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
    sort(candidates.begin(), candidates.end());  // Sort the candidates to
      handle duplicates
    vector<vector<int>> ans;  // To store the final answer
    vector<int> ds;  // To store the current combination
    findCombination(0, target, candidates, ans, ds);  // Call the helper
      function
    return ans;  // Return all valid combinations
}
```

Q136. Subsets I

MEDIUM

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Input: N = 3, arr[] = {5,2,1}
Output: 0,1,2,3,5,6,7,8
Explanation: We have to find all the subset's sum and print them.in this case the generated subsets are [[], [1], [2], [2,1], [5], [5,1], [5,2]. [5,2,1],so the sums we get will be 0,1,2,3,5,6,7,8
Input: N... (cont.)

[BETTER] Recursive approach

Complexity: *Time Complexity: O(2n),Each element has two choices: include or exclude, leading to 2*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Helper function to generate subset sums
      recursively
    void findSums(int index, int currentSum,
      vector<int>& arr, vector<int>& sums) {
        // Base case: if we have considered
          all elements
        if (index == arr.size()) {
            sums.push_back(currentSum);
            return;
        }

        // Include current element
        findSums(index + 1, currentSum +
          arr[index], arr, sums);

        // Exclude current element
        findSums(index + 1, currentSum, arr,
          sums);
    }

    // Main function
    vector<int> subsetSums(vector<int>& arr) {
        vector<int> sums;
        findSums(0, 0, arr, sums);
        sort(sums.begin(), sums.end()); //
          Sort in increasing order
        return sums;
    }
};

// Driver code
```

[OPTIMAL] Bitmasking Approach

Complexity: *Time Complexity: O(2n * n),We generate all possible subsets, which is 2n. For each su*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find all subset sums using
      bitmasking
    vector<int> subsetSums(vector<int>& arr) {
        int n = arr.size();
        vector<int> sums;

        // Iterate through all possible
          bitmasks from 0 to 2^n - 1
        for (int mask = 0; mask < (1 << n);
          mask++) {
            int sum = 0; // store sum of
              current subset
            for (int i = 0; i < n; i++) {
                // If ith bit is set, include
                  arr[i] in sum
                if (mask & (1 << i)) {
                    sum += arr[i];
                }
            }
            sums.push_back(sum);
        }

        // Sort sums to get increasing order
        sort(sums.begin(), sums.end());
        return sums;
    }
};

// Driver code
```

Q137. Subsets II

MEDIUM

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Input: array[] = [1,2,2]
Output: [[ ],[1],[1,2],[1,2,2],[2],[2,2]]
Explanation: We can have subsets ranging from length 0 to 3. which are listed above. Also the subset [1,2] appears twice but is printed only once as we require only unique subsets.
Input: array[] = [1]
Output: [[ ], [1]]
Explanation: Only two unique subsets are available.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(2^N),In the worst case (all unique elements), we generate all possible subsets, which is 2^

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to generate all unique subsets
    void backtrack(int start, vector<int>&
      nums, vector<int>& current,
      vector<vector<int>>& result) {
        // Add the current subset to result
        result.push_back(current);

        // Iterate over the array from 'start'
          index
        for (int i = start; i < nums.size();
          i++) {
            // Skip duplicates: if current
              number is same as previous and
              not at the start index
            if (i > start && nums[i] == nums[i
              - 1]) continue;

            // Include nums[i] in current
              subset
            current.push_back(nums[i]);

            // Recurse for next index
            backtrack(i + 1, nums, current,
              result);

            // Backtrack: remove last added
              element
            current.pop_back();
        }
    }

    // Main function to get unique subsets
    vector<vector<int>>
      subsetsWithDup(vector<int>& nums) {
        // Sort the array to handle duplicates
        sort(nums.begin(), nums.end());

        vector<vector<int>> result;
        vector<int> current;

        // Start backtracking from index 0
        backtrack(0, nums, current, result);

        return result;
    }
};

// Driver code
```

Q138. Combination Sum III

MEDIUM

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Example 1:
Input: k = 3, n = 7
Output: [[1, 2, 4]]
Explanation:
1 + 2 + 4 = 7
There are no other valid combinations.

Example 2:
Input: k = 3, n = 9
Output: [[1, 2, 6],[1, 3, 5],[2, 3, 4]]
Explanation:
1 + 2 + 6 = 9
1 + 3 + 5 = 9
2 + 3 + 4 = 9
There are no other valid combinations.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(2^9 * k), due to the exploration of all subsets of the set {1, 2, ..., 9}..

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
private:
    // Recursive function to find combinations
    void func(int sum, int last, vector<int>
      &nums, int k, vector<vector<int>> &ans)
      {
        // If the sum is zero and the number
          of elements is k
        if(sum == 0 && nums.size() == k) {
            // Add the current combination to
              the answer
            ans.push_back(nums);
            return;
        }
        // If the sum is less than or equal to
          zero or the number of elements
          exceeds k
        if(sum <= 0 || nums.size() > k)
          return;

        // Iterate from the last number to 9
        for(int i = last; i <= 9; i++) {
            // If the current number is less
              than or equal to the sum
            if(i <= sum) {
                // Add the number to the
                  current combination
                nums.push_back(i);
                // Recursive call with updated
                  sum and next number
                func(sum - i, i + 1, nums, k,
                  ans);
                // Remove the last number to
                  backtrack
                nums.pop_back();
            } else {
                // If the number is greater
                  than the sum, break the loop
                break;
            }
        }
    }
public:
    // Function to find all possible
      combinations of k numbers that add up to
      n
    vector<vector<int>> combinationSum3(int k,
      int n) {
        vector<vector<int>> ans;
        vector<int> nums;
        // Call the recursive function with
          initial parameters
        func(n, 1, nums, k, ans);
        return ans;
    }
};
```

Q139. Letter Combinations of a Phone Number

HARD

Topic: Recursion [PatternWise] > Subsequences Pattern

Practice Link

Example 1:
Input: digits = "34"
Output: ["dg", "dh", "di", "eg", "eh", "ei", "fg", "fh", "fi"]
Explanation: The 3 is mapped with "def" and 4 is mapped with "ghi".
So all possible combinations by replacing the digits with characters are shown in the output.

Example 2:
Input: digits = "3"
Output: ["d", "e", "f"]
Explanation: The 3 is mapped with "def".

[OPTIMAL SOLUTION] Approach — Time Complexity: O(4^N * N), where n is the length of the input digits. This is because each digit can map to

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
private:
    // Recursive function to generate letter combinations
    void func(int ind, string digits, string s, vector<string> &ans, string
      combos[]) {
        // Base case: if index reaches the end of digits
        if(ind == digits.size()) {
            // Add the current combination to the answer
            ans.push_back(s);
            return;
        }
        // Convert the current character to an integer
        int digit = digits[ind] - '0';
        // Loop through the corresponding characters
        for(int i = 0; i < combos[digit].size(); i++) {
            // Recursively call function with next index
            // Add current character to the string
            func(ind + 1, digits, s + combos[digit][i], ans, combos);
        }
    }

public:
    // Function to get all letter combinations for a given digit string
    vector<string> letterCombinations(string digits) {
        // Mapping digits to corresponding characters
        string combos[] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs",
          "tuv", "wxyz"};
        vector<string> ans; // Vector to store results
        string s = ""; // Temporary string to build combinations
        // Initiate recursive function
        func(0, digits, s, ans, combos);
        return ans; // Return the result
    }
};
```

Q140. Word Search

HARD

Topic: Recursion [PatternWise] > Trying out all Combos / Hard

Practice Link

Input: [["A", "B", "C", "E"], ["S", "F", "C", "S"]["A", "D", "E", "E"]] word = "ABCCED"
Output: true
Explanation: We can easily find the given word in the matrix.
Input:[["A", "B", "C", "E"],["S", "F", "C", "S"],["A", "D", "E", "E"]]word = "ABCB"
Output: false
Explanation: There is no such word in the given matrix.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(m * n * 4^L),We may start from each of the m×n cells, and explore up to 4 directions for ea

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to check if the word exists in
      the board
    bool exist(vector<vector<char>>& board,
      string word) {
        // Get number of rows
        int rows = board.size();
        // Get number of columns
        int cols = board[0].size();
        // Loop through every cell in the grid
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                // Start DFS if first letter
                  matches
                if (dfs(board, word, i, j, 0))
                  {
                    return true;
                }
            }
        }
        // If no path found, return false
        return false;
    }
private:
    // DFS function to search for the word
    bool dfs(vector<vector<char>>& board,
      string& word, int i, int j, int idx) {
        // If all characters matched, return
          true
        if (idx == word.size()) return true;
        // Check boundaries and character
          match
        if (i < 0 || j < 0 || i >=
          board.size() || j >= board[0].size()
          || board[i][j] != word[idx]) {
            return false;
        }
        // Store current character and mark as
          visited
        char temp = board[i][j];
        board[i][j] = '#';
        // Explore all four directions
        bool found = dfs(board, word, i + 1,
          j, idx + 1) ||
                     dfs(board, word, i - 1,
                       j, idx + 1) ||
                     dfs(board, word, i, j +
                       1, idx + 1) ||
                     dfs(board, word, i, j -
                       1, idx + 1);
        // Restore the character
          (backtracking)
        board[i][j] = temp;
        return found;
    }
};
```

Q141. N Queen

HARD

Topic: Recursion [PatternWise] > Trying out all Combos / Hard

Practice Link

Input: N = 4
Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown below

Input : N = 1
Output: [["Q"]]
Explanation : There is only one way to place 1 queen on 1x1 chessboard.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N!), we try all possible permutations of placing the queens. Sp

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to solve N-Queens problem
    void solve(int col, vector<string>& board,
      int n,
               vector<int>& leftRow,
                 vector<int>& upperDiagonal,
                 vector<int>& lowerDiagonal,
               vector<vector<string>>& ans) {
        // If all queens are placed
        if (col == n) {
            ans.push_back(board);
            return;
        }
        // Iterate through all rows
        for (int row = 0; row < n; row++) {
            // Check if it's safe to place the
              queen
            if (leftRow[row] == 0 &&
              lowerDiagonal[row + col] == 0 &&
                upperDiagonal[n - 1 + col -
                  row] == 0) {
                // Place the queen
                board[row][col] = 'Q';
                // Mark the row and diagonals
                leftRow[row] = 1;
                lowerDiagonal[row + col] = 1;
                upperDiagonal[n - 1 + col -
                  row] = 1;
                // Recurse to next column
                solve(col + 1, board, n,
                  leftRow, upperDiagonal,
                  lowerDiagonal, ans);
                // Backtrack and remove the
                  queen
                board[row][col] = '.';
                leftRow[row] = 0;
                lowerDiagonal[row + col] = 0;
                upperDiagonal[n - 1 + col -
                  row] = 0;
            }
        }
    }
    // Main function
    vector<vector<string>> solveNQueens(int n)
      {
        vector<vector<string>> ans;
        vector<string> board(n, string(n,
          '.'));
        vector<int> leftRow(n, 0),
          upperDiagonal(2 * n - 1, 0),
          lowerDiagonal(2 * n - 1, 0);
        solve(0, board, n, leftRow,
          upperDiagonal, lowerDiagonal, ans);
        return ans;
    }
};
```

Q142. Rat in a Maze

HARD

Topic: Recursion [PatternWise] > Trying out all Combos / Hard

Practice Link

Input: n = 4 , grid = [[1, 0, 0, 0] , [1, 1, 0, 1], [1, 1, 0, 0], [0, 1, 1, 1]]
Output: ["DDRDRR" , "DRDDRR"]
Explanation: The rat has two different path to reach (3, 3).
The first path is (0, 0) => (1, 0) => (2, 0) => (2, 1) => (3, 1) => (3, 2) => (3, 3).
The second path is (0,0) => (1,0) => (1,1) => (2,1) => (3,1) => (3,2) => (3,3).

Input: n = 2 , grid = [[1, 0] , [1, 0]]
Output: []
Explanation: There is no path that rat can choose to travel from (0,0) to (1,1).

[OPTIMAL SOLUTION] Approach — Time Complexity: O(4^(N*N)),, because on every cell we need to try 4 different directions.

```cpp
#include <bits/stdc++.h>
using namespace std;
// Class to solve the Rat in a Maze problem
class Solution {
public:
    // Function to check if the cell is within
      maze and valid to move
    bool isSafe(int x, int y, int n,
      vector<vector<int>> &maze,
                vector<vector<int>> &visited)
                  {
        return (x >= 0 && x < n && y >= 0 && y
          < n &&
                maze[x][y] == 1 &&
                  visited[x][y] == 0);
    }
    // Function to solve maze using
      backtracking
    void solve(int x, int y, int n,
      vector<vector<int>> &maze,
               vector<vector<int>> &visited,
                 string path,
               vector<string> &res) {
        // If destination reached, store the
          path
        if (x == n - 1 && y == n - 1) {
            res.push_back(path);
            return;
        }
        // Mark the cell visited
        visited[x][y] = 1;
        // Try moving Down
        if (isSafe(x + 1, y, n, maze,
          visited)) {
            solve(x + 1, y, n, maze, visited,
              path + "D", res);
        }
        // Try moving Left
        if (isSafe(x, y - 1, n, maze,
          visited)) {
            solve(x, y - 1, n, maze, visited,
              path + "L", res);
        }
        // Try moving Right
        if (isSafe(x, y + 1, n, maze,
          visited)) {
            solve(x, y + 1, n, maze, visited,
              path + "R", res);
        }
        // Try moving Up
        if (isSafe(x - 1, y, n, maze,
          visited)) {
            solve(x - 1, y, n, maze, visited,
              path + "U", res);
        }
        // Backtrack: unmark cell as visited
        visited[x][y] = 0;
    }
    // Main function to find all paths
    vector<string>
      findPath(vector<vector<int>> &maze, int
      n) {
        vector<string> res;
        vector<vector<int>> visited(n,
          vector<int>(n, 0));
        if (maze[0][0] == 1) {
            solve(0, 0, n, maze, visited, "",
              res);
        }
        return res;
    }
};
// Driver code
```

Q143. M Coloring Problem

HARD

Topic: Recursion [PatternWise] > Trying out all Combos / Hard

Practice Link

Example 1:
Input:
N = 4
M = 3
E = 5
Edges[] = {
(0, 1),
(1, 2),
(2, 3),
(3, 0),
(0, 2)
}
Output: 1
Explanation:
It is possible to color the given graph using 3 colors, so the answer is 1 (possible).

Example 2:
Input:
N = 3
M = 2
E = 3
Edges[] = {
(0, 1),
(1, 2),
(0, 2)
}
Output: 0
Explanation:
It is not possible to color the graph using 2 colors as it forms a triangle, which requires at least 3 colors. Hence, the answer is 0 (n... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(m^N), where m is the number of colors and N is the number of nodes in the graph. This is be

```cpp
#include<bits/stdc++.h>
using namespace std;

// Function to check if it is safe to assign a
  color to a node
bool isSafe(int node, int color[], bool
  graph[101][101], int n, int col) {
    for (int k = 0; k < n; k++) {
        // Check if adjacent node has the same
          color
        if (k != node && graph[k][node] == 1
          && color[k] == col) {
            return false;
        }
    }
    return true;  // Safe to assign the color
}

// Recursive function to solve the coloring
  problem
bool solve(int node, int color[], int m, int
  N, bool graph[101][101]) {
    // If all nodes are assigned colors,
      return true
    if (node == N) {
        return true;
    }
    // Try different colors for the node
    for (int i = 1; i <= m; i++) {
        if (isSafe(node, color, graph, N, i))
          {
            color[node] = i;
            // Recursively check for the next
              node
            if (solve(node + 1, color, m, N,
              graph)) return true;
            color[node] = 0;  // Backtrack if
              the color assignment fails
        }
    }
    return false;  // If no solution is found
}

// Function to check if graph can be colored
  with m colors
bool graphColoring(bool graph[101][101], int
  m, int N) {
    int color[N] = { 0 };
    // Start solving from node 0
    if (solve(0, color, m, N, graph)) return
      true;
    return false;
}
```

Q144. Sudoku Solver

HARD

Topic: Recursion [PatternWise] > Trying out all Combos / Hard

Practice Link

Input :

Output :

Explanation : The output is a valid Sudoku solution that fills in all empty cells such that each number 1 through 9 appears exactly once in every row, column, and 3x3 sub-box.
Input :

Output :

Explanation : The output is one of the possible valid Sudoku solutions that fills all empty cells following the Sudoku rules.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(9(n ^ 2)), in the worst case, for each cell in the n2 board, we have 9 possible numbers. Sp

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
bool isValid(vector<vector<char>> &board, int
  row, int col, char c) {
    for (int i = 0; i < 9; i++) {
        if (board[i][col] == c)
            return false;
    }
    for (int j = 0; j < 9; j++) {
        if (board[row][j] == c)
            return false;
    }
    int boxRowStart = 3 * (row / 3);
    int boxColStart = 3 * (col / 3);
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[boxRowStart +
              i][boxColStart + j] == c)
                return false;
        }
    }
    return true;
}
bool solveSudoku(vector<vector<char>> &board)
  {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (board[i][j] == '.') {
                for (char c = '1'; c <= '9';
                  c++) {
                    if (isValid(board, i, j,
                      c)) {
                        board[i][j] = c;
                        if
                          (solveSudoku(board))
                            return true;
                        board[i][j] = '.';
                    }
                }
                return false;
            }
        }
    }
    return true;
}
};
```

Q145. Expression Add Operators

HARD

Topic: Recursion [PatternWise] > Trying out all Combos / Hard

Practice Link

Example 1:
Input: num = "123", target = 6
Output: ["1*2*3","1+2+3"]
Explanation: Both "1*2*3" and "1+2+3" evaluate to 6.

Example 2:
Input: num = "232", target = 8
Output: ["2*3+2","2+3*2"]
Explanation: Both "2*3+2" and "2+3*2" evaluate to 8.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(4^n), since in each recursive call, we can choose 4 possibilities for each substring (three

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to evaluate the expression
    void dfs(string& num, int target, int
      start, long long current_value, long
      long last_operand, string expression,
      vector<string>& result) {
        // Base case: If we've reached the end
          of the string
        if (start == num.size()) {
            // If the expression evaluates to
              the target, add it to result
            if (current_value == target)
                result.push_back(expression);
            return;
        }
        // Loop through all substrings
          starting from 'start' index
        for (int i = start; i < num.size();
          i++) {
            // Skip leading zeros in numbers
            if (i > start && num[start] ==
              '0') return;
            // Get the current number
            string current_num =
              num.substr(start, i - start +
              1);
            long long current_num_val =
              stoll(current_num);
            // If we are at the first number,
              just start the expression
            if (start == 0) {
                dfs(num, target, i + 1,
                  current_num_val,
                  current_num_val,
                  current_num, result);
            } else {
                // Add the current number with
                  '+'
                dfs(num, target, i + 1,
                  current_value +
                  current_num_val,
                  current_num_val, expression
                  + "+" + current_num,
                  result);
                // Add the current number with
                  '-'
                dfs(num, target, i + 1,
                  current_value -
                  current_num_val,
                  -current_num_val, expression
                  + "-" + current_num,
                  result);
                // Add the current number with
                  '*'
                dfs(num, target, i + 1,
                  current_value - last_operand
                  + last_operand *
                  current_num_val,
                  last_operand *
                  current_num_val, expression
                  + "*" + current_num,
                  result);
            }
        }
    }
    // To store the valid expressions
    vector<string> addOperators(string num,
      int target) {
        vector<string> result;
        // Start DFS with empty expression
        dfs(num, target, 0, 0, 0, "", result);
        return result;
    }
};
```

Q146. Introduction to Bits and Tricks

EASY

Topic: Bit Manipulation [Concepts & Problems] > Learn Bit Manipulation

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```
1 * 2^0 = 1
0 * 2^1 = 0
1 * 2^2 = 4
1 * 2^3 = 8
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```
13 / 2 = 6 remainder 1
6 / 2 = 3 remainder 0
3 / 2 = 1 remainder 1
1 / 2 = 0 remainder 1
```

Q147. Check if the i-th bit is Set or Not

EASY

Topic: Bit Manipulation [Concepts & Problems] > Learn Bit Manipulation

Practice Link

Example 1:
Input: n = 5, i = 0
Output: true
Explanation: Binary representation of 5 is 101. The 0-th bit from LSB is set (1).

Example 2:
Input: n = 10, i = 1
Output: true
Explanation: Binary representation of 10 is 1010. The 1-st bit from LSB is set (1).

[OPTIMAL SOLUTION] Optinal Approach — Time Complexity: O(1), constant time bitwise operation. Space Complexity: O(1), no ad

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to check if the i-th bit of number n is set (1)
    bool checkIthBit(int n, int i) {
        // Check if the i-th bit is set using bitwise AND operation
        return (n & (1 << i)) != 0;  // If the i-th bit is 1, the result will
          be non-zero
    }
};
```

Q148. Check if a Number is Odd or Not

EASY

Topic: Bit Manipulation [Concepts & Problems] > Learn Bit Manipulation

Practice Link

Example 1:
Input: n = 7
Output: true
Explanation: 7 is not divisible by 2. Hence, it is odd.

Example 2:
Input: n = 10
Output: false
Explanation: 10 is divisible by 2. Hence, it is not odd.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1) — The modulus operation takes constant time. Space Complexity: O(1)

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to check if a number is odd
    bool isOdd(int n) {
        return (n % 2 != 0);  // Return true if the number is odd, else false
    }
};
```

Q149. Check if a Number is Power of 2 or Not

EASY

Topic: Bit Manipulation [Concepts & Problems] > Learn Bit Manipulation

Practice Link

Example 1:
Input: n = 16
Output: true
Explanation: 2■ = 16, so 16 is a power of two.

Example 2:
Input: n = 3
Output: false
Explanation: 3 is not a power of two.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1), because bitwise operations take constant time. Space Complexity: O

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to check if a number is a power of two
    bool isPowerOfTwo(int n) {
        return n > 0 && (n & (n - 1)) == 0;  // Check if n is greater than 0
          and has only one bit set
    }
};
```

Q150. Count the Number of Set Bits

EASY

Topic: Bit Manipulation [Concepts & Problems] > Learn Bit Manipulation

Practice Link

Example 1:
Input: n = 5
Output: 2
Explanation: The binary representation of 5 is 101, which has 2 set bits.

Example 2:
Input: n = 15
Output: 4
Explanation: The binary representation of 15 is 1111, which has 4 set bits.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(k), where k is the number of set bits (often faster than checking all bits). Space Complexi

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to count the number of set bits (1s) in the binary
      representation of n using Brian Kernighan's Algorithm
    int countSetBits(int n) {
        int count = 0;  // Variable to store the count of set bits

        // Step 1: While n is non-zero, turn off the rightmost set bit
        while (n) {
            n &= (n - 1);  // Turn off the rightmost set bit
            count++;  // Increment the count
        }

        // Step 2: Return the count of set bits
        return count;
    }
};

// Main function to test the solution
```

Q151. Set/Unset the rightmost unset bit

EASY

Topic: Bit Manipulation [Concepts & Problems] > Learn Bit Manipulation

Practice Link

Example 1:
Input: n = 10 (binary: 1010)
Output: 11 (binary: 1011)
Explanation: The rightmost unset bit is the least significant bit (LSB). Setting it to 1 gives 1011 = 11.

Example 2:
Input: n = 7 (binary: 111)
Output: 7 (binary: 111)
Explanation: All bits are already set to 1, so the number remains the same.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1) since only one bitwise operation is performed. Space Complexity: O(1) since no extra spa

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to set the rightmost unset bit (0 -> 1)
int setRightmostUnsetBit(int n) {
    // Use bitwise OR with n+1 to set rightmost 0
    return n | (n + 1);
}
```

Q152. Swap Two Numbers

EASY

Topic: Bit Manipulation [Concepts & Problems] > Learn Bit Manipulation

Practice Link

Example 1:
Input: a = 5, b = 10
Output: a = 10, b = 5

Example 2:
Input: a = -100, b = -200
Output: a = -200, b = -100

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1) Constant operations. Space Complexity: O(1) No extra space used.

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to swap using XOR bit manipulation
void swapXOR(int &a, int &b) {
    // Step 1: XOR a and b, store in a
    a = a ^ b;

    // Step 2: XOR new a with b, result is original a → store in b
    b = a ^ b;

    // Step 3: XOR new a with new b, result is original b → store in a
    a = a ^ b;
}
```

Q153. Divide two numbers without multiplication and division

MEDIUM

Topic: Bit Manipulation [Concepts & Problems] > Learn Bit Manipulation

Practice Link

Example 1:
Input: Dividend = 10, Divisor = 3
Output: 3
Explanation: 10/3 = 3.33, truncated to 3.

Example 2:
Input: Dividend = 7, Divisor = -3
Output: -2
Explanation: 7/-3 = -2.33, truncated to -2.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O((logN)^2) – (where N is the absolute value of dividend). The outer loop runs for O(logN) ti

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    int divide(int dividend, int divisor) {
        // Base case
        if(dividend == divisor) return 1;
        if(dividend == INT_MIN && divisor == -1) return INT_MAX;
        if(divisor == 1) return dividend;
        // Variable to store the sign of result
        bool isPositive = true;
        // Updating the sign of quotient
        if(dividend >= 0 && divisor < 0)
            isPositive = false;
        else if(dividend < 0 && divisor > 0)
            isPositive = false;
        // Storing absolute dividend & divisor
        long long n = dividend;
        long long d = divisor;
        n = abs(n);
        d = abs(d);
        // Variable to store the answer and sum
        long long ans = 0, sum = 0;
        while(sum + d <= n) {
            // Increment the count
           ans++;
           // Update the sum
           sum += d;
        }
        // Handling overflowing condition
        if(ans > INT_MAX && isPositive)
            return INT_MAX;
        if(ans > INT_MAX && !isPositive)
            return INT_MIN;
        return isPositive ? ans : -1*ans;
    }
};
```

Q154. Minimum Bit Flips to Convert Number

MEDIUM

Topic: Bit Manipulation [Concepts & Problems] > Interview Problems

Practice Link

Example 1:
Input : start = 10 , goal = 7
Output : 3
Explanation : The binary representation of 10 is "1010". The binary representation of 7 is "111". If we flip the underlined bits in binary representation of 10 then we will obtain our goal.

Example 2 :
Input : start = 3 , goal = 4
Output : 3
Explanation : The binary representation of 3 is "011".The binary representation of 4 is "100". So if we flip all the three bits of 3 then we will reach our goal number.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1), The XOR operation between two integers is performed in constant time, O(1). The for loo

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    /* Function to get the minimum
     bit flips to convert number */
    int minBitsFlip(int start, int goal) {

        /* Variable to store bits that
        are different in both numbers */
        int num = start ^ goal;

        /* Variable to count
        number of set bits */
        int count = 0;

        for(int i = 0; i < 32; i++) {
            /* Update count if the
            rightmost bit is set */
            count += (num & 1);

            /* Shift the number every
            time by 1 place */
            num = num >> 1;
        }
        return count;
    }
};
```

Q155. Single Number - I

MEDIUM

Topic: Bit Manipulation [Concepts & Problems] > Interview Problems

Practice Link

Example 1:
Input Format: arr[] = {2,2,1}
Result: 1
Explanation: In this array, only the element 1 appear once and so it is the answer.

Example 2:
Input Format: arr[] = {4,1,2,1,2}
Result: 4
Explanation: In this array, only element 4 appear once and the other... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N)+O(N)+O(N), where N = size of the array. One O(N) is for finding*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the single
      non-repeating element using a hash array
    int getSingleElement(vector<int>& arr) {
        int n = arr.size();

        // Step 1: Find the maximum value in
          the array
        int maxi = arr[0];
        for (int i = 0; i < n; i++) {
            maxi = max(maxi, arr[i]);
        }

        // Step 2: Declare and initialize hash
          array of size maxi+1
        vector<int> hash(maxi + 1, 0);

        // Step 3: Count the frequency of each
          number
        for (int i = 0; i < n; i++) {
            hash[arr[i]]++;
        }

        // Step 4: Find the element that
          occurs exactly once
        for (int i = 0; i < n; i++) {
            if (hash[arr[i]] == 1)
                return arr[i];
        }

        return -1; // fallback, shouldn't
          occur in valid input
    }
};
```

[OPTIMAL] Optimised Approach

Complexity: *Time Complexity: O(N). Where N is the size of the array Space*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the single
      non-repeating element using XOR
    int getSingleElement(vector<int>& arr) {
        int n = arr.size();
        int xorr = 0;

        // XOR all elements. Duplicates cancel
          out, leaving the single element.
        for (int i = 0; i < n; i++) {
            xorr = xorr ^ arr[i];
        }

        return xorr;
    }
};
```

Q156. Power Set Bit Manipulation

MEDIUM

Topic: Bit Manipulation [Concepts & Problems] > Interview Problems

Practice Link

Input: nums = [1, 2, 3]
Output: [[ ], [1], [2], [3], [1, 2], [2, 3], [3, 1], [1, 2, 3]]
Explanation: A power set of an array is an array of all possible subarrays, including an empty array. It contains all combinations of elements from the original array. By iterating through all possible combinations of the elements in the input array we are able to get the power set of the array.
Input: nums = [5, 7, 8]
Output: [[ ], [5], [7], [8], [5, 7], [7, 8], [8, 5], [5, 7, 8]]
Explanation: A power set... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N x 2N) where N is the number of elements in the input array.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to generate all subsets (the
      power set) of the input array
    vector<vector<int>>
      getPowerSet(vector<int>& nums) {
        // Get the size of the input array
        int n = nums.size();

        // Calculate the total number of
          subsets (2^n) using bitwise shift
        int subsets = 1 << n;

        // Vector to store all subsets
        vector<vector<int>> ans;

        // Iterate through all numbers from 0
          to 2^n - 1
        for (int num = 0; num < subsets;
          num++) {
            // Temporary vector to hold the
              current subset
            vector<int> subset;

            // Iterate through each bit of the
              number
            for (int i = 0; i < n; i++) {
                // If the ith bit is set,
                  include nums[i] in the
                  subset
                if (num & (1 << i)) {
                    subset.push_back(nums[i]);
                }
            }

            // Add the constructed subset into
              the result
            ans.push_back(subset);
        }

        // Return all subsets
        return ans;
    }
};

// Driver code
```

Q157. XOR of numbers in a given range

MEDIUM

Topic: Bit Manipulation [Concepts & Problems] > Interview Problems

Practice Link

Example 1:
Input : L = 3 , R = 5
Output : 2
Explanation : answer = (3 ^ 4 ^ 5) = 2.

Example 2:
Input : L = 1, R = 3
Output : 0
Explanation : answer = (1 ^ 2 ^ 3) = 0.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(1) Using constant time operations. Space Complexity: O(1) Using a c

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
private:
    /* Function to find the XOR
    of numbers from 1 to n*/
    int XORtillN(int n) {
        if(n % 4 == 1) return 1;
        if(n % 4 == 2) return n+1;
        if(n % 4 == 3) return 0;
        return n;
    }

public:
    /* Function to find the XOR
    of numbers from L to R*/
    int findRangeXOR(int l, int r){
    return XORtillN(l-1) ^ XORtillN(r);
    }
};
```

Q158. Single Number - III

MEDIUM

Topic: Bit Manipulation [Concepts & Problems] > Interview Problems

Practice Link

Example 1:
Input: nums = [1, 2, 1, 3, 5, 2]
Output: [3, 5]
Explanation:
The integers 3 and 5 have appeared only once.

Example 2:
Input: nums = [-1, 0]
Output: [-1, 0]
Explanation:
The integers -1 and 0 have appeared only once.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), traversing the array twice results in O(2*N) time complexity. Space Complexity: O(1),

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    /* Function to get the single
    numbers in the given array */
    vector<int> singleNumber(vector<int>&
      nums){
        // Variable to store size of array
        int n = nums.size();

        // Variable to store XOR of all
          elements
        long XOR = 0;

        // Traverse the array
        for(int i=0; i < n; i++) {

            // Update the XOR
            XOR = XOR ^ nums[i];
        }

        /* Variable to get the rightmost
        set bit in overall XOR */
        int rightmost = (XOR & (XOR - 1)) ^
          XOR;

        /* Variables to stores XOR of
        elements in bucket 1 and 2 */
        int XOR1 = 0, XOR2 = 0;

        // Traverse the array
        for(int i=0; i < n; i++) {

            /* Divide the numbers among bucket
              1
             and 2 based on rightmost set bit
              */
            if(nums[i] & rightmost) {
                XOR1 = XOR1 ^ nums[i];
            }
            else {
                XOR2 = XOR2 ^ nums[i];
            }
        }

        // Return the result in sorted order
        if(XOR1 < XOR2) return {XOR1, XOR2};
        return {XOR2, XOR1};
    }
};
```

Q159. Print Prime Factors of a Number

HARD

Topic: Bit Manipulation [Concepts & Problems] > Advanced Maths

Practice Link

Example 1:
Input: nums = [1, 2, 1, 3, 5, 2]
Output: [3, 5]
Explanation:
The integers 3 and 5 have appeared only once.

Example 2:
Input: nums = [-1, 0]
Output: [-1, 0]
Explanation:
The integers -1 and 0 have appeared only once.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), traversing the array twice results in O(2*N) time complexity. Space Complexity: O(1),

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    /* Function to get the single
    numbers in the given array */
    vector<int> singleNumber(vector<int>&
      nums){
        // Variable to store size of array
        int n = nums.size();

        // Variable to store XOR of all
          elements
        long XOR = 0;

        // Traverse the array
        for(int i=0; i < n; i++) {

            // Update the XOR
            XOR = XOR ^ nums[i];
        }

        /* Variable to get the rightmost
        set bit in overall XOR */
        int rightmost = (XOR & (XOR - 1)) ^
          XOR;

        /* Variables to stores XOR of
        elements in bucket 1 and 2 */
        int XOR1 = 0, XOR2 = 0;

        // Traverse the array
        for(int i=0; i < n; i++) {

            /* Divide the numbers among bucket
              1
             and 2 based on rightmost set bit
              */
            if(nums[i] & rightmost) {
                XOR1 = XOR1 ^ nums[i];
            }
            else {
                XOR2 = XOR2 ^ nums[i];
            }
        }

        // Return the result in sorted order
        if(XOR1 < XOR2) return {XOR1, XOR2};
        return {XOR2, XOR1};
    }
};
```

Q160. Divisors of a Number

EASY

Topic: Bit Manipulation [Concepts & Problems] > Advanced Maths

Practice Link

Input: N = 36
Output: [1, 2, 3, 4, 6, 9, 12, 18, 36]
Explanation: The divisors of 36 are 1, 2, 3, 4, 6, 9, 12, 18, 36.

Input: N = 12
Output: [1, 2, 3, 4, 6, 12]
Explanation: The divisors of 12 are 1, 2, 3, 4, 6, 12.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(sqrt(N)), we check for every number between 1 and sqaure root of N.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to get all divisors
    vector<int> getDivisors(int N) {
        // Create a vector to store divisors
        vector<int> res;

        // Loop from 1 to square root of N
        for (int i = 1; i * i <= N; i++) {
            // Check if i divides N
            if (N % i == 0) {
                // Add i to the result
                res.push_back(i);

                // If N / i is different from i, add N / i too
                if (i != N / i) {
                    res.push_back(N / i);
                }
            }
        }

        // Return the list of divisors
        return res;
    }
};
```

Q161. Count primes in range L to R

HARD

Topic: Bit Manipulation [Concepts & Problems] > Advanced Maths

Practice Link

Example 1:
Input: queries = [[2, 5], [4, 7]]
Output: [3, 2]
Explanation:
The range 2 to 5 contains three prime numbers: 2, 3, 5.
The range 4 to 7 contains two prime numbers: 5, 7.

Example 2:
Input: queries = [[1, 7], [3, 7]]
Output: [4, 3]
Explanation:
The range 1 to 7 contains four prime numbers: 2, 3, 5, 7.
The range 3 to 7 contains three prime numbers: 3, 5, 7.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N log (log N)) for the Sieve of Eratosthenes, where n is the maximum value in the queries.

```cpp
#include<bits/stdc++.h>
using namespace std;
class Solution {
public:
    std::vector<int>
      primesInRange(std::vector<std::vector<in
      t>>& queries) {
        if (queries.empty()) {
            return {};
        }
        // Find the maximum value in the
          queries
        // to determine the sieve range
        int maxVal = 0;
        for (const auto& query : queries) {
            maxVal = std::max(maxVal,
              query[1]);
        }
        // Step 1: Use the Sieve of
          Eratosthenes
        // to find all primes up to maxVal
        std::vector<bool> isPrime(maxVal + 1,
          true);
        isPrime[0] = isPrime[1] = false;  // 0
          and 1 are not primes
        for (int p = 2; p * p <= maxVal; ++p)
          {
            if (isPrime[p]) {
                for (int i = p * p; i <=
                  maxVal; i += p) {
                    isPrime[i] = false;
                }
            }
        }
        // Step 2: Create a prefix sum array
        // to count primes up to each number
        std::vector<int> primeCount(maxVal +
          1, 0);
        for (int i = 1; i <= maxVal; ++i) {
            primeCount[i] = primeCount[i - 1];
            if (isPrime[i]) {
                primeCount[i]++;
            }
        }
        // Step 3: Process each query to find
          the number of primes
        // in the given range
        std::vector<int> result;
        for (const auto& query : queries) {
            int start = query[0];
            int end = query[1];
            if (start == 0) {
result.push_back(primeCount[end]);
            } else {
result.push_back(primeCount[end] -
                    primeCount[start - 1]);
            }
        }
        return result;
    }
};
```

Q162. Prime factorisation of a Number

HARD

Topic: Bit Manipulation [Concepts & Problems] > Advanced Maths

Practice Link

Example 1:
Input: nums = [1, 2, 1, 3, 5, 2]
Output: [3, 5]
Explanation:
The integers 3 and 5 have appeared only once.

Example 2:
Input: nums = [-1, 0]
Output: [-1, 0]
Explanation:
The integers -1 and 0 have appeared only once.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), traversing the array twice results in O(2*N) time complexity. Space Complexity: O(1),

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    /* Function to get the single
    numbers in the given array */
    vector<int> singleNumber(vector<int>&
      nums){
        // Variable to store size of array
        int n = nums.size();

        // Variable to store XOR of all
          elements
        long XOR = 0;

        // Traverse the array
        for(int i=0; i < n; i++) {

            // Update the XOR
            XOR = XOR ^ nums[i];
        }

        /* Variable to get the rightmost
        set bit in overall XOR */
        int rightmost = (XOR & (XOR - 1)) ^
          XOR;

        /* Variables to stores XOR of
        elements in bucket 1 and 2 */
        int XOR1 = 0, XOR2 = 0;

        // Traverse the array
        for(int i=0; i < n; i++) {

            /* Divide the numbers among bucket
              1
             and 2 based on rightmost set bit
              */
            if(nums[i] & rightmost) {
                XOR1 = XOR1 ^ nums[i];
            }
            else {
                XOR2 = XOR2 ^ nums[i];
            }
        }

        // Return the result in sorted order
        if(XOR1 < XOR2) return {XOR1, XOR2};
        return {XOR2, XOR1};
    }
};
```

Q163. Pow(x,n)

EASY

Topic: Bit Manipulation [Concepts & Problems] > Advanced Maths

Practice Link

Example 1:
Input: x = 2.0000, n = 10
Output: 1024.0000
Explanation: The answer is calculated as 2^10, which equals 1024.

Example 2:
Input: x = 2.0000, n = -2
Output: 0.2500
Explanation: The answer is calculated as 2^(-2), which is equal to 1/4 = 0.25.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(log n), where n is the absolute value of the exponent. This is because we reduce the proble

```cpp
#include<bits/stdc++.h>
using namespace std;

// Class definition for the solution
class Solution {
private:
    // Function to calculate power
    // of 'x' raised to 'n'
    double power(double x, long n) {
        // Base case: anything raised to 0 is
          1
        if (n == 0) return 1.0;

        // Base case: anything raised to 1 is
          itself
        if (n == 1) return x;

        // If 'n' is even
        if (n % 2 == 0) {
            // Recursive call: x * x, n / 2
            return power(x * x, n / 2);
        }
        // If 'n' is odd
        // Recursive call: x * power(x, n-1)
        return x * power(x, n - 1);
    }

public:
    // Function to calculate x raised to n
    double myPow(double x, int n) {
        // Store the value of n in a separate
          variable
        int num = n;

        // If n is negative
        if (num < 0) {
            // Calculate the power of -n and
              take reciprocal
            return (1.0 / power(x, -1 * num));
        }
        // If n is non-negative
        return power(x, num);
    }
};
```

Q164. Implement Stack using Arrays

EASY

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Learning

Practice Link

Example 1:
Input:
["ArrayStack", "push", "push", "top", "pop", "isEmpty"]
[[], [5], [10], [], [], []]
Output: [null, null, null, 10, 10, false]
Explanation: ArrayStack stack = new ArrayStack();
- stack.push(5);
- stack.push(10);
- stack.top(); // returns 10
- stack.pop(); // returns 10
- stack.isEmpty(); // returns false

Example 2:
Input:
["ArrayStack", "isEmpty", "push", "pop", "isEmpty"]
[[], [], [1], [], []]
Output: [null, true, null, 1, true]
Explanation:... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1) for all operations (push, pop, top, isEmpty).

```cpp
#include <bits/stdc++.h>
using namespace std;
class ArrayStack {
private:
    // Array to hold elements
    int* stackArray;
    // Maximum capacity
    int capacity;
     // Index of top element
    int topIndex;
public:
    // Constructor
    ArrayStack(int size = 1000) {
        capacity = size;
        stackArray = new int[capacity];
        // Initialize stack as empty
        topIndex = -1;
    }
    // Destructor
    ~ArrayStack() {
        delete[] stackArray;
    }
    // Pushes element x
    void push(int x) {
        if (topIndex >= capacity - 1) {
            cout << "Stack overflow" << endl;
            return;
        }
        stackArray[++topIndex] = x;
    }
    // Removes and returns top element
    int pop() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            // Return invalid value
            return -1;
        }
        return stackArray[topIndex--];
    }
    // Returns top element
    int top() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return -1;
        }
        return stackArray[topIndex];
    }
    bool isEmpty() {
        return topIndex == -1;
    }
};
// Main Function
```

Q165. Implement Queue using Arrays

EASY

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Learning

Practice Link

Example 1:
Input:
["ArrayQueue", "push", "push", "peek", "pop", "isEmpty"]
[[], [5], [10], [], [], []]
Output: [null, null, null, 5, 5, false]
Explanation:
ArrayQueue queue = new ArrayQueue();
- queue.push(5);
- queue.push(10);
- queue.peek();... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1) for all operations (push, pop, peek, isEmpty) since they involve simple index manipulati

```cpp
#include <bits/stdc++.h>
using namespace std;
class ArrayQueue {
    int* arr;
    int start, end;
    int currSize, maxSize;
public:
    ArrayQueue() {
        arr = new int[10];
        start = -1;
        end = -1;
        currSize = 0;
        maxSize = 10;
    }
    void push(int x) {
        if (currSize == maxSize) {
            cout << "Queue is
              full\nExiting..." << endl;
            exit(1);
        }
        if (end == -1) {
            start = 0;
            end = 0;
        }
        else {
            end = (end + 1) % maxSize;
        }
        arr[end] = x;
        currSize++;
    }
    int pop() {
        if (start == -1) {
            cout << "Queue Empty\nExiting..."
              << endl;
            exit(1);
        }
        int popped = arr[start];
        if (currSize == 1) {
            start = -1;
            end = -1;
        }
        else {
            start = (start + 1) % maxSize;
        }
        currSize--;
        return popped;
    }
    int peek() {
        if (start == -1) {
            cout << "Queue is Empty" << endl;
            exit(1);
        }
        return arr[start];
    }
    bool isEmpty() {
        return (currSize == 0);
    }
};
```

Q166. Implement Stack using Queue

EASY

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Learning

Practice Link

Example 1:
Input:
["QueueStack", "push", "push", "pop", "top", "isEmpty"]
[[], [4], [8], [], [], []]
Output: [null, null, null, 8, 4, false]
Explanation: QueueStack stack = new QueueStack();
- stack.push(4);
- stack.push(8);
- stack.pop(); // returns 8
- stack.top(); // returns 4
- stack.isEmpty(); // returns false

Example 2:
Input:
["QueueStack", "isEmpty"]
[[]]
Output: [null, true]
Explanation: QueueStack stack = new QueueStack();
- stack.isEmpty(); // ret... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: Push operation: O(n) (where n is the number of elements in the queue at that time) because ev

```cpp
#include <bits/stdc++.h>
using namespace std;

// Stack implementation using Queue
class QueueStack {
    // Queue
    queue<int> q;

public:
    // Method to push element in the stack
    void push(int x) {
        // Get size
        int s = q.size();
        // Add element
        q.push(x);

        // Move elements before new element to
          back
        for (int i = 0; i < s; i++) {
            q.push(q.front());
            q.pop();
        }
    }

    // Method to pop element from stack
    int pop() {
        // Get front element
        int n = q.front();
        // Remove front element
        q.pop();
        // Return removed element
        return n;
    }

    // Method to return the top of stack
    int top() {
        // Return front element
        return q.front();
    }

    // Method to check if the stack is empty
    bool isEmpty() {
        return q.empty();
    }
};
```

Q167. Implement Queue using Stack

EASY

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Learning

Practice Link

Example 1:
Input:
["StackQueue", "push", "push", "pop", "peek", "isEmpty"]
[[], [4], [8], [], [], []]
Output: [null, null, null, 4, 8, false]
Explanation:
StackQueue queue = new StackQueue();
- queue.push(4);
- queue.push(8);
- queue.pop(); // returns 4
- queue.peek(); // returns 8
- queue.isEmpty(); // returns false

Example 2:
Input:
["StackQueue", "isEmpty"]
[[]]
Output: [null, true]
Explanation:
StackQueue queue = new StackQueue();
- queue.isEmpty(); // r... (cont.)

[OPTIMAL SOLUTION] General — Time Complexity: O(1) for push operation, O(n) for pop and peek operations in the worst case when elements nee

```cpp
#include <bits/stdc++.h>
using namespace std;
class StackQueue {
  public:
    stack<int> input, output;
    // Initialize your data structure here
    StackQueue() {}
    // Push element x to the back of queue
    void push(int x) {
        input.push(x);
    }
    // Removes the element from in front of
      queue and returns that element
    int pop() {
        // Shift input to output if output is
          empty
        if (output.empty()) {
            while (!input.empty()) {
                output.push(input.top());
                input.pop();
            }
        }
        // If queue is still empty, return -1
          (or throw an error if preferred)
        if (output.empty()) {
            cout << "Queue is empty, cannot
              pop." << endl;
            return -1;
        }
        int x = output.top();
        output.pop();
        return x;
    }
    // Get the front element
    int peek() {
        // Shift input to output if output is
          empty
        if (output.empty()) {
            while (!input.empty()) {
                output.push(input.top());
                input.pop();
            }
        }
        // If queue is still empty, return -1
          (or throw an error if preferred)
        if (output.empty()) {
            cout << "Queue is empty, cannot
              peek." << endl;
            return -1;
        }
        return output.top();
    }
    // Returns true if the queue is empty,
      false otherwise
    bool isEmpty() {
        return input.empty() &&
          output.empty();
    }
};
```

Q168. Implement stack using Linkedlist

EASY

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Learning

Practice Link

Example 1:
Input:
["LinkedListStack", "push", "push", "pop", "top", "isEmpty"]
[[], [3], [7], [], [], []]
Output: [null, null, null, 7, 3, false]
Explanation: LinkedListStack stack = new LinkedListStack();
- stack.push(3);
- stack.push(7);
- stack.pop(); // returns 7
- stack.top(); // returns 3
- stack.isEmpty(); // returns false

Example 2:
Input:
["LinkedListStack", "isEmpty"]
[[]]
Output: [null, true]
Explanation: LinkedListStack stack = new LinkedListStack... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1) for push, pop, size, isEmpty, peek operations. Space Complexity: O(N) because the stack

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int val;
    Node *next;
    Node(int d) {
        val = d;
        next = NULL;
    }
};
class LinkedListStack {
private:
    Node *head;
    int size;
public:
    LinkedListStack() {
        head = NULL;
        size = 0;
    }
    void push(int x) {
        Node *element = new Node(x);
        element->next = head;
        head = element;
        size++;
    }
    int pop() {
        if (head == NULL) {
            return -1;
        }
        int value = head->val;
        Node *temp = head;
        head = head->next;
        delete temp;
        size--;
        return value;
    }
    int top() {
        if (head == NULL) {
            return -1;
        }
        return head->val;
    }
    bool isEmpty() {
        return (size == 0);
    }
};
```

Q169. Implement queue using Linkedlist

EASY

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Learning

Practice Link

Example 1:
Input:
["LinkedListQueue", "push", "push", "peek", "pop", "isEmpty"]
[[], [3], [7], [], [], []]
Output: [null, null, null, 3, 3, false]
Explanation:
LinkedListQueue queue = new LinkedListQueue();
- queue.push(3);
- queue.push(7);
- queue.peek(); // returns 3
- queue.pop(); // returns 3
- queue.isEmpty(); // returns false

Example 2:
Input: ["LinkedListQueue", "push", "pop", "isEmpty"]
[[], [2], [], []]
Output: [null, null, 2, true]
Explanation:
LinkedL... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(1) for all operations (push, pop, peek, isEmpty) since they involve constant time operation

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int val;
    Node *next;
    Node(int d) {
        val = d;
        next = NULL;
    }
};
class LinkedListQueue {
private:
    Node *start;
    Node *end;
    int size;
public:
    LinkedListQueue() {
        start = end = NULL;
        size = 0;
    }
    void push(int x) {
        Node *element = new Node(x);
        if(start == NULL) {
            start = end = element;
        }
        else {
            end->next = element;
            end = element;
        }
        size++;
    }
    int pop() {
        if (start == NULL) {
            return -1;
        }
        int value = start->val;
        Node *temp = start;
        start = start->next;
        delete temp;
        size--;
        return value;
    }
    int peek() {
        if (start == NULL) {
            return -1;
        }
        return start->val;
    }
    bool isEmpty() {
        return (size == 0);
    }
};
```

Q170. Balanced Paranthesis

EASY

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Learning

Practice Link

Example 1:
Input: str = “( )[{ } ( )]”
Output: True
Explanation: As every open bracket has its corresponding close bracket. Match parentheses are in correct order hence they are balanced.

Example 2:
Input: str = “[( )”
Output: False
Explanation: As ‘[‘ does not have ‘]’ hence it is not valid and will return false.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n). Single for loop used Space Complexity: O(N). Stack space

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class containing the isValid method
class Solution {
public:
    // Function to check if the input string has valid parentheses
    bool isValid(string s) {
        stack<char> st;  // Stack to store opening brackets

        for (auto it : s) {
            if (it == '(' || it == '{' || it == '[')
                st.push(it);  // Push opening brackets to stack
            else {
                if (st.empty()) return false;  // No matching opening bracket
                char ch = st.top();
                st.pop();

                // Check for matching pair
                if ((it == ')' && ch == '(') ||
                    (it == ']' && ch == '[') ||
                    (it == '}' && ch == '{'))
                    continue;
                else
                    return false;
            }
        }
        return st.empty();  // True if all brackets matched
    }
};
```

Q171. Implement Min Stack

HARD

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Learning

Practice Link

Example 1:
Input: ["MinStack", "push", "push", "push", "getMin", "pop", "top", "getMin"]
[[], [-2], [0], [-3], [ ], [ ], [ ], [ ]]
Output: [null, null, null, null, -3, null, 0, -2]
Explanation:
MinStack minStack = new MinStack();
- minStack.push(-2);
- minStack.push(0);
- minStack.push(-3);
- minStack.getMin(); // returns -3
- minStack.pop();
- minStack.top(); // returns 0
- minStack.getMin(); // returns -2

Example 2:
Input: ["MinStack", "push", "push", "getMin", "... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(1) for all operations (push, pop, top, getMin) as they involve constant time operations on

```cpp
#include <bits/stdc++.h>
using namespace std;
class MinStack {
private:
    stack <int> st;
    int mini;
public:
    MinStack() {
    }
    void push(int value) {
        if(st.empty()) {
            mini = value;
            st.push( value );
            return;
        }
        if(value > mini) {
            st.push(value);
        }
        else {
            st.push(2 * value - mini);
            mini = value;
        }
    }
    void pop() {
        if(st.empty()) return;
        int x = st.top();
        st.pop();
        if(x < mini) {
            mini = 2 * mini - x;
        }
    }
    int top() {
        if(st.empty()) return -1;
        int x = st.top();
        if(mini < x) return x;
        return mini;
    }
    int getMin() {
        return mini;
    }
};
```

Q172. Infix to Postfix Conversion

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Prefix, Infix, PostFix Conversion Problems

Practice Link

Example 1:
Input: a + b * (c^d - e) ^ (f + g * h) - i
Output: abcd^e-fgh*+^*+i-
Explanation:
The infix expression "a + b * (c^d - e) ^ (f + g * h) - i" is converted to postfix form as "abcd^e-fgh*+^*+i-" by applying the rules of infix to postfix conversion.

Example 2:
Input: (p + q) * (m - n)
Output: pq+mn-*
Explanation:
The infix expression "(p + q) * (m - n)" is converted to postfix form as "pq+mn-*".

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the length of the infix expression. Each character in the expression is proc

```cpp
#include<bits/stdc++.h>
using namespace std;
// Function to return precedence of operators
int prec(char c) {
    if (c == '^')  // Exponent operator has
      highest precedence
        return 3;
    else if (c == '/' || c == '*')  //
      Multiplication and division have higher
      precedence than addition
        return 2;
    else if (c == '+' || c == '-')  //
      Addition and subtraction have lowest
      precedence
        return 1;
    else
        return -1;
}
// The main function to convert infix
  expression to postfix expression
void infixToPostfix(string s) {
    stack<char> st; // Stack to hold operators
      and parentheses
    string result;  // String to hold the
      resulting postfix expression
    for (int i = 0; i < s.length(); i++) {
        char c = s[i];
        // If the scanned character is an
          operand, add it to the result string
        if ((c >= 'a' && c <= 'z') || (c >=
          'A' && c <= 'Z') || (c >= '0' && c
          <= '9'))
            result += c;
        // If the scanned character is an '(',
          push it to the stack
        else if (c == '(')
            st.push('(');
        // If the scanned character is a ')',
          pop from stack until an '(' is
          encountered
        else if (c == ')') {
            while (st.top() != '(') {
                result += st.top();
                st.pop();
            }
            st.pop();  // Pop the '(' from the
              stack
        }
        // If an operator is scanned
        else {
            while (!st.empty() && prec(s[i])
              <= prec(st.top())) {
                result += st.top();
                st.pop();
            }
            st.push(c);  // Push the current
              operator to the stack
        }
    }
    // Pop all the remaining elements from the
      stack
    while (!st.empty()) {
        result += st.top();
        st.pop();
    }
    cout << "Postfix expression: " << result
      << endl;  // Output the result
}
```

Q173. Prefix to Infix Conversion

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Prefix, Infix, PostFix Conversion Problems

Practice Link

Example 1:
Input: expression = "+ab
Output: (a+b)

Example 2:
Input: expression = "*+ab-cd
Output: ((a+b)*(c-d))

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n) where n is the length of the prefix expression (only one pass). Space Complexity: O(n) f

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to convert prefix to infix
string prefixToInfix(string prefix) {
    stack<string> s;
    int n = prefix.size();

    // Traverse the prefix expression from right to left
    for (int i = n - 1; i >= 0; i--) {
        char c = prefix[i];

        // If the character is an operand, push it to the stack
        if (isalnum(c)) {
            s.push(string(1, c));
        } else {
            // Pop two operands from the stack
            string op1 = s.top(); s.pop();
            string op2 = s.top(); s.pop();

            // Form the new infix expression and push back to stack
            s.push("(" + op1 + c + op2 + ")");
        }
    }

    // The final element in the stack is the result
    return s.top();
}
```

Q174. Prefix to Postfix Conversion

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Prefix, Infix, PostFix Conversion Problems

Practice Link

Example 1:
Input: expression = "+ab
Output: ab+

Example 2:
Input: expression = "*+ab-cd
Output: ab+cd-*

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), single pass through the expression. Space Complexity: O(n), stack space for storing int

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to convert prefix to postfix
string prefixToPostfix(string prefix) {
    stack<string> s;
    int n = prefix.size();

    // Traverse the prefix expression from right to left
    for (int i = n - 1; i >= 0; i--) {
        char c = prefix[i];

        // If the character is an operand, push it to the stack
        if (isalnum(c)) {
            s.push(string(1, c));
        } else {
            // Pop two operands from the stack
            string op1 = s.top(); s.pop();
            string op2 = s.top(); s.pop();

            // Form the new postfix expression and push back to stack
            s.push(op1 + op2 + c);
        }
    }

    // The final element in the stack is the result
    return s.top();
}
```

Q175. Postfix to Prefix Conversion

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Prefix, Infix, PostFix Conversion Problems

Practice Link

Example 1:
Input: expression = "ab+"
Output: "+ab"
Explanation: Postfix → Prefix

Example 2:
Input: expression = "abc*+d-"
Output: "-+a*bcd"

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), as we traverse the expression only once. Space Complexity: O(n) for the stack to store

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to convert postfix to prefix
string postfixToPrefix(string postfix) {
    stack<string> s;
    int n = postfix.size();

    // Traverse the postfix expression from left to right
    for (int i = 0; i < n; i++) {
        char c = postfix[i];

        // If the character is an operand, push it to the stack
        if (isalnum(c)) {
            s.push(string(1, c));
        } else {
            // Pop two operands from the stack
            string op2 = s.top(); s.pop();
            string op1 = s.top(); s.pop();

            // Form the new prefix expression and push back to stack
            s.push(c + op1 + op2);
        }
    }

    // The final element in the stack is the result
    return s.top();
}
```

Q176. Postfix to Infix Conversion

EASY

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Prefix, Infix, PostFix Conversion Problems

Practice Link

Example 1:
Input: "ab+c*"
Output: "(a+b)*c"

Example 2:
Input: "ab*cd/+"
Output: "(a*b)+(c/d)"

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), a single pass over the postfix expression. Space Complexity: O(n), stack space for sto

```cpp
#include <iosbits/stdc++.htream>
using namespace std;

// Function to convert postfix to infix
string postfixToInfix(string postfix) {
    stack<string> s;
    int n = postfix.size();

    // Traverse the postfix expression from left to right
    for (int i = 0; i < n; i++) {
        char c = postfix[i];

        // If the character is an operand, push it to the stack
        if (isalnum(c)) {
            s.push(string(1, c));
        } else {
            // Pop two operands from the stack
            string op2 = s.top(); s.pop();
            string op1 = s.top(); s.pop();

            // Form the new infix expression and push back to stack
            s.push("(" + op1 + c + op2 + ")");
        }
    }

    // The final element in the stack is the result
    return s.top();
}
```

Q177. Infix to Prefix Conversion

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Prefix, Infix, PostFix Conversion Problems

Practice Link

Example 1:
Input: x + y * z / w + u
Output: ++x/*yzwu
Explanation:
The infix expression "x + y * z / w + u" is converted to prefix form as "++x/*yzwu" by applying the rules of infix to prefix conversion.

Example 2:
Input: a + b
Output: +ab
Expla... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the length of the infix expression. Each character is processed once.

```cpp
#include <bits/stdc++.h>
using namespace std;
bool isOperator(char c) {
    return (!isalpha(c) && !isdigit(c));
}
int getPriority(char C) {
    if (C == '-' || C == '+')
        return 1;
    else if (C == '*' || C == '/')
        return 2;
    else if (C == '^')
        return 3;
    return 0;
}
string infixToPostfix(string infix) {
    infix = '(' + infix + ')';
    int l = infix.size();
    stack<char> char_stack;
    string output;
    for (int i = 0; i < l; i++) {
        if (isalpha(infix[i]) ||
          isdigit(infix[i]))
            output += infix[i];
        else if (infix[i] == '(')
            char_stack.push('(');
        else if (infix[i] == ')') {
            while (char_stack.top() != '(') {
                output += char_stack.top();
                char_stack.pop();
            }
            char_stack.pop();
        }
        else {
            if (isOperator(char_stack.top()))
              {
                if (infix[i] == '^') {
                    while
                      (getPriority(infix[i])
                      <= getPriority(char_stac
                      k.top())) {
                        output +=
                          char_stack.top();
                        char_stack.pop();
                    }
                } else {
                    while
                      (getPriority(infix[i]) <
                      getPriority(char_stack.t
                      op())) {
                        output +=
                          char_stack.top();
                        char_stack.pop();
                    }
                }
                char_stack.push(infix[i]);
            }
        }
    }
    while (!char_stack.empty()) {
        output += char_stack.top();
        char_stack.pop();
    }
    return output;
}
string infixToPrefix(string infix) {
    int l = infix.size();
    reverse(infix.begin(), infix.end());
    for (int i = 0; i < l; i++) {
        if (infix[i] == '(') {
            infix[i] = ')';
            i++;
        } else if (infix[i] == ')') {
            infix[i] = '(';
            i++;
        }
    }
    string prefix = infixToPostfix(infix);
    reverse(prefix.begin(), prefix.end());
    return prefix;
}
```

Q178. Next Greater Element

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Input: arr = [1, 3, 2, 4]
Output: [3, 4, 4, -1]
Explanation: In the array, the next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4 is -1, since it does not exist.

Input : arr = [6, 8, 0, 1, 3]
Output: [8, -1, 1, 3, -1]
Explanation : In the array, the next larger element to 6 is 8, for 8 there is no larger elements hence it is -1, for 0 it is 1 , for 1 it is 3 and then for 3 there is no larger element on the right and hence -1.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), we traverse the entire array once and find next greater element in linear time.

```cpp
#include <bits/stdc++.h>
using namespace std;

// Solution class to find next greater
  elements
class Solution {
public:
    // Function to find next greater elements
    vector<int> nextGreater(vector<int>& nums)
      {
        // Stack to store elements
        stack<int> st;

        // Result array of same size
        int n = nums.size();
        vector<int> res(n);

        // Traverse from right to left
        for (int i = n - 1; i >= 0; i--) {

            // Pop all smaller or equal
              elements
            while (!st.empty() && st.top() <=
              nums[i]) {
                st.pop();
            }

            // If stack is empty, no greater
              element
            if (st.empty()) res[i] = -1;

            // Else top of stack is the answer
            else res[i] = st.top();

            // Push current element
            st.push(nums[i]);
        }

        // Return the result
        return res;
    }
};

// Main function
```

Q179. Next Greater Element - 2

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Example 1:
Input: arr = [3, 10, 4, 2, 1, 2, 6, 1, 7, 2, 9]
Output: [10, -1, 6, 6, 2, 6, 7, 7, 9, 9, 10]
Explanation: For the first element in arr i.e, 3, the greater element which comes next to it while traversing and is closest to it is 10. Hence,10 is present on index 0 in the resultant array. Now for the second element i.e, 10, there is no greater number and hence -1 is it's next greater element (NGE). Similarly, we got the NGEs for all other elements present in arr.

Example 2:
Input: arr =... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), since traversing on the array takes O(N) time and traversing the stack will take overal

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    vector<int> nextGreaterElements(vector<int> arr) {
        int n = arr.size(); //size of array
        // To store the next greater elements
        vector<int> ans(n);
        // Stack to get elements in LIF fashion
        stack<int> st;
        // Start traversing from the back
        for(int i = 2*n-1; i >= 0; i--) {
            // Get the actual index
            int ind = i % n;
            // Get the current element
            int currEle = arr[ind];
            while(!st.empty() && st.top() <= currEle) {
                st.pop();
            }
            // Store the answer for the second half
            if(i < n) {
                if(st.empty())
                    ans[i] = -1;
                // Else store the answer
                else
                    ans[i] = st.top();
            }
            st.push(currEle);
        }
        // Return the result
        return ans;
    }
};
```

Q180. Next Smaller Element

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Example 1:
Input: arr = [4, 8, 5, 2, 25]
Output: [2, 5, 2, -1, -1]
Explanation:
- For 4, the next smaller element is 2.
- For 8, the next smaller element is 5.
- For 5, the next smaller element is 2.
- For 2, there is no smaller element to its right → -1.
- For 25, no smaller element exists → -1.

Example 2:
Input: arr = [10, 9, 8, 7]
Output: [9, 8, 7, -1]
Explanation:
Each element's next right neighbor is smaller.
Each element's next right neighbor is smaller.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), since each element is pushed and popped at most once. Space Com

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    vector<int> nextSmallerElement(vector<int>& arr) {
        int n = arr.size();

        // Stack to store potential next smaller elements
        stack<int> st;

        // Answer array initialized with -1
        vector<int> ans(n, -1);

        // Traverse the array from right to left
        for (int i = n - 1; i >= 0; i--) {

            // Pop elements from stack while they are >= current element
            while (!st.empty() && st.top() >= arr[i]) {
                st.pop();
            }

            // If stack is not empty, top is the next smaller element
            if (!st.empty()) {
                ans[i] = st.top();
            }

            // Push current element to stack
            st.push(arr[i]);
        }

        // Return result
        return ans;
    }
};
```

Q181. Number of Greater Elements to the Right

EASY

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Example 1:
Input: arr = [1, 3, 2, 4]
Output: [3, 4, 4, -1]
Explanation:
The next greater element for 1 is 3, for 3 is 4, for 2 is 4, and for 4 is -1 (no greater element on the right).

Example 2:
Input: arr = [6, 8, 0, 1, 3]
Output: [8, -1, 1, 3, -1]
Explanation:
The next greater element for 6 is 8, for 8 is -1, for 0 is 1, for 1 is 3, and for 3 is -1.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N) (where N is the size of the array). Traversing on the hypothetical array takes O(2N) tim

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:

    /* Function to find the next greater
    element for each element in the array */
    vector<int> nextLargerElement(vector<int>
      arr) {

        int n = arr.size(); //size of array

        // To store the next greater elements
        vector<int> ans(n);

        // Stack to get elements in LIFO
          fashion
        stack<int> st;

        // Start traversing from the back
        for(int i=n-1; i >= 0; i--) {

            // Get the current element
            int currEle = arr[i];

            /* Pop the elements in the stack
              until
              the stack is not empty and the top
              element is not the greater element
                */
              while(!st.empty() && st.top() <=
                currEle) {
                  st.pop();
              }

              /* If the greater element is not
              found, stack will be empty */
              if(st.empty())
                  ans[i] = -1;

              // Else store the answer
              else
                  ans[i] = st.top();

              /* Push the current element in the
                stack
              maintaining the decreasing order
                */
              st.push(currEle);
        }

        // Return the result
        return ans;
    }
};
```

Q182. Trapping Rainwater

HARD

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Input : height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output : 6
Explanation : Water is trapped in the dips between bars. The total trapped water units add up to 6 (1+1+2+1+1).
Input : height = [4,2,0,3,2,5]
Output : 9
Explanation : The elevation map traps 9 units of water in total, as water fills the spaces between higher bars on both sides.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n) because the two pointers traverse the array only once, each pointer moving inward and co

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to calculate trapped rainwater
      using the optimal two-pointer approach
    int trap(vector<int>& height) {
        int n = height.size();
        // Initialize two pointers at both
          ends of the array
        int left = 0;
        int right = n - 1;
        // Variables to track the maximum
          height to the left and right
        int maxLeft = 0;
        int maxRight = 0;
        // Variable to store total trapped
          water
        int totalWater = 0;
        // Iterate until left pointer meets
          right pointer
        while (left <= right) {
            // If left bar is smaller or equal
              to right bar
            if (height[left] <= height[right])
              {
                // If current left bar is
                  higher than maxLeft, update
                  maxLeft
                if (height[left] >= maxLeft) {
                    maxLeft = height[left];
                } else {
                    // Water trapped on left
                      is difference between
                      maxLeft and current
                      height
                    totalWater += maxLeft -
                      height[left];
                }
                left++; // Move left pointer
                  to the right
            } else {
                // If current right bar is
                  higher than maxRight, update
                  maxRight
                if (height[right] >= maxRight)
                  {
                    maxRight = height[right];
                } else {
                    // Water trapped on right
                      is difference between
                      maxRight and current
                      height
                    totalWater += maxRight -
                      height[right];
                }
                right--; // Move right pointer
                  to the left
            }
        }
        // Return total trapped water
        return totalWater;
    }
};
// Driver code
```

Q183. Sum of Subarray Minimums

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Example 1:
Input: arr = [3, 1, 2, 5]
Output: 18
Explanation: The minimum of subarrays: [3], [1], [2], [5], [3, 1], [1, 2], [2, 5], [3, 1, 2], [1, 2, 5], [3, 1, 2, 5] are 3, 1, 2, 5, 1, 1, 2, 1, 1, 1 respectively and their sum is 18.

Example 2:
Input: arr = [2, 3, 1]
Output: 10
Explanation: The minimum of subarrays: [2], [3], [1], [2,3], [3,1], [2,3,1] are 2, 3, 1, 2, 1, 1 respectively and their sum is 10.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), since finding the indices of next smaller elements and previous smaller elements take O

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    vector<int> findNSE(vector<int> &arr) {
        int n = arr.size();
        vector<int> ans(n);
        stack<int> st;
        for(int i = n - 1; i >= 0; i--) {
            int currEle = arr[i];
            while(!st.empty() && arr[st.top()]
              >= arr[i]){
                st.pop();
            }
            ans[i] = !st.empty() ? st.top() :
              n;
            st.push(i);
        }
        return ans;
    }
    vector<int> findPSEE(vector<int> &arr) {
        int n = arr.size();
        vector<int> ans(n);
        stack<int> st;
        for(int i=0; i < n; i++) {
            int currEle = arr[i];
            while(!st.empty() && arr[st.top()]
              > arr[i]){
                st.pop();
            }
            ans[i] = !st.empty() ? st.top() :
              -1;
            st.push(i);
        }
        return ans;
    }
public:
    int sumSubarrayMins(vector<int> &arr) {
        vector<int> nse =
            findNSE(arr);
        vector<int> psee =
            findPSEE(arr);
        int n = arr.size();
        int mod = 1e9 + 7;
        int sum = 0;
        for(int i=0; i < n; i++) {
            int left = i - psee[i];
            int right = nse[i] - i;
            long long freq = left*right*1LL;
            int val = (freq*arr[i]*1LL) % mod;
            sum = (sum + val) % mod;
        }
        return sum;
    }
};
```

Q184. Asteroid Collision

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Example 1:
Input: asteroids = [2, -2]
Output: []
Explanation: The asteroid with size 2 and the one with size -2 collide, exploding each other.

Example 2:
Input: asteroids = [10, 20, -10]
Output: [10, 20]
Explanation: The asteroid with size 20 and the one with size -10 collide, resulting in the remaining asteroid with size 20. The asteroids with sizes 10 and 20 never collide.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), since traversing all the asteroids takes O(N) time. Space Complexit

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    vector<int> asteroidCollision(vector<int> &asteroids){
        // Size of the array
        int n = asteroids.size();
        // List implementation of stack
        vector<int> st;
        // Traverse all the asteroids
        for(int i=0; i < n; i++) {
            if(asteroids[i] > 0) {
                st.push_back(asteroids[i]);
            }
            else {
                while(!st.empty() && st.back() > 0 &&
                      st.back() < abs(asteroids[i])) {
                    // Destroy the asteroid
                    st.pop_back();
                }
                if(!st.empty() &&
                    st.back() == abs(asteroids[i])) {
                    // Destroy both the asteroids
                    st.pop_back();
                }
                else if(st.empty() ||
                        st.back() < 0){
                    // Storing the array in final state
                    st.push_back(asteroids[i]);
                }
            }
        }
        // Return the final state of asteroids
        return st;
    }
};
```

Q185. Sum of Subarray Ranges

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Example 1:
Input: nums = [1, 2, 3]
Output: 4
Explanation: The 6 subarrays of nums are the following:
[1], range = largest - smallest = 1 - 1 = 0
[2], range = 2 - 2 = 0
[3], range = 3 - 3 = 0
[1,2], range = 2 - 1 = 1
[2,3], range = 3 - 2 = 1
[1,2,3], range = 3... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), since calculating the sum of subarray maximums takes O(N) time and calculating the sum

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    vector<int> findNSE(vector<int>
      &arr) {
        int n = arr.size();
        vector<int> ans(n);
        stack<int> st;
        for(int i = n - 1; i >= 0;
          i--) {
            int currEle = arr[i];
            while(!st.empty() &&
              arr[st.top()] >=
              currEle){
                st.pop();
            }
            ans[i] = !st.empty() ?
              st.top() : n;
            st.push(i);
        }
        return ans;
    }
    vector<int> findNGE(vector<int>
      &arr) {
        int n = arr.size();
        vector<int> ans(n);
        stack<int> st;
        for(int i = n - 1; i >= 0;
          i--) {
            int currEle = arr[i];
            while(!st.empty() &&
              arr[st.top()] <=
              currEle){
                st.pop();
            }
            ans[i] = !st.empty() ?
              st.top() : n;
            st.push(i);
        }
        return ans;
    }
    vector<int> findPSEE(vector<int>
      &arr) {
        int n = arr.size();
        vector<int> ans(n);
        stack<int> st;
        for(int i=0; i < n; i++) {
            int currEle = arr[i];
            while(!st.empty() &&
              arr[st.top()] >
              currEle){
                st.pop();
            }
            ans[i] = !st.empty() ?
              st.top() : -1;
            st.push(i);
        }
        return ans;
    }
    vector<int> findPGEE(vector<int>
      &arr) {
        int n = arr.size();
        vector<int> ans(n);
        stack<int> st;
        for(int i=0; i < n; i++) {
            int currEle = arr[i];
            while(!st.empty() &&
              arr[st.top()] <
              currEle){
                st.pop();
            }
            ans[i] = !st.empty() ?
              st.top() : -1;
            st.push(i);
        }
        return ans;
    }
    long long
      sumSubarrayMins(vector<int>
      &arr) {
        vector<int> nse =
          findNSE(arr);
        vector<int> psee =
          findPSEE(arr);
        int n = arr.size();
        long long sum = 0;
        for(int i=0; i < n; i++) {
            int left = i - psee[i];
            int right = nse[i] - i;
            long long freq =
              left*right*1LL;
            long long val =
              (freq*arr[i]*1LL);
            sum += val;
        }
        return sum;
    }
    long long
      sumSubarrayMaxs(vector<int>
      &arr) {
        vector<int> nge =
          findNGE(arr);
        vector<int> pgee =
          findPGEE(arr);
        int n = arr.size();
        long long sum = 0;
        for(int i=0; i < n; i++) {
            int left = i - pgee[i];
            int right = nge[i] - i;
            long long freq =
              left*right*1LL;
            long long val =
              (freq*arr[i]*1LL);
            sum += val;
        }
        return sum;
    }
public:
    long long
      subArrayRanges(vector<int>
      &arr) {
        return (
          sumSubarrayMaxs(arr) -
sumSubarrayMins(arr) );
    }
};
```

Q186. Remove K Digits

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Example 1:
Input: nums = "541892", k = 2
Output: "1892"
Explanation: Removing the two digits 5 and 4 yields the smallest number, 1892.

Example 2:
Input: nums = "1002991", k = 3
Output: "21"
Explanation: Remove the three digits 1(leading one), 9, and 9 to form the new number 21(Note that the output must not contain leading zeroes) which is the smallest.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), since traversing the given string takes O(N) time, each element is pushed onto and popp

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    string removeKdigits(string nums, int k) {
        stack <char> st; // Stack
        // Traverse on the given string
        for(int i=0; i < nums.size(); i++) {
            // Current digit
            char digit = nums[i];
            while(!st.empty() && k > 0
                  && st.top() > digit) {
                st.pop(); // Pop the last
                  digit
                k--; // Decrement K by 1
            }
            // Push the current digit
            st.push(digit);
        }
        // If more digits can be removed
        while(!st.empty() && k > 0) {
            st.pop(); // Pop the last added
              digits
            k--; // Decrement K by 1
        }
        // Handling edge case
        if(st.empty()) return "0";
        // To store the result
        string res = "";
        // Adding digits in stack to result
        while(!st.empty()) {
            res.push_back(st.top());
            st.pop();
        }
        // Trimming the zeroes at the back
        while(res.size() > 0 &&
              res.back() == '0') {
            res.pop_back();
        }
        // Reverse to get the actual number
        reverse(res.begin(), res.end());
        // Edge case
        if(res.empty()) return "0";
        // Return the stored result
        return res;
    }
};
```

Q187. Largest rectangle in a histogram

HARD

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Example:

Input: N =6, heights[] = {2,1,5,6,2,3}
Output: 10

[OPTIMAL SOLUTION] Optimised Approach 2 — Time Complexity: O(N) + O(N). For loop used along with a while loop Space Complexity:

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
  public:
    int largestRectangleArea(vector<int> &histo) {
        stack<int> st; // Stack to store indices of the histogram bars
        int maxA = 0;  // Variable to keep track of the maximum area
        int n = histo.size();

        // Loop through each bar including an imaginary bar at the end
        for (int i = 0; i <= n; i++) {
            // While current bar is smaller than the top of the stack or we
              reached the end
            while (!st.empty() && (i == n || histo[st.top()] >= histo[i])) {
                int height = histo[st.top()]; // Get the height of the bar at
                  top of the stack
                st.pop(); // Remove that bar

                int width;
                if (st.empty()) {
                    width = i; // All bars before were higher
                } else {
                    width = i - st.top() - 1; // Width between current index
                      and index at top of stack
                }

                // Calculate area and update maximum area
                maxA = max(maxA, width * height);
            }
            // Push current index into stack
            st.push(i);
        }
        return maxA;
    }
};
```

Q188. Maximum Rectangles

HARD

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Monotonic Stack/Queue Problems [VVV. Imp]

Practice Link

Input: matrix = [[1,0,1,0,0],[1,0,1,1,1],[1,1,1,1,1],[1,0,0,1,0]]

Output: 6
Explanation: The highlighted part depicts the rectangle with the largest area i.e. 6

Input: matrix = [[1]]
Output: 1
Explanation: In this case, there is only one rectangle with area 1.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N*(M+M)), where N = total no. of rows and M = total no. of columns. Space Complexity: O

```cpp
#include <bits/stdc++.h>
using namespace std;
// Class containing the solution functions
class Solution {
public:
    // Function to calculate largest rectangle
      area in a histogram
    int largestRectangleArea(vector<int>&
      heights) {
        // Stack to store indices of histogram
          bars
        stack<int> st;
        // Variable to store the max area
        int maxArea = 0;
        // Add a sentinel zero height at the
          end
        heights.push_back(0);
        // Iterate through all bars
        for (int i = 0; i < heights.size();
          i++) {
            // While current bar is smaller
              than the bar at stack top
            while (!st.empty() && heights[i] <
              heights[st.top()]) {
                // Pop the top height
                int height =
                  heights[st.top()];
                st.pop();
                // Calculate width
                int width = st.empty() ? i : i
                  - st.top() - 1;
                // Update max area
                maxArea = max(maxArea, height
                  * width);
            }
            // Push current index
            st.push(i);
        }
        return maxArea;
    }
    // Function to find maximal rectangle in
      binary matrix
    int maximalRectangle(vector<vector<char>>&
      matrix) {
        // Return 0 if matrix is empty
        if (matrix.empty()) return 0;
        // Get number of columns
        int m = matrix[0].size();
        // Height array to build histogram
          row-wise
        vector<int> height(m, 0);
        // Variable to track max rectangle
          area
        int maxArea = 0;
        // Traverse each row of the matrix
        for (auto& row : matrix) {
            // Update histogram based on
              current row
            for (int i = 0; i < m; i++) {
                if (row[i] == '1')
                  height[i]++;
                else height[i] = 0;
            }
            // Compute area for this histogram
            maxArea = max(maxArea,
              largestRectangleArea(height));
        }
        return maxArea;
    }
};
// Main driver function
```

Q189. Sliding Window Maximum

HARD

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Implementation Problems

Practice Link

Input: arr = [4,0,-1,3,5,3,6,8], k = 3
Output: [4,3,5,5,6,8]
Explanation:

Window position Max
--------------------- -----
[4 0 -1] 3 5 3 6 8 4
4 [0 -1 3] 5 3 6 8 3
4 0 [-1 3 5] 3 6 8 5
4 0 -1 [3 5 3] 6 8 5
4 0 -1 3 [5 3 6] 8 6
4 0 -1 3 5 [3 6 8] 8

For each window of size k=3, we find the maximum element in the window and add it to our output array.
Input: arr= [20,25],... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n) Each element is pushed and popped from the deque at most once, so overall traversal is l

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to return the max of each
      sliding window of size k
    vector<int> maxSlidingWindow(vector<int>&
      nums, int k) {
        // Deque to store indices of useful
          elements in the current window
        deque<int> dq;

        // Result vector to store the maximums
        vector<int> result;

        // Loop through each element in the
          array
        for (int i = 0; i < nums.size(); i++)
          {
            // Remove elements from the front
              if they are out of this window's
              range
            if (!dq.empty() && dq.front() <= i
              - k) {
                dq.pop_front();
            }

            // Remove all elements from the
              back that are smaller than
              current element
            while (!dq.empty() &&
              nums[dq.back()] < nums[i]) {
                dq.pop_back();
            }

            // Add the current index to the
              deque
            dq.push_back(i);

            // Once the first window is
              completed, add front element to
              result
            if (i >= k - 1) {
result.push_back(nums[dq.front()]);
            }
        }

        // Return the final result
        return result;
    }
};

// Driver code
```

Q190. Stock span problem

HARD

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Implementation Problems

Practice Link

Example 1:
Input: n = 7, arr = [120, 100, 60, 80, 90, 110, 115]
Output: 1 1 1 2 3 5 6
Explanation:
Traversing the given input span:
120 is greater than or equal to 120 and there are no more elements behind it so the span is 1,
100 is greater than or equal to 100 and smaller than 120 so the span is 1,
60 is greater than or equal to 60 and smaller than 100 so the span is 1,
80 is greater than or equal to 60, 80 and smaller than 100 so the span is 2,
90 is greater than or equal to 60, 80, 90 and sm... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), since finding the indices of previous greater elements takes O(N) time and we traverse

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    vector<int> findPGE(vector<int> arr) {
        int n = arr.size(); //size of array
        // To store the previous greater
          elements
        vector<int> ans(n);
        // Stack to get elements in LIFO
          fashion
        stack<int> st;
        // Start traversing from the front
        for(int i=0; i < n; i++) {
            // Get the current element
            int currEle = arr[i];
            while(!st.empty() && arr[st.top()]
              <= currEle) {
                st.pop();
            }
            if(st.empty())
                ans[i] = -1;
            // Else store the answer
            else
                ans[i] = st.top();
            // Push the current index in the
              stack
            st.push(i);
        }
        // Return the result
        return ans;
    }
public:
    // Function to find the span of stock
      prices for each day
    vector <int> stockSpan(vector<int> arr,
      int n) {
        // Get the indices of previous greater
          elements
        vector<int> PGE = findPGE(arr);
        // To store the answer
        vector<int> ans(n);
        // Compute the result
        for(int i=0; i < n; i++) {
            ans[i] = i - PGE[i];
        }
        // Return the result
        return ans;
    }
};
```

Q191. Celebrity Problem

HARD

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Implementation Problems

Practice Link

Example 1:
Input: M = [[0, 1, 1, 0], [0, 0, 0, 0], [1, 1, 0, 0], [0, 1, 1, 0]]
Output: 1
Explanation: Person 1 does not know anyone and is known by persons 0, 2, and 3. Therefore, person 1 is the celebrity.

Example 2:
Input: M = [[0, 1], [1, 0]]
Output: -1
Explanation: Both persons know each other, so there is no celebrity.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), since eliminating persons and checking if the last candidate is a celebrity both take O

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find the index of celebrity
    int celebrity(vector<vector<int>> &M){
        // Size of given matrix
        int n = M.size();
        // Top and Down pointers
        int top = 0, down = n-1;
        // Traverse for all the people
        while(top < down) {
            if(M[top][down] == 1) {
                top = top + 1;
            }
            else if(M[down][top] == 1) {
                down = down - 1;
            }
            else {
                top++;
                down--;
            }
        }
        // Return -1 if no celebrity is found
        if(top > down) return -1;
        for(int i=0; i < n; i++) {
            if(i == top) continue;
            // Check if it is not a celebrity
            if(M[top][i] == 1 || M[i][top] == 0) {
                return -1;
            }
        }
        // Return the index of celebrity
        return top;
    }
};
```

Q192. LRU Cache

MEDIUM

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Implementation Problems

Practice Link

Input: ["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]

Output: [null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation:
LRUCache lRUCache = new LRUCache(2);
lRUCa... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: get() function: O(1), accessing a value in the HashMap is O(1) on ave

```cpp
#include <bits/stdc++.h>
using namespace std;
class LRUCache {
public:
    class Node {
    public:
        int key;
        int val;
        Node* next;
        Node* prev;
        Node(int _key, int _val) {
            key = _key;
            val = _val;
        }
    };
    Node* head = new Node(-1, -1);
    Node* tail = new Node(-1, -1);
    int cap;
    unordered_map<int, Node*> m;
    LRUCache(int capacity) {
        cap = capacity;
        head->next = tail;
        tail->prev = head;
    }
    void addNode(Node* newNode) {
        Node* temp = head->next;
        newNode->next = temp;
        newNode->prev = head;
        head->next = newNode;
        temp->prev = newNode;
    }
    void deleteNode(Node* delNode) {
        Node* delPrev = delNode->prev;
        Node* delNext = delNode->next;
        delPrev->next = delNext;
        delNext->prev = delPrev;
    }
    int get(int key_) {
        if (m.find(key_) != m.end()) {
            Node* resNode = m[key_];
            int res = resNode->val;
            m.erase(key_);
            deleteNode(resNode);
            addNode(resNode);
            m[key_] = head->next;
            return res;
        }
        return -1;
    }
    void put(int key_, int value) {
        if (m.find(key_) != m.end()) {
            Node* existingNode = m[key_];
            m.erase(key_);
            deleteNode(existingNode);
        }
        if (m.size() == cap) {
            m.erase(tail->prev->key);
            deleteNode(tail->prev);
        }
        addNode(new Node(key_, value));
        m[key_] = head->next;
    }
};
```

Q193. LFU Cache

HARD

Topic: Stack and Queues [Learning, Pre-In-Post-fix, Monotonic Stack, Implementation] > Implementation Problems

Practice Link

Example 1:
Input: ["LFUCache", "put", "put", "get", "put", "get", "get", "put", "get", "get", "get"]
Output: [null, null, null, 1, null, -1, 3, null, -1, 3, 4]
Explanation:
LFUCache lfu = new LFUCache(2); Initializing cache with capacity of 2.
lfu.put(1, 1);... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the number of queries on the LFU cache. Each get and put method takes an ave

```
#include <bits/stdc++.h>
using namespace std;
struct Node {
   int key, value, cnt;
   Node *next;
   Node *prev;
   Node(int _key, int _value) {
       key = _key;
       value = _value;
       cnt = 1;
   }
};
struct List {
   int size;
   Node *head;
   Node *tail;
   List() {
       head = new Node(0, 0);
       tail = new Node(0,0);
       head->next = tail;
       tail->prev = head;
       size = 0;
   }
   void addFront(Node *node) {
       Node* temp = head->next;
       node->next = temp;
       node->prev = head;
       head->next = node;
       temp->prev = node;
       size++;
   }
   void removeNode(Node* delnode) {
       Node* prevNode =
         delnode->prev;
       Node* nextNode =
         delnode->next;
       prevNode->next = nextNode;
       nextNode->prev = prevNode;
       size--;
   }
};
class LFUCache {
private:
   map<int, Node*> keyNode;
   map<int, List*> freqListMap;
   int maxSizeCache;
   int minFreq;
   int curSize;
public:
   LFUCache(int capacity) {
       maxSizeCache = capacity;
       minFreq = 0;
       curSize = 0;
   }
   void updateFreqListMap(Node
     *node) {
       keyNode.erase(node->key);
       freqListMap[node->cnt]->remov
         eNode(node);
       if(node->cnt == minFreq &&
freqListMap[node->cnt]->size == 0) {
           minFreq++;
       }
       List* nextHigherFreqList =
         new List();
       if(freqListMap.find(node->cnt
         + 1) !=
          freqListMap.end()) {
          nextHigherFreqList =
            freqListMap[node->cnt +
            1];
       }
       node->cnt += 1;
nextHigherFreqList->addFront(node);
       freqListMap[node->cnt] =
         nextHigherFreqList;
       keyNode[node->key] = node;
   }
   int get(int key) {
       if(keyNode.find(key) !=
         keyNode.end()) {
           Node* node =
             keyNode[key];
           int val = node->value;
           updateFreqListMap(node);
           return val;
       }
       return -1;
   }
   void put(int key, int value) {
       if (maxSizeCache == 0) {
           return;
       }
       if(keyNode.find(key) !=
         keyNode.end()) {
           Node* node =
             keyNode[key];
           node->value = value;
           updateFreqListMap(node);
       }
       else {
           if(curSize ==
             maxSizeCache) {
               List* list =
                 freqListMap[minFreq
                 ];
               keyNode.erase(list->t
                 ail->prev->key);
freqListMap[minFreq]->removeNode(
                   list->tail->prev
               );
               curSize--;
           }
           curSize++;
           minFreq = 1;
           List* listFreq = new
             List();
if(freqListMap.find(minFreq) !=
            freqListMap.end()) {
             listFreq =
               freqListMap[minFreq
               ];
           }
           Node* node = new
             Node(key, value);
           listFreq->addFront(node);
           keyNode[key] = node;
           freqListMap[minFreq] =
             listFreq;
       }
   }
};
```

Q194. Longest Substring Without Repeating Characters

MEDIUM

Topic: Sliding Window & Two Pointer Combined Problems > Medium Problems

Practice Link

Example 1:
Input: S = "abcddabac"
Output: 4
Explanation: The longest substring with distinct characters is "abcd", which has a length of 4.

Example 2:
Input: S = "aaabbbccc"
Output: 2
Explanation: The longest substrings with distinct characters are "ab" and "bc", both having a length of 2.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n), where n is the length of the string. This is because we are using a single pass through

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    int longestNonRepeatingSubstring(string& s) {
        int n = s.size();
        // Assuming all ASCII characters
        int HashLen = 256;
        int hash[HashLen];
        for (int i = 0; i < HashLen; ++i) {
            hash[i] = -1;
        }
        int l = 0, r = 0, maxLen = 0;
        while (r < n) {
            if (hash[s[r]] != -1) {
                l = max(hash[s[r]] + 1, l);
            }
            // Calculate the current substring length
            int len = r - l + 1;
            // Update maximum length found so far
            maxLen = max(len, maxLen);
            hash[s[r]] = r;
            // Move right pointer to next position
            r++;
        }
        // Return the maximum length found
        return maxLen;
    }
};
```

Q195. Max Consecutive Ones III

MEDIUM

Topic: Sliding Window & Two Pointer Combined Problems > Medium Problems

Practice Link

Input : nums = [1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0] , k = 3
Output : 10
Explanation : The maximum number of consecutive 1's are obtained only if we flip the 0's present at position 3, 4, 5 (0 base indexing).
The array after flipping becomes [1, 1, 1, 1, 1, 1, 1,... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N), We traverse the array only once using two pointers (left and r*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the longest subarray
      with at most k zero flips
    int longestOnes(vector<int>& nums, int k)
      {

        // Left pointer for the sliding window
        int left = 0;

        // Counter to track how many zeros are
          in the current window
        int zeros = 0;

        // Variable to store the maximum
          length found
        int maxLen = 0;

        // Iterate through the array using the
          right pointer
        for (int right = 0; right <
          nums.size(); right++) {

            // If current element is zero,
              increment zero counter
            if (nums[right] == 0) {
                zeros++;
            }

            // If number of zeros exceeds k,
              shrink the window from left
            while (zeros > k) {
                if (nums[left] == 0) {
                    zeros--;
                }
                // Move the left pointer
                  forward
                left++;
            }
            // Update the max length of valid
              window
            maxLen = max(maxLen, right - left
              + 1);
        }

        // Return the result
        return maxLen;
    }
};

// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N), Each element is processed at most once by the right pointer an*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the longest subarray
      with at most k zero flips
    int longestOnes(vector<int>& nums, int k)
      {

        // Left pointer of the sliding window
        int left = 0;

        // Counter for zeros in the current
          window
        int zerocount = 0;

        // Variable to store the maximum valid
          window length
        int maxlen = 0;

        // Right pointer expands the window
        for (int right = 0; right <
          nums.size(); right++) {

            // If current element is 0,
              increment zerocount
            if (nums[right] == 0) {
                zerocount++;
            }

            // If zero count exceeds k, move
              left pointer and adjust
              zerocount
            if (zerocount > k) {
                if (nums[left] == 0) {
                    zerocount--;
                }
                // Shrink the window from left
                left++;
            }

            // Update the maximum window size
            maxlen = max(maxlen, right - left
              + 1);
        }

        // Return the maximum length found
        return maxlen;
    }
};

// Driver code
```

Q196. Fruit Into Baskets

MEDIUM

Topic: Sliding Window & Two Pointer Combined Problems > Medium Problems

Practice Link

Input :fruits = [1, 2, 1]
Output :3
Explanation : We will start from first tree.
The first tree produces the fruit of kind '1' and we will put that in the first basket.
The second tree produces the fruit of kind '2' and we will put that in the second basket.
T... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(n), where n is the length of the input array. The sliding window e*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the maximum number of
      fruits we can collect with at most two
      fruit types
    int totalFruit(vector<int>& fruits) {
        // Hash map to store frequency of each
          fruit in the current window
        unordered_map<int, int> basket;

        // Variable to store the maximum
          length of valid window
        int maxFruits = 0;

        // Left pointer of the sliding window
        int left = 0;

        // Traverse the fruits array using
          right pointer
        for (int right = 0; right <
          fruits.size(); right++) {
            // Include the current fruit in
              the basket
            basket[fruits[right]]++;

            // Shrink the window if more than
              2 types of fruits exist
            while (basket.size() > 2) {
                // Decrease count of fruit at
                  left pointer
                basket[fruits[left]]--;

                // If count becomes zero,
                  remove it from map
                if (basket[fruits[left]] == 0)
                  {
basket.erase(fruits[left]);
                }

                // Move the left pointer to
                  shrink the window
                left++;
            }

            // Update the maximum window
              length found so far
            maxFruits = max(maxFruits, right -
              left + 1);
        }

        // Return the maximum fruits we can
          collect in a valid window
        return maxFruits;
    }
};

// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n), where n is the total number of elements in the input array. Sp*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the maximum number of
      fruits we can collect
    // with at most two types of fruits in the
      baskets.
    int totalFruit(vector<int>& fruits) {

        // Variable to store the maximum
          number of fruits collected
        int maxlen = 0;

        // Variables to track the last two
          fruit types and their counts
        int lastfruit = -1, secondlastfruit =
          -1;

        // Count of current window size and
          the last fruit streak
        int currcount = 0, lastfruitstreak =
          0;

        // Traverse through each fruit in the
          array
        for (int fruit : fruits) {

            // If current fruit is same as
              last or second last, increase
              window
            if (fruit == lastfruit || fruit ==
              secondlastfruit) {
                currcount++;
            } else {
                // Reset window size to size
                  of last fruit streak + 1
                  (current fruit)
                currcount = lastfruitstreak +
                  1;
            }

            // Update streak count
            if (fruit == lastfruit) {
                lastfruitstreak++;
            } else {
                lastfruitstreak = 1;
                secondlastfruit = lastfruit;
                lastfruit = fruit;
            }

            // Update the maximum length
            maxlen = max(maxlen, currcount);
        }

        return maxlen;
    }
};

// Driver code
```

Q197. Longest Repeating Character Replacement

HARD

Topic: Sliding Window & Two Pointer Combined Problems > Medium Problems

Practice Link

Input: s = "BAABAABBBAAA", k = 2
Output: 6
Explanation: We can change the B at index 0 and 3 (0-based indexing) to A. The new string becomes "AAAAAABBBAAA". The substring "AAAAAA" is the longest substring with the same letter, and its length is 6.
Input:... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N),We iterate through the entire string once using a sliding windo*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to return the length of the
      longest substring that can be made of
      repeating characters
    // by replacing at most k characters
    int characterReplacement(string s, int k)
      {

        // Map to count frequency of
          characters in current window
        unordered_map<char, int> freq;

        // Left pointer of the sliding window
        int left = 0;

        // Tracks the count of the most
          frequent character in the window
        int max_freq = 0;

        // Stores the result (maximum length
          found)
        int max_len = 0;

        // Traverse the string with right
          pointer
        for (int right = 0; right <
          s.length(); right++) {

            // Increase frequency of the
              current character
            freq[s[right]]++;

            // Update max frequency seen so
              far in the window
            max_freq = max(max_freq,
              freq[s[right]]);

            // If window is invalid (needs
              more than k replacements)
            while ((right - left + 1) -
              max_freq > k) {

                // Decrease frequency of the
                  character at left
                freq[s[left]]--;

                // Shrink the window from the
                  left
                left++;
            }

            // Update max_len with current
              valid window size
            max_len = max(max_len, right -
              left + 1);
        }

        // Return the final result
        return max_len;
    }
};

// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n), where n is the length of the string,each character is processe*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to return the length of the
      longest substring that can be made of
      repeating characters
    // by replacing at most k characters
    int characterReplacement(string s, int k)
      {
        // Frequency array for A-Z
        vector<int> freq(26, 0);

        // Left and right pointers of sliding
          window
        int left = 0, right = 0;

        // Tracks the count of the most
          frequent character in current window
        int maxCount = 0;

        // Stores the maximum length of valid
          window
        int maxLength = 0;

        // Iterate through the string with
          right pointer
        while (right < s.size()) {

            // Increment the frequency of
              current character
            freq[s[right] - 'A']++;

            // Update maxCount with the max
              frequency seen so far
            maxCount = max(maxCount,
              freq[s[right] - 'A']);

            // If the current window needs
              more than k replacements, move
              left
            while ((right - left + 1) -
              maxCount > k) {
                freq[s[left] - 'A']--;
                left++;
            }

            // Update the maximum window
              length
            maxLength = max(maxLength, right -
              left + 1);

            // Move right pointer forward
            right++;
        }

        // Return the maximum valid window
          length
        return maxLength;
    }
};

// Driver code
```

Q198. Binary Subarrays With Sum

HARD

Topic: Sliding Window & Two Pointer Combined Problems > Medium Problems

Practice Link

Input: nums = [1, 0, 0, 1, 1, 0], goal = 2
Output: 6
Explanation: There are 6 subarrays with sum exactly equal to 2:
[1, 0, 0, 1], [0, 0, 1, 1], [0, 1, 1], [1, 1], [1, 1, 0], [0,0,1,1,0]
Input: nums = [0,0,0,0,0,0], goal = 0
Output: 21
Explanation: All s... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(n), where n is the length of the input array . Each element is vis*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to count number of subarrays
      with sum equal to goal
    int numSubarraysWithSum(vector<int>& nums,
      int goal) {
        // Hashmap to store prefix sum
          frequencies
        unordered_map<int, int>
          prefixSumCount;

        // Initialize count of valid subarrays
          and current sum
        int count = 0, sum = 0;

        // Add base case: prefix sum 0 has
          frequency 1
        prefixSumCount[0] = 1;

        // Iterate through the array
        for (int num : nums) {
            // Add current element to prefix
              sum
            sum += num;

            // If (sum - goal) exists in map,
              add its frequency to count
            if (prefixSumCount.find(sum -
              goal) != prefixSumCount.end()) {
                count += prefixSumCount[sum -
                  goal];
            }

            // Increment frequency of current
              prefix sum
            prefixSumCount[sum]++;
        }

        // Return total count of valid
          subarrays
        return count;
    }
};

// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n), where n is the size of the input array.This is because the alg*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to calculate number of
      subarrays with sum exactly equal to goal
    int numSubarraysWithSum(vector<int>& nums,
      int goal) {
        // Return difference between subarrays
          with sum at most goal and at most
          (goal - 1)
        return atMost(nums, goal) -
          atMost(nums, goal - 1);
    }

private:
    // Helper function to compute number of
      subarrays with sum at most k
    int atMost(vector<int>& nums, int k) {
        // If k is negative, no such subarrays
          exist
        if (k < 0) return 0;

        int left = 0;
        int sum = 0;
        int count = 0;

        // Traverse the array using right
          pointer
        for (int right = 0; right <
          nums.size(); right++) {
            // Add current element to sum
            sum += nums[right];

            // Shrink the window from the left
              if sum exceeds k
            while (sum > k) {
                sum -= nums[left];
                left++;
            }

            // Add the number of valid
              subarrays ending at right
            count += (right - left + 1);
        }

        return count;
    }
};

// Driver code
```

Q199. Count number of Nice subarrays

HARD

Topic: Sliding Window & Two Pointer Combined Problems > Medium Problems

Practice Link

Input :nums = [1, 1, 2, 1, 1] , k = 3
Output :2
Explanation :The subarrays with three odd numbers are [1, 1, 2, 1] [1, 2, 1, 1]
Input : nums = [4, 8, 2] , k = 1
Output :0
Explanation :The array does not contain any odd number.

[BETTER] Better Approach

Complexity: *Time Complexity:O(N) ,We traverse the array once and each operation (map lookup, inse*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to count subarrays with
      exactly k odd numbers
    int numberOfSubarrays(vector<int>& nums,
      int k) {

        // Frequency map to track how often a
          certain odd count has occurred
        unordered_map<int, int> freq;

        // Initialize with 0 count of odd
          numbers seen so far
        freq[0] = 1;

        // Running count of odd numbers in the
          current prefix
        int oddCount = 0;

        // Total number of nice subarrays
        int result = 0;

        // Traverse through each element in
          the array
        for (int num : nums) {

            // Check if current number is odd
              and update count
            if (num % 2 == 1) oddCount++;

            // If there exists a prefix with
              (current odd count - k), add its
              frequency to result
            if (freq.count(oddCount - k)) {
                result += freq[oddCount - k];
            }

            // Update the frequency of current
              odd count
            freq[oddCount]++;
        }

        // Return the total number of valid
          subarrays
        return result;
    }
};

// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity:O(n) ,We scan the array two times using the sliding window helper. Ea*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // function to count subarrays with at
      most k odd numbers
    int countAtMost(vector<int>& nums, int k)
      {
        // Initialize variables
        int left = 0, res = 0;

        // Traverse through the array
        for (int right = 0; right <
          nums.size(); right++) {
            // If current number is odd,
              reduce k
            if (nums[right] % 2 != 0)
                k--;

            // Shrink the window until k is
              valid
            while (k < 0) {
                if (nums[left] % 2 != 0)
                    k++;
                left++;
            }

            // Add valid subarrays ending at
              right
            res += (right - left + 1);
        }

        // Return the count of valid subarrays
        return res;
    }

    // Main function to get number of
      subarrays with exactly k odd numbers
    int numberOfSubarrays(vector<int>& nums,
      int k) {
        return countAtMost(nums, k) -
          countAtMost(nums, k - 1);
    }
};

// Driver code
```

Q200. Number of Substrings Containing All Three Characters

HARD

Topic: Sliding Window & Two Pointer Combined Problems > Medium Problems

Practice Link

Input : s = "abcba"
Output : 5
Explanation : The substrings containing at least one occurrence of the characters 'a' , 'b' , 'c' are "abc" , "abcb" , "abcba" , "bcba" , "cba".
Input : s = "ccabcc"
Output : 8
Explanation : The substrings containing at least one occurrence of the characters 'a' , 'b' , 'c' are "ccab" , "ccabc" , "ccabcc" , "cab" , "cabc" , "cabcc" , "abc" , "abcc".

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity:O(n) ,We traverse the string once with the right pointer and adjust the left pointer in a line

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to count substrings containing at least one 'a', 'b', and 'c'
      using sliding window
    int numberOfSubstrings(string s) {
        // Initialize frequency map for 'a', 'b', and 'c'
        vector<int> freq(3, 0);

        // Initialize result to store count of valid substrings
        int res = 0;

        // Initialize left pointer of the sliding window
        int left = 0;

        // Traverse the string using right pointer
        for (int right = 0; right < s.length(); right++) {
            // Increment frequency of current character
            freq[s[right] - 'a']++;

            // Shrink the window from the left while all three characters are
              present
            while (freq[0] > 0 && freq[1] > 0 && freq[2] > 0) {
                // Count all substrings from current right to end
                res += (s.length() - right);

                // Decrease frequency of character at left and move left
                  forward
                freq[s[left] - 'a']--;
                left++;
            }
        }

        return res;
    }
};

// Driver code
```

Q201. Maximum Points You Can Obtain from Cards

MEDIUM

Topic: Sliding Window & Two Pointer Combined Problems > Medium Problems

Practice Link

Input :cardScore = [1, 2, 3, 4, 5, 6] , k = 3
Output : 15
Explanation :Choosing the rightmost cards will maximize your total score.
So optimal cards chosen are the rightmost three cards 4 , 5 , 6.
Th score is 4 + 5 + 6 => 15.
Input :cardScore = [5, 4, 1, 8, 7, 1, 3] , k = 3
Output :12
Explanation : In first step we will choose card from beginning with score of 5.
In second step we will choose the card from beginning again with score of 4.
In third step we will choose the card from end with sco... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(k) ,We calculate the initial sum of the first k cards , O(k) Then we slide the window k tim

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to return maximum score by picking k cards from either end
    int maxScore(vector<int>& cardPoints, int k) {
        // Get the total number of cards
        int n = cardPoints.size();

        // Calculate initial sum by picking first k cards from front
        int total = 0;
        for (int i = 0; i < k; ++i) {
            total += cardPoints[i];
        }

        // Store current max score
        int maxPoints = total;

        // Move the window from front to back k times
        for (int i = 0; i < k; ++i) {
            // Subtract card from front
            total -= cardPoints[k - 1 - i];

            // Add card from back
            total += cardPoints[n - 1 - i];

            // Update max score if needed
            maxPoints = max(maxPoints, total);
        }

        // Return the best score
        return maxPoints;
    }
};

// Driver code
```

Q202. Longest Substring With At Most K Distinct Characters

HARD

Topic: Sliding Window & Two Pointer Combined Problems > Hard Problems

Practice Link

Input :s = "aababbcaacc" , k = 2
Output :6
Explanation :The longest substring with at most two distinct characters is "aababb".
The length of the string 6

Input : s = "abcddefg" , k = 3
Output : 4
Explanation : The longest substring with at most three distinct characters is "bcdd".
The length of the string 4.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity:O(n) ,We iterate through the string once, and each character is added and removed from the map

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find longest substring with
      at most K distinct characters
    int
      lengthOfLongestSubstringKDistinct(string
      s, int k) {
        // Edge case: if k is 0 or string is
          empty
        if (k == 0 || s.empty()) return 0;

        // Hash map to store frequency of
          characters in current window
        unordered_map<char, int> freq;

        // Initialize left pointer of sliding
          window
        int left = 0;

        // Initialize variable to store
          maximum length
        int maxLen = 0;

        // Loop through the string using right
          pointer
        for (int right = 0; right <
          s.length(); right++) {
            // Include current character into
              frequency map
            freq[s[right]]++;

            // Shrink window if number of
              distinct characters exceeds k
            while (freq.size() > k) {
                freq[s[left]]--;

                // If character count becomes
                  zero, erase from map
                if (freq[s[left]] == 0) {
                    freq.erase(s[left]);
                }

                // Move left pointer ahead
                left++;
            }

            // Update maxLen with current
              valid window size
            maxLen = max(maxLen, right - left
              + 1);
        }

        // Return the final answer
        return maxLen;
    }
};

// Driver code
```

Q203. Subarrays with K Different Integers

MEDIUM

Topic: Sliding Window & Two Pointer Combined Problems > Hard Problems

Practice Link

Input: nums = [1, 2, 1, 2, 3], k = 2
Output: 7
Explanation: The 7 subarrays with exactly 2 different integers are:
[1,2], [2,1], [1,2], [2,3], [1,2,1], [2,1,2], [1,2,1,2]
Input: nums = [1, 2, 1, 3, 4], k = 3
Output: 3
Explanation: The 3 subarrays with exactly 3 different integers are:
[1,2,1,3], [2,1,3], [1,3,4]

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity:O(N) ,where n is the length of the array. Both calls to atMostK() are linear. Space Complexity

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Helper function to count subarrays with
      at most K distinct integers
    int atMostK(vector<int>& nums, int K) {
        unordered_map<int, int> freq;
        int left = 0, count = 0;

        // Traverse the array with right
          pointer
        for (int right = 0; right <
          nums.size(); right++) {
            // If it's a new unique element,
              decrease K
            if (freq[nums[right]] == 0) {
                K--;
            }

            // Increment frequency of current
              element
            freq[nums[right]]++;

            // Shrink the window if distinct
              count > K
            while (K < 0) {
                freq[nums[left]]--;
                if (freq[nums[left]] == 0) {
                    K++;
                }
                left++;
            }

            // Count all subarrays ending at
              current right
            count += (right - left + 1);
        }

        return count;
    }

    // Main function to return number of
      subarrays with exactly K distinct
      integers
    int subarraysWithKDistinct(vector<int>&
      nums, int k) {
        return atMostK(nums, k) -
          atMostK(nums, k - 1);
    }
};

// Driver code
```

Q204. Heaps (Theory Video)

EASY

Topic: Heaps [Learning, Medium, Hard Problems] > Learning

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```java
import java.util.*;
class BinaryHeap {
  static int capacity;
  static int size;
  static int arr[];
  BinaryHeap(int cap) {
    capacity = cap;
    size = 0;
    arr = new int[capacity];
  }
  static int parent(int i) {
    return (i - 1) / 2;
  }
  static int left(int i) {
    return 2 * i + 1;
  }
  static int right(int i) {
    return 2 * i + 2;
  }
  static void Insert(int x) {
    if (size == capacity) {
      System.out.println("Binary Heap
        Overflown");
      return;
    }
    arr[size] = x;
    int k = size;
    size++;
    while (k != 0 && arr[parent(k)] > arr[k])
      {
      int temp = arr[parent(k)];
      arr[parent(k)] = arr[k];
      arr[k] = temp;
      k = parent(k);
    }
  }
  static void Heapify(int ind) {
    int ri = right(ind);
    int li = left(ind);
    int smallest = ind;
    if (li < size && arr[li] < arr[smallest])
      smallest = li;
    if (ri < size && arr[ri] < arr[smallest])
      smallest = ri;
    if (smallest != ind) {
      int temp = arr[ind];
      arr[ind] = arr[smallest];
      arr[smallest] = temp;
      Heapify(smallest);
    }
  }
  static int getMin() {
    return arr[0];
  }
  static int ExtractMin() {
    if (size <= 0)
      return Integer.MAX_VALUE;
    if (size == 1) {
      size--;
      return arr[0];
    }
    int mini = arr[0];
    arr[0] = arr[size - 1];
    size--;
    Heapify(0);
    return mini;
  }
  static void Decreasekey(int i, int val) {
    arr[i] = val;
    while (i != 0 && arr[parent(i)] > arr[i])
      {
      int temp = arr[parent(i)];
      arr[parent(i)] = arr[i];
      arr[i] = temp;
      i = parent(i);
    }
  }
  static void Delete(int i) {
    Decreasekey(i, Integer.MIN_VALUE);
    ExtractMin();
  }
  static void print() {
    for (int i = 0; i < size; i++)
      System.out.print(arr[i] + " ");
    System.out.println();
  }
  public static void main(String args[]) {
    BinaryHeap h = new BinaryHeap(20);
    h.Insert(4);
    h.Insert(1);
    h.Insert(2);
    h.Insert(6);
    h.Insert(7);
    h.Insert(3);
    h.Insert(8);
    h.Insert(5);
    System.out.println("Min value is " +
      h.getMin());
    h.Insert(-1);
    System.out.println("Min value is " +
      h.getMin());
    h.Decreasekey(3, -2);
    System.out.println("Min value is " +
      h.getMin());
    h.ExtractMin();
    System.out.println("Min value is " +
      h.getMin());
    h.Delete(0);
    System.out.println("Min value is " +
      h.getMin());
  }
}
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```cpp
#include <bits/stdc++.h>
#include
using namespace std;
class BinaryHeap {
public:
    int capacity;
    int size;
    int* arr;
    BinaryHeap(int cap) {
        capacity = cap;
        size = 0;
        arr = new int[capacity];
    }
    int parent(int i) {
        return (i - 1) / 2;
    }
    int left(int i) {
        return 2 * i + 1;
    }
    int right(int i) {
        return 2 * i + 2;
    }
    void Insert(int x) {
        if (size == capacity) {
            cout << "Binary Heap Overflow" <<
              endl;
            return;
        }
        arr[size] = x;
        int k = size;
        size++;
        while (k != 0 && arr[parent(k)] >
          arr[k]) {
            swap(&arr[parent(k)], &arr[k]);
            k = parent(k);
        }
    }
    void Heapify(int ind) {
        int ri = right(ind);
        int li = left(ind);
        int smallest = ind;
        if (li < size && arr[li] <
          arr[smallest])
            smallest = li;
        if (ri < size && arr[ri] <
          arr[smallest])
            smallest = ri;
        if (smallest != ind) {
            swap(&arr[ind], &arr[smallest]);
            Heapify(smallest);
        }
    }
    int getMin() {
        return arr[0];
    }
    int ExtractMin() {
        if (size <= 0)
            return INT_MAX;
        if (size == 1) {
            size--;
            return arr[0];
        }
        int mini = arr[0];
        arr[0] = arr[size - 1];
        size--;
        Heapify(0);
        return mini;
    }
    void Decreasekey(int i, int val) {
        arr[i] = val;
        while (i != 0 && arr[parent(i)] >
          arr[i]) {
            swap(&arr[parent(i)], &arr[i]);
            i = parent(i);
        }
    }
    void Delete(int i) {
        Decreasekey(i, INT_MIN);
        ExtractMin();
    }
    void swap(int* x, int* y) {
        int temp = *x;
        *x = *y;
        *y = temp;
    }
    void print() {
        for (int i = 0; i < size; i++)
            cout << arr[i] << " ";
        cout << endl;
    }
};
```

Q205. Check if an array represents a min heap

MEDIUM

Topic: Heaps [Learning, Medium, Hard Problems] > Learning

Practice Link

Input: nums = [10, 20, 30, 21, 23]
Output: true
Explanation: Each node has a lower or equal value than its children.
Input: nums = [10, 20, 30, 25, 15]
Output: false
Explanation: The node with value 20 has a child with value 15, thus it is not a min-heap.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), where n is the number of elements in the array. We only iterate through the non-leaf no

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to check if the given array is a min-heap
    bool isMinHeap(vector<int>& nums) {
        int n = nums.size();

        // Iterate through all non-leaf nodes
        for (int i = 0; i <= (n / 2) - 1; i++) {

            // Calculate the left child index
            int left = 2 * i + 1;

            // If left child exists and is smaller than parent, not a min-heap
            if (left < n && nums[i] > nums[left]) {
                return false;
            }

            // Calculate the right child index
            int right = 2 * i + 2;

            // If right child exists and is smaller than parent, not a
              min-heap
            if (right < n && nums[i] > nums[right]) {
                return false;
            }
        }

        // If no violations found, it is a min-heap
        return true;
    }
};

// Driver code
```

Q206. K-th Largest element in an array

MEDIUM

Topic: Heaps [Learning, Medium, Hard Problems] > Medium Problems

Practice Link

Example 1:
Input: nums = [1, 2, 3, 4, 5], k = 2
Output: 4
Explanation: The 2nd largest number in the list [1, 2, 3, 4, 5] is 4.

Example 2:
Input: nums = [-5, 4, 1, 2, -3], k = 5
Output: -5
Explanation: The 5th largest number in the list [-5, 4, 1, 2, -3] is -5.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the size of the given array. In the average case (when the pivot is chosen r

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to get the Kth largest element
    int kthLargestElement(vector<int>& nums,
      int k) {
        // Return -1, if the Kth largest
          element does not exist
        if(k > nums.size()) return -1;
        // Pointers to mark the part of
          working array
        int left = 0, right = nums.size() - 1;
        // Until the Kth largest element is
          found
        while(true) {
            // Get the pivot index
            int pivotIndex = randomIndex(left,
              right);
            // Update the pivotIndex
            pivotIndex =
              partitionAndReturnIndex(nums,
              pivotIndex, left, right);
            // If Kth largest element is
              found, return
            if(pivotIndex == k-1) return
              nums[pivotIndex];
            // Else adjust the end pointers in
              array
            else if(pivotIndex > k-1) right =
              pivotIndex - 1;
            else left = pivotIndex + 1;
        }
        return -1;
    }
private:
    // Function to get a random index
    int randomIndex(int &left, int &right) {
        // length of the array
        int len = right - left + 1;
        // Return a random index from the
          array
        return (rand() % len) + left;
    }
    // Function to perform the partition and
      return the updated index of pivot
    int partitionAndReturnIndex(vector<int>
      &nums, int pivotIndex, int left, int
      right) {
        int pivot = nums[pivotIndex]; // Get
          the pivot element
        // Swap the pivot with the left
          element
        swap(nums[left], nums[pivotIndex]);
        int ind = left + 1; // Index to mark
          the start of right portion
        // Traverse on the array
        for(int i = left + 1; i <= right; i++)
          {
            // If the current element is
              greater than the pivot
            if(nums[i] > pivot) {
                // Place the current element
                  in the left portion
                swap(nums[ind], nums[i]);
                // Move the right portion
                  index
                ind++;
            }
        }
        swap(nums[left], nums[ind-1]); //
          Place the pivot at the correct index
        return ind-1; // Return the index of
          pivot now
    }
};
// Driver code
```

Q207. Kth smallest element in an array [use priority queue]

MEDIUM

Topic: Heaps [Learning, Medium, Hard Problems] > Medium Problems

Practice Link

Example 1:
Input: nums = [1, 2, 3, 4, 5], k = 2
Output: 4
Explanation: The 2nd largest number in the list [1, 2, 3, 4, 5] is 4.

Example 2:
Input: nums = [-5, 4, 1, 2, -3], k = 5
Output: -5
Explanation: The 5th largest number in the list [-5, 4, 1, 2, -3] is -5.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the size of the given array. In the average case (when the pivot is chosen r

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to get the Kth largest element
    int kthLargestElement(vector<int>& nums,
      int k) {
        // Return -1, if the Kth largest
          element does not exist
        if(k > nums.size()) return -1;
        // Pointers to mark the part of
          working array
        int left = 0, right = nums.size() - 1;
        // Until the Kth largest element is
          found
        while(true) {
            // Get the pivot index
            int pivotIndex = randomIndex(left,
              right);
            // Update the pivotIndex
            pivotIndex =
              partitionAndReturnIndex(nums,
              pivotIndex, left, right);
            // If Kth largest element is
              found, return
            if(pivotIndex == k-1) return
              nums[pivotIndex];
            // Else adjust the end pointers in
              array
            else if(pivotIndex > k-1) right =
              pivotIndex - 1;
            else left = pivotIndex + 1;
        }
        return -1;
    }
private:
    // Function to get a random index
    int randomIndex(int &left, int &right) {
        // length of the array
        int len = right - left + 1;
        // Return a random index from the
          array
        return (rand() % len) + left;
    }
    // Function to perform the partition and
      return the updated index of pivot
    int partitionAndReturnIndex(vector<int>
      &nums, int pivotIndex, int left, int
      right) {
        int pivot = nums[pivotIndex]; // Get
          the pivot element
        // Swap the pivot with the left
          element
        swap(nums[left], nums[pivotIndex]);
        int ind = left + 1; // Index to mark
          the start of right portion
        // Traverse on the array
        for(int i = left + 1; i <= right; i++)
          {
            // If the current element is
              greater than the pivot
            if(nums[i] > pivot) {
                // Place the current element
                  in the left portion
                swap(nums[ind], nums[i]);
                // Move the right portion
                  index
                ind++;
            }
        }
        swap(nums[left], nums[ind-1]); //
          Place the pivot at the correct index
        return ind-1; // Return the index of
          pivot now
    }
};
// Driver code
```

Q208. Sort K sorted array

EASY

Topic: Heaps [Learning, Medium, Hard Problems] > Medium Problems

Practice Link

Input : arr = [6, 5, 3, 2, 8, 10, 9], k = 3
Output : [2, 3, 5, 6, 8, 9, 10]
Explanation : The element 2 was at index 3, it moved to index 0. The element 3 was at index 2, it moved to index 1. The element 5 moved from index 1 to index 2. The element 6 moved from index 0 to index 3. The rest (8, 9, 10) were near their correct spots and shifted slightly.
Input : arr = [1, 4, 5, 2, 3, 6, 7, 8, 9, 10], k = 2
Output : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Explanation : The element 2 moved from... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n log k),We insert n elements into a min heap of size k + 1. Each insertion/removal from he

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to sort a k-sorted array using
      a min heap
    vector<int>
      sortNearlySortedArray(vector<int>& arr,
      int k) {
        // Create a min heap using
          priority_queue with greater
          comparator
        priority_queue<int, vector<int>,
          greater<int>> minHeap;

        // Store the final sorted result
        vector<int> result;

        // Push first k+1 elements into the
          heap
        for (int i = 0; i <= k && i <
          arr.size(); i++) {
            minHeap.push(arr[i]);
        }

        // Process the remaining elements of
          the array
        for (int i = k + 1; i < arr.size();
          i++) {
            // Push the smallest element from
              the heap to the result
            result.push_back(minHeap.top());
            minHeap.pop();

            // Push the current element into
              the heap
            minHeap.push(arr[i]);
        }

        // Pop remaining elements from the
          heap
        while (!minHeap.empty()) {
            result.push_back(minHeap.top());
            minHeap.pop();
        }

        // Return the sorted array
        return result;
    }
};

// Driver code
```

Q209. Merge K sorted Lists

HARD

Topic: Heaps [Learning, Medium, Hard Problems] > Medium Problems

Practice Link

Input : heads = [[head -> 1 -> 2 -> 3 -> 4], [head -> -4 -> -3], [head -> -5 -> -3 -> 1 -> 2 -> 3 -> 4]]
Output: head -> -5 -> -4 -> -3 -> -3 -> 1 -> 1 -> 2 -> 2 -> 3 -> 3 -> 4 -> 4
Explanation: head -> -5 -> -4 -> -3 -> -3 -> 1 -> 1 -> 2 -> 2 -> 3 -> 3 -> 4 -> 4
The nodes in bold come from the 3rd list, the underlined nodes come from the 2nd list, rest are from the 1st list.
Input : heads = [[head -> -5 -> -4 -> -1], [head -> 10 -> 11 -> 12]]
Output : head -> -5 -> -4 -> -1 -> 10 -> 11 -> 12... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity:O(N * log K) ,Where N is the total number of nodes across all K linked lists. Each insertion a

```cpp
#include <bits/stdc++.h>
using namespace std;
// Definition for singly-linked list
struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(NULL) {}
};
class Compare {
public:
    // Comparator to order ListNode pointers
      based on node values
    bool operator()(ListNode* a, ListNode* b)
      {
        return a->val > b->val;
    }
};
class Solution {
public:
    // Function to merge k sorted linked lists
      using a min-heap
    ListNode* mergeKLists(vector<ListNode*>&
      lists) {
        // Create a min-heap (priority queue)
          with custom comparator
        priority_queue<ListNode*,
          vector<ListNode*>, Compare> pq;
        // Push the head of each non-empty
          list into the heap
        for (auto list : lists) {
            if (list != NULL)
                pq.push(list);
        }
        // Create a dummy node to build the
          result list
        ListNode* dummy = new ListNode(0);
        ListNode* tail = dummy;
        // While the heap is not empty
        while (!pq.empty()) {
            // Extract the node with the
              smallest value
            ListNode* smallest = pq.top();
            pq.pop();
            // Add it to the result list
            tail->next = smallest;
            tail = tail->next;
            // If there's a next node, push it
              into the heap
            if (smallest->next != NULL)
                pq.push(smallest->next);
        }
        // Return the head of the merged list
        return dummy->next;
    }
};
// Driver code
```

Q210. Replace Elements by Their Rank

EASY

Topic: Heaps [Learning, Medium, Hard Problems] > Medium Problems

Practice Link

Input: 20 15 26 2 98 6
Output: 4 3 5 1 6 2
Explanation: When sorted,the array is 2,6,15,20,26,98. So the rank of 2 is 1,rank of 6 is 2,rank of 15 is 3 and so.

Input: 1 5 8 15 8 25 9
Output: 1 2 3 5 3 6 4
Explanation :When sorted,the array is 1,5,8,8,9,15,25. So the rank of 1 is 1,rank of 5 is 2,rank of 8 is 3 and so.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N log N), We sort the array which takes O(N log N), then loop over the array multiple times

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to replace elements by their
      rank in the array
    vector<int> replaceWithRank(vector<int>&
      arr) {
        // Copy the original array for sorting
        vector<int> sortedArr = arr;

        // Sort the array
        sort(sortedArr.begin(),
          sortedArr.end());

        // Create a map to store rank of each
          unique number
        unordered_map<int, int> rankMap;

        // Initialize rank counter
        int rank = 1;

        // Assign rank to each unique element
          in sorted order
        for (int num : sortedArr) {
            // If this number is not already
              assigned a rank
            if (rankMap.find(num) ==
              rankMap.end()) {
                rankMap[num] = rank;
                rank++;
            }
        }

        // Replace each element in original
          array with its rank
        vector<int> result;
        for (int num : arr) {
            result.push_back(rankMap[num]);
        }

        return result;
    }
};

// Driver code
```

Q211. Task Scheduler

MEDIUM

Topic: Heaps [Learning, Medium, Hard Problems] > Medium Problems

Practice Link

Input : tasks = ["A","A","A","B","B","B"], n = 2
Output : 8
Explanation : One valid execution order is:
A -> B -> idle -> A -> B -> idle -> A -> B
Total intervals = 8
Input : tasks = ["A","C","A","B","D","B"], n = 1
Output : 6
Explanation A possible execution:
A -> B -> C -> D -> A -> B
No idle interval is needed as cooldown = 1.

[OPTIMAL SOLUTION] Approach — Time Complexity:O(N log K), We count frequencies in O(N) and use a max heap of K unique tasks. Each task can

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    int leastInterval(vector<char>& tasks, int n) {
        unordered_map<char, int> freq;
        for (char task : tasks) {
            freq[task]++;
        }
        priority_queue<int> maxHeap;
        for (auto& entry : freq) {
            maxHeap.push(entry.second);
        }
        int time = 0;
        while (!maxHeap.empty()) {
            vector<int> temp;
            int cycle = n + 1;
            int i = 0;
            while (i < cycle && !maxHeap.empty()) {
                int cnt = maxHeap.top();
                maxHeap.pop();
                cnt--;
                if (cnt > 0) {
                    temp.push_back(cnt);
                }
                time++;
                i++;
            }
            for (int val : temp) {
                maxHeap.push(val);
            }
            if (maxHeap.empty()) break;
            time += (cycle - i);
        }
        return time;
    }
};
```

Q212. Hand of Straights

MEDIUM

Topic: Heaps [Learning, Medium, Hard Problems] > Medium Problems

Practice Link

Input : hand = [1,2,3,6,2,3,4,7,8], groupSize = 3
Output : True
Explanation : One possible partition is [1,2,3] [2,3,4] [6,7,8].
Input : hand = [1,2,3,4,5], groupSize = 4
Output : false
Explanation : There is no way to split the hand into groups of 4 consecutive cards.

[OPTIMAL SOLUTION] Approach — Time Complexity:O(N log N) , where N is the number of cards. We insert each card into a map which takes log N

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    bool isNStraightHand(vector<int>& hand,
      int groupSize) {
        // If total cards can't be divided
          evenly, return false
        if (hand.size() % groupSize != 0)
          return false;

        // Store the frequency of each card
        map<int, int> freq;
        for (int card : hand) {
            freq[card]++;
        }

        // Create an iterator pointing to the
          beginning of the frequency map
        auto it = freq.begin();

        // Loop through all keys in the map
        while (it != freq.end()) {

            // Skip cards that are already
              used up
            if (it->second == 0) {
                ++it;
                continue;
            }

            // Store the starting card of the
              group
            int start = it->first;

            // Number of times we need to form
              this group
            int count = it->second;

            // Try to form a group of
              consecutive cards of size
              groupSize
            for (int i = 0; i < groupSize;
              ++i) {

                // If the current required
                  card is missing or doesn't
                  have enough frequency
                if (freq[start + i] < count)
                  return false;

                // Decrease the frequency for
                  current card
                freq[start + i] -= count;
            }

            // Move the iterator to the next
              element
            ++it;
        }

        // All groups formed successfully
        return true;
    }
};

// Driver code
```

Q213. Design Twitter

MEDIUM

Topic: Heaps [Learning, Medium, Hard Problems] > Hard Problems

Practice Link

Input: [postTweet(1, 2), postTweet(2, 6), getNewsFeed(1), follow(1, 2), getNewsFeed(1), unfollow(1, 2), postTweet(1,7), getNewsFeed(1)]
Output: [null, null, [2], null, [6, 2], null, [7, 2]]
Explanation:postTweet(1, 2): User with userId 1 posts tweet with tweetId 2.
postTweet(2, 6): User with userId 2 posts tweet with tweetId 6.
getNewsFeed(1): Outputs feed of userId 1 = [2].
follow(1, 2): User with userId 1 follows user with userId 2.
getNewsFeed(1): Outputs feed of userId 1 = [6, 2], it contain... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: postTweet, follow, and unfollow run in O(1). getNewsFeed takes O(log k) where k is the nu

```cpp
#include <bits/stdc++.h>
using namespace std;
class Twitter {
private:
    unordered_map<int, vector<pair<int, int>>>
      tweets;
    unordered_map<int, unordered_set<int>>
      following;
    int time;
public:
    Twitter() {
        time = 0;
    }
    void postTweet(int userId, int tweetId) {
        tweets[userId].push_back({time++,
          tweetId});
    }
    vector<int> getNewsFeed(int userId) {
        priority_queue<pair<int, int>,
          vector<pair<int, int>>, greater<>>
          pq;
        for (auto& t : tweets[userId]) {
            pq.push(t);
            if (pq.size() > 10)
                pq.pop();
        }
        for (int followee : following[userId])
          {
            for (auto& t : tweets[followee]) {
                pq.push(t);
                if (pq.size() > 10)
                    pq.pop();
            }
        }
        vector<int> res;
        while (!pq.empty()) {
            res.push_back(pq.top().second);
            pq.pop();
        }
        reverse(res.begin(), res.end());
        return res;
    }
    void follow(int followerId, int
      followeeId) {
following[followerId].insert(followeeId);
    }
    void unfollow(int followerId, int
      followeeId) {
following[followerId].erase(followeeId);
    }
};
```

Q214. Kth largest element in a stream of running integers

HARD

Topic: Heaps [Learning, Medium, Hard Problems] > Hard Problems

Practice Link

Input: [KthLargest(3, [1, 2, 3, 4]), add(5), add(2), add(7)]
Output: [null, 3, 3, 4]
Explanation: initial stream = [1, 2, 3, 4], k = 3.
add(5): stream = [1, 2, 3, 4, 5] -> returns 3
add(2): stream = [1, 2, 2, 3, 4, 5] -> returns 3
add(7): stream = [1, 2, 2, 3, 4, 5, 7] -> returns 4
Input: [KthLargest(2, [5, 5, 5, 5], add(2), add(6), add(60)]
Output: [null, 5, 5, 6]
Explanation: initial stream = [5, 5, 5, 5], k = 2.
add(2): stream = [5, 5, 5, 5, 2] -> returns 5
add(6): stream = [5, 5, 5, 5, 2, 6]... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(log k), per insertion each insertion into the min-heap takes O(log k) time. Since we mainta

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
    // Min-heap to store the top k largest
      elements
    priority_queue<int, vector<int>,
      greater<int>> minHeap;
    int size;

public:
    // Constructor to initialize the heap with
      initial elements
    Solution(int k, vector<int>& nums) {
        size = k;

        // Add each number to the heap
        for (int num : nums) {
            minHeap.push(num);

            // If heap exceeds size k, remove
              the smallest element
            if (minHeap.size() > k) {
                minHeap.pop();
            }
        }
    }

    // Adds a new value to the stream and
      returns the k-th largest
    int add(int val) {
        // Insert the new element
        minHeap.push(val);

        // If size exceeds k, remove the
          smallest (to maintain top k largest)
        if (minHeap.size() > size) {
            minHeap.pop();
        }

        // The top of min-heap is the k-th
          largest
        return minHeap.top();
    }
};

// Driver code
```

Q215. Maximum Sum Combination

HARD

Topic: Heaps [Learning, Medium, Hard Problems] > Hard Problems

Practice Link

Input : nums1 = [7, 3], nums2 = [1, 6], k = 2
Output : [13, 9]
Explanation : The 2 maximum combinations are made by: nums1[0] + nums2[1] = 13 nums1[1] + nums2[1] = 9

Input : nums1 = [3, 4, 5], nums2 = [2, 6, 3], k = 2
Output : [11, 10]
Explanation : The 2 maximum combinations are made by: nums1[2] + nums2[1] = 11 nums1[1] + nums2[1] = 10

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(k * log k),The loop runs k times, and in each iteration, we perform heap insertions and del

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find k maximum sum
      combinations
    vector<int> maxCombinations(vector<int>&
      nums1, vector<int>& nums2, int k) {
        // Sort both arrays in descending
          order
        sort(nums1.begin(), nums1.end(),
          greater<int>());
        sort(nums2.begin(), nums2.end(),
          greater<int>());
        // Max-heap to store pairs with their
          indices
        priority_queue<tuple<int, int, int>>
          maxHeap;
        // Set to keep track of visited index
          pairs
        set<pair<int, int>> visited;
        // Push the initial maximum pair
          (nums1[0] + nums2[0])
        maxHeap.push({nums1[0] + nums2[0], 0,
          0});
        visited.insert({0, 0});
        // Vector to store the result
        vector<int> result;
        // Extract top k elements from the
          heap
        while(k-- && !maxHeap.empty()) {
            // Get the current maximum sum and
              its indices
            auto [sum, i, j] = maxHeap.top();
            maxHeap.pop();
            // Add this sum to the result
            result.push_back(sum);
            // If (i + 1, j) is valid and not
              visited, add it to the heap
            if(i + 1 < nums1.size() &&
              !visited.count({i + 1, j})) {
                maxHeap.push({nums1[i + 1] +
                  nums2[j], i + 1, j});
                visited.insert({i + 1, j});
            }
            // If (i, j + 1) is valid and not
              visited, add it to the heap
            if(j + 1 < nums2.size() &&
              !visited.count({i, j + 1})) {
                maxHeap.push({nums1[i] +
                  nums2[j + 1], i, j + 1});
                visited.insert({i, j + 1});
            }
        }
        // Return the final k max combinations
        return result;
    }
};
// Driver code
```

Q216. Find Median from Data Stream

HARD

Topic: Heaps [Learning, Medium, Hard Problems] > Hard Problems

Practice Link

Input : [MedianFinder(), addNum(1), addNum(2), addNum(3), findMedian()]
Output : [null, null, null, null, 2]
Explanation : MedianFinder() initializes the object. addNum(1) adds 1 to the data stream → [1] addNum(2) adds 2 → [1, 2] addNum(3) adds 3 → [1, 2, 3] findMedian() returns 2 as the median of [1, 2, 3]
Input : [MedianFinder(), addNum(1), addNum(6), findMedian(), addNum(3), findMedian()]
Output : [null, null, null, 3.5, null, 3]
Explanation : MedianFinder() initializes the object. addNum(1)... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity:O(N log N + M) , Because each call to addNum() takes O(log N), and each call to findMedian() i

```cpp
#include <bits/stdc++.h>
using namespace std;

class MedianFinder {
private:
    // Max-heap to store the smaller half of
      the data
    priority_queue<int> maxHeap;

    // Min-heap to store the larger half of
      the data
    priority_queue<int, vector<int>,
      greater<int>> minHeap;

public:
    // Constructor to initialize MedianFinder
      object
    MedianFinder() {
    }

    // Function to add a new number to the
      data stream
    void addNum(int num) {
        // Step 1: Add to maxHeap first
        maxHeap.push(num);

        // Step 2: Balance by pushing the
          largest from maxHeap to minHeap
        minHeap.push(maxHeap.top());
        maxHeap.pop();

        // Step 3: If minHeap has more
          elements, move top back to maxHeap
        if (minHeap.size() > maxHeap.size()) {
            maxHeap.push(minHeap.top());
            minHeap.pop();
        }
    }

    // Function to find the current median
    double findMedian() {
        // If both heaps are of equal size,
          take the average of the tops
        if (maxHeap.size() == minHeap.size())
          {
            return (maxHeap.top() +
              minHeap.top()) / 2.0;
        }

        // If odd size, the top of maxHeap is
          the median
        return maxHeap.top();
    }
};

// Driver code
```

Q217. Assign Cookies

EASY

Topic: Greedy Algorithms [Easy, Medium/Hard] > Easy Problems

Practice Link

Input : Student = [1, 2, 3] , Cookie = [1, 1]
Output :1
Explanation : Only the first cookie (1) satisfies the first student (1), therefore only 1 student is content.

Input : Student = [1, 2] , Cookie = [1, 2, 3]
Output : 2
Explanation : Cookie 1 satisfies student 1 and cookie 2 satisfies student 2. Therefore, 2 students are content.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n*logn + m*logm), Both the arrays are sorted in increasing order.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the maximum number of content students
    int findContentChildren(vector<int>& student, vector<int>& cookie) {
        // Sort both arrays to apply the greedy strategy
        sort(student.begin(), student.end());
        sort(cookie.begin(), cookie.end());

        int studentIndex = 0;
        int cookieIndex = 0;

        // Try to assign cookies until any one list is fully processed
        while (studentIndex < student.size() && cookieIndex < cookie.size()) {
            // If the cookie satisfies the student's greed
            if (cookie[cookieIndex] >= student[studentIndex]) {
                studentIndex++;
            }
            // Move to next cookie in both cases
            cookieIndex++;
        }

        // Number of students satisfied is equal to studentIndex
        return studentIndex;
    }
};
```

Q218. Fractional Knapsack

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Easy Problems

Practice Link

Example 1:
Input: val = [60, 100, 120], wt = [10, 20, 30], capacity = 50
Output: 240.000000
Explanation:
- Take item 0 (w = 10, v = 60)
- Take item 1 (w = 20, v = 100)
- Take 2/3 of item 2 (w = 20, v = 80)
Total value = 60 + 100 + 80 = 240

Example 2:
Input: val = [60, 100], wt = [10, 20], capacity = 50
Output: 160.000000
Explanation:
Both items fit entirely since total weight 10 + 20 = 30 ≤ 50.
Total value = 60 + 100 = 160

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n log n + n). O(n log n) to sort the items and O(n) to iterate through all the items for ca

```cpp
#include<bits/stdc++.h>
using namespace std;

// Struct to represent an item with value and
  weight
struct Item {
    int value;
    int weight;
};

class Solution {
public:
    // Comparator function to sort items by
      value/weight ratio
    bool static comp(Item a, Item b) {
        double r1 = (double) a.value /
          (double) a.weight;
        double r2 = (double) b.value /
          (double) b.weight;
        return r1 > r2;  // Return true if the
          ratio of item a is greater than item
          b
    }

    // Function to calculate the maximum value
      we can get with fractional knapsack
    double fractionalKnapsack(int W, Item
      arr[], int n) {

        // Sort items based on the
          value/weight ratio
        sort(arr, arr + n, comp);

        int curWeight = 0;  // Current weight
          of knapsack
        double finalvalue = 0.0;  // Final
          value we can achieve

        // Iterate through the sorted items
        for (int i = 0; i < n; i++) {

            // If the current item can be
              fully added to the knapsack
            if (curWeight + arr[i].weight <=
              W) {
                curWeight += arr[i].weight;
                finalvalue += arr[i].value;
                  // Add the full value of the
                  item
            } else {
                // If the current item can't
                  be fully added, take the
                  fractional part
                int remain = W - curWeight;
                finalvalue += (arr[i].value /
                  (double) arr[i].weight) *
                  (double) remain;
                break;  // Break as we have
                  filled the knapsack
            }
        }

        return finalvalue;  // Return the
          maximum value that can be carried
    }
};
```

Q219. Lemonade Change

EASY

Topic: Greedy Algorithms [Easy, Medium/Hard] > Easy Problems

Practice Link

Example 1:
Input: bills = [5, 5, 5, 10, 20]
Output: True
Explanation:
Initially we have 0 change and the queue of customers is [5, 5, 5, 10, 20].
First Customer pays 5$, no change required.
Second Customer pays 5$, no change required.
Third Customer pays 5$, no change required.
The Fourth Customer pays 10$, out of the three 5$ bills we have, we pay a 5$ bill and accept the 10$ bill.
Fifth Customer pays 20$, out of the two 5$ bills and one 10$ bill we have, we pay 15$ in change and have... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N) where N is the number of people in queue/ bills we will deal with. We iterate through ea

```cpp
#include <vector>
#include <iostream>
using namespace std;
// Class that handles the Lemonade Change
  problem
class LemonadeStand {
public:
    // Function to check if all customers can
      be given correct change
    bool lemonadeChange(vector<int>& bills) {
        int five = 0; // Counter for $5 bills
        int ten = 0;  // Counter for $10 bills
        // Process each customer's bill
        for (int bill : bills) {
            if (bill == 5) {
                // Customer pays with $5 -> no
                  change needed
                five++;
            }
            else if (bill == 10) {
                // Customer pays with $10 ->
                  needs $5 as change
                if (five > 0) {
                    five--; // Give one $5 as
                      change
                    ten++;  // Accept the $10
                      bill
                } else {
                    return false; // Cannot
                      provide change
                }
            }
            else { // bill == 20
                // Customer pays with $20 ->
                  needs $15 as change
                if (five > 0 && ten > 0) {
                    five--; // Use one $5
                    ten--;  // Use one $10
                }
                else if (five >= 3) {
                    five -= 3; // Use three $5
                      bills
                }
                else {
                    return false; // Cannot
                      provide change
                }
            }
        }
        return true; // Successfully gave
          change to all customers
    }
};
```

Q220. Valid Paranthesis Checker

HARD

Topic: Greedy Algorithms [Easy, Medium/Hard] > Easy Problems

Practice Link

Input :s = (*))
Output : True
Explanation :The * can be replaced by an opening '(' bracket. The string after replacing the * mark is "(())" and is a valid string.
Input : s = *(()
Output :false
Explanation :The * replaced with any bracket does not form a valid string.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the length of the input string.We traverse the string once in a single pass,

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to validate the string using greedy approach
    bool isValid(string s) {
        // Track minimum and maximum open brackets
        int minOpen = 0, maxOpen = 0;

        // Traverse each character in the string
        for (char c : s) {
            if (c == '(') {
                minOpen++;
                maxOpen++;
            } else if (c == ')') {
                minOpen--;
                maxOpen--;
            } else {
                // Treat '*' as '(', ')' or ''
                minOpen--;
                maxOpen++;
            }

            // If maxOpen goes negative, too many closing brackets
            if (maxOpen < 0) return false;

            // minOpen can't be negative
            minOpen = max(minOpen, 0);
        }

        // String is valid if all opens are closed
        return minOpen == 0;
    }
};

// Driver code
```

Q221. N meetings in one room

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Input: N = 6, start[] = {1,3,0,5,8,5}, end[] = {2,4,5,7,9,9}
Output: [1, 2, 4, 5]
Explanation: These meeting can be conducted in the room.

Input: N = 2, start[] = {1,5}, end[] = {7,8}
Output: [1]
Explanation: Any one out of the two meeting can take place.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N*logN) + O(N), We sort the entire start and end time array and then iterate over every int

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to hold solution logic
class Solution {
public:
    // Function to get all meetings that can
      be scheduled
    vector<int> maxMeetings(vector<int>&
      start, vector<int>& end) {
        // Store meetings as (end_time,
          start_time, original_index)
        vector<tuple<int, int, int>> meetings;
        for (int i = 0; i < start.size(); i++)
          {
            // i+1 for 1-based indexing
            meetings.push_back({end[i],
              start[i], i + 1});

        }

        // Sort by end time
        sort(meetings.begin(),
          meetings.end());

        vector<int> result; // To store
          meeting indices
        int lastEnd = -1;

        // Traverse sorted meetings
        for (auto& m : meetings) {
            int e = get<0>(m);
            int s = get<1>(m);
            int idx = get<2>(m);

            // If meeting starts after last
              one ends
            if (s > lastEnd) {
                // Store index
                result.push_back(idx);
                // Update last end time
                lastEnd = e;
            }
        }
        return result;
    }
};

// Main driver code
```

Q222. Jump Game - I

EASY

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Example 1:
Input:nums = [2, 3, 1, 0, 4]
Output: True
Explanation:
We start at index 0, with value 2 this means we can jump to index 1 or 2.
From index 1, with value 3, we can jump to index 2, 3, or 4. However, if we jump to index 2 with value 1, we can only jump to index 3.
So we jump to index 1 then index 4 reaching the end of the array.
Hence, we return true.

Example 2:
Input:nums = [3, 2, 1, 0, 4]
Output: False
Explanation:
We start at index 0, with value 3 which means we can j... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N) where N is the length of the input array. We iterate through the input array exactly onc

```cpp
#include <vector>
using namespace std;

// Class to solve the Jump Game problem
class JumpGame {
public:
    // Function to determine if you can reach the last index
    bool canJump(vector<int>& nums) {
        int maxIndex = 0; // The farthest index we can currently reach

        // Traverse the array
        for (int i = 0; i < nums.size(); i++) {
            // If current index is beyond the farthest reachable point
            if (i > maxIndex) {
                return false; // We cannot move further
            }

            // Update the farthest index we can reach
            maxIndex = max(maxIndex, i + nums[i]);
        }

        // If we finish the loop, we can reach the last index
        return true;
    }
};
```

Q223. Jump Game II

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Input: nums = [2, 3, 1, 1, 4]

Output: 2
Explanation: Jump from index 0 → 1 → 4.

Input: nums = [2, 3, 0, 1, 4]

Output: 2
Explanation: Jump from index 0 → 1 → 4.

[BETTER] Better Approach

Complexity: *Time Complexity: O(N2) ,We use two nested loops where outer loop runs for N elements*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to compute the minimum number
      of jumps to reach the last index
    int jump(vector<int>& nums) {
        // Get the size of input array
        int n = nums.size();

        // Initialize DP array with large
          value
        vector<int> dp(n, INT_MAX);

        // It takes 0 jumps to reach the
          starting index
        dp[0] = 0;

        // Iterate through all indices
        for (int i = 0; i < n; ++i) {
            // For each position, calculate
              max jump
            for (int j = 1; j <= nums[i] && i
              + j < n; ++j) {
                // Update dp value for the
                  position we can reach
                dp[i + j] = min(dp[i + j],
                  dp[i] + 1);
            }
        }

        // Return the minimum jumps to reach
          last index
        return dp[n - 1];
    }
};

// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N),We traverse the entire array once from start to end. Space Com*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to compute the minimum number
      of jumps to reach the last index
    int jump(vector<int>& nums) {
        // Initialize variables to keep track
          of range and jumps
        int jumps = 0, currentEnd = 0,
          farthest = 0;

        // Traverse through the array
          (excluding the last element)
        for (int i = 0; i < nums.size() - 1;
          ++i) {
            // Update the farthest index that
              can be reached so far
            farthest = max(farthest, i +
              nums[i]);

            // When we reach the end of the
              current jump range
            if (i == currentEnd) {
                // We need to make a jump
                jumps++;

                // Move the current end to the
                  farthest index we can reach
                currentEnd = farthest;
            }
        }

        // Return the total jumps needed
        return jumps;
    }
};
// Driver Code
```

Q224. Minimum number of platforms required for a railway

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Input: N=6,
arr[] = {9:00, 9:45, 9:55, 11:00, 15:00, 18:00}
dep[] = {9:20, 12:00, 11:30, 11:50, 19:00, 20:00}
Output: 3
Explanation: There are at-most three trains at a time. The train at 11:00 arrived but the trains which had arrived at 9:45 and 9:55 have still not departed. So, we need at least three platforms here.

Input : N=2,
arr[]={10:20,12:00}
dep[]={10:50,12:30}
Output: 1
Explanation : Before the arrival of the new train, the earlier train already departed. So, we don't require mul... (cont.)

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N*logN), we sort both the arrival and departure arrays. Space C

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find minimum platforms required
    int countPlatforms(int n, int arr[], int dep[]) {
        // Sort the arrival and departure times
        sort(arr, arr + n);
        sort(dep, dep + n);

        // Initialize pointers and counters
        int platforms = 1;
        int result = 1;
        int i = 1, j = 0;

        // Traverse both arrays
        while (i < n && j < n) {
            // If next train arrives before current one departs
            if (arr[i] <= dep[j]) {
                // One more platform needed
                platforms++;
                i++;
            } else {
                // One train departs, platform freed
                platforms--;
                j++;
            }

            // Update maximum required platforms
            result = max(result, platforms);
        }

        return result;
    }
};
```

Q225. Job sequencing Problem

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Example 1:
Input:
N = 4, Jobs = {(1, 4, 20), (2, 1, 10), (3, 1, 40), (4, 1, 30)}
Output: 2 60
Explanation:
- The 3rd job with a deadline of 1 is performed during the first unit of time.
- The 1st job is performed during the second unit of time as its deadline is 4.
Profit = 40 + 20 = 60.
So, the result is 2 jobs with a total profit of 60.

Example 2:
Input:
N = 5, Jobs = {(1, 2, 100), (2, 1, 19), (3, 2, 27), (4, 1, 25), (5, 1, 15)}
Output: 2 127
Explanation:
The first a... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N log N) + O(N * M), O(N log N) for sorting the jobs in decreasing order of profit. O(N * M

```cpp
#include<bits/stdc++.h>
using namespace std;
// A structure to represent a job
struct Job {
   // Job Id
   int id;
   // Deadline of job
   int dead;
   // Profit if job is completed before or on
     the deadline
   int profit;
};
class Solution {
   public:
      // Comparator function to sort jobs in
        decreasing order of profit
      bool static comparison(Job a, Job b) {
          // Sort by profit in descending
            order
         return (a.profit > b.profit);
      }
   // Function to find the maximum profit and
     the number of jobs done
   pair < int, int > JobScheduling(Job arr[],
     int n) {
      // Sort the jobs based on decreasing
        profit
      sort(arr, arr + n, comparison);
      // Find the maximum deadline among all
        jobs
      int maxi = arr[0].dead;
      for (int i = 1; i < n; i++) {
        // Get the latest deadline
        maxi = max(maxi, arr[i].dead);
      }
      // Initialize the slot array to track
        which time slots are taken
      int slot[maxi + 1];
      for (int i = 0; i <= maxi; i++)
        // Mark all slots as unoccupied
          initially
        slot[i] = -1;
      int countJobs = 0, jobProfit = 0;
      // Iterate through all jobs to find the
        best possible schedule
      for (int i = 0; i < n; i++) {
         // Find a slot for the current job
           (starting from its deadline)
         for (int j = arr[i].dead; j > 0; j--)
           {
           // If the slot is available
           if (slot[j] == -1) {
               // Assign the job to the slot
               slot[j] = i;
               // Increment the number of
                 jobs done
               countJobs++;
               // Add the profit of the job
               jobProfit += arr[i].profit;
              break;
           }
         }
      }
      // Return the number of jobs done and
        the total profit
      return make_pair(countJobs, jobProfit);
   }
};
```

Q226. Candy

HARD

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Input: ratings = [1, 0, 5]
Output: 5
Explanation: The distribution of candies will be 2, 1, 2 to the first, second, and third child respectively.
Input: ratings = [1, 2, 2]
Output: 4
Explanation: The distribution of candies will be 1, 2, 1 to the first... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(n) ,The algorithm makes two passes over the ratings array (left to*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
   // Function to calculate the minimum number
     of candies
    int candy(vector<int>& ratings) {
        // Initialize each child with 1 candy
        int n = ratings.size();
        vector<int> candies(n, 1);

        // Traverse from left to right
        for (int i = 1; i < n; ++i) {
            // If current rating is higher
              than previous, give more candies
            if (ratings[i] > ratings[i - 1])
                candies[i] = candies[i - 1] +
                  1;
        }

        // Traverse from right to left
        for (int i = n - 2; i >= 0; --i) {
            // If current rating is higher
              than next, adjust candy count
            if (ratings[i] > ratings[i + 1])
                candies[i] = max(candies[i],
                  candies[i + 1] + 1);
        }

        // Sum up all candies
        return accumulate(candies.begin(),
          candies.end(), 0);
    }
};

// Driver code
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(n), We iterate through the ratings array only once using a single*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
  // Function to calculate the minimum number
    of candies
    int candy(vector<int>& ratings) {
        // Get number of children
        int n = ratings.size();
        // Initially give 1 candy to each
          child
        int candies = n;
        // Start from second child
        int i = 1;
        while (i < n) {
            // Skip equal ratings, no need to
              change candy count
            if (ratings[i] == ratings[i - 1])
              {
                i++;
                continue;
            }
            // Initialize increasing slope
              counter
            int peak = 0;
            // Traverse strictly increasing
              ratings
            while (i < n && ratings[i] >
              ratings[i - 1]) {
                peak++;
                candies += peak;
                i++;
            }
            // Initialize decreasing slope
              counter
            int valley = 0;
            // Traverse strictly decreasing
              ratings
            while (i < n && ratings[i] <
              ratings[i - 1]) {
                valley++;
                candies += valley;
                i++;
            }
            // Remove extra candy given to
              peak (overlap of increasing and
              decreasing)
            candies -= min(peak, valley);
        }
        // Return total minimum candies
          required
        return candies;
    }
};
// Driver Code
```

Q227. Shortest Job First

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Example 1:
Input: jobs = [3, 1, 4, 2, 5]
Output: 4
Explanation:
The first job that will be executed is of duration 1 and the waiting time for it will be 0.
After the first job, the next shortest job with a duration of 2 will be executed with a waiting time of 1.
Following the completion of the first two jobs, the next shortest job with a duration of 3 will be executed with a waiting time of 3 (1 + 2).
Then, the job with a duration of 4 will be executed with a waiting time of 6 (1 + 2... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N logN + N) where N is the length of the jobs array. We sort the jobs array giving complexi

```cpp
#include <vector>
using namespace std;

// Class that implements Shortest Job First (SJF) scheduling
class ShortestJobFirst {
public:
    // Function to calculate average waiting time
    float calculateAverageWaitTime(vector<int>& jobs) {
        // Sort jobs in ascending order (Shortest Job First)
        sort(jobs.begin(), jobs.end());

        float waitTime = 0;  // Stores cumulative waiting time
        int totalTime = 0;   // Tracks elapsed execution time
        int n = jobs.size(); // Number of jobs

        // Iterate through each job to calculate waiting time
        for (int i = 0; i < n; i++) {
            waitTime += totalTime;  // Add current total time to waiting time
            totalTime += jobs[i];   // Execute current job
        }

        // Return the average waiting time
        return waitTime / n;
    }
};
```

Q228. Program for Least Recently Used (LRU) Page Replacement Algorithm

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Input: ["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]

Output: [null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation:
LRUCache lRUCache = new LRUCache(2);
lRUCa... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: get() function: O(1), accessing a value in the HashMap is O(1) on ave

```cpp
#include <bits/stdc++.h>
using namespace std;
class LRUCache {
public:
    class Node {
    public:
        int key;
        int val;
        Node* next;
        Node* prev;
        Node(int _key, int _val) {
            key = _key;
            val = _val;
        }
    };
    Node* head = new Node(-1, -1);
    Node* tail = new Node(-1, -1);
    int cap;
    unordered_map<int, Node*> m;
    LRUCache(int capacity) {
        cap = capacity;
        head->next = tail;
        tail->prev = head;
    }
    void addNode(Node* newNode) {
        Node* temp = head->next;
        newNode->next = temp;
        newNode->prev = head;
        head->next = newNode;
        temp->prev = newNode;
    }
    void deleteNode(Node* delNode) {
        Node* delPrev = delNode->prev;
        Node* delNext = delNode->next;
        delPrev->next = delNext;
        delNext->prev = delPrev;
    }
    int get(int key_) {
        if (m.find(key_) != m.end()) {
            Node* resNode = m[key_];
            int res = resNode->val;
            m.erase(key_);
            deleteNode(resNode);
            addNode(resNode);
            m[key_] = head->next;
            return res;
        }
        return -1;
    }
    void put(int key_, int value) {
        if (m.find(key_) != m.end()) {
            Node* existingNode = m[key_];
            m.erase(key_);
            deleteNode(existingNode);
        }
        if (m.size() == cap) {
            m.erase(tail->prev->key);
            deleteNode(tail->prev);
        }
        addNode(new Node(key_, value));
        m[key_] = head->next;
    }
};
```

Q229. Insert Interval

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Refer to the code implementation and logic below.

[OPTIMAL SOLUTION] Snippet 1 — N/A (Conceptual Tutorial)

```
1
// Traverse the array once and maintain a running total by adding each
  element,
2
// ensuring an efficient and straightforward way to compute the final sum.
3

4
public class Main {
5
    public static void main(String[] args) {
6
        int[] arr = {1, 2, 3, 4, 5};
7

8
        int sum = 0;
9
        for (int num : arr) {
10
            sum += num;
11
        }
12

13
        System.out.println(sum); // Output: 15
14
    }
15
}
```

Q230. Merge Intervals

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Input : intervals=[[1,3],[2,6],[8,10],[15,18]]
Output : [[1,6],[8,10],[15,18]]
Explanation : Since intervals [1,3] and [2,6] are overlapping we can merge them to form [1,6] intervals.

Input : [[1,4],[4,5]]
Output : [[1,5]]
Explanation : Since intervals [1,4] and [4,5] are overlapping we can merge them to form [1,5].

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N*logN) + O(N), we sort the entire array and then merge them in a single pass.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to merge overlapping intervals
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        // Sort intervals based on starting time
        sort(intervals.begin(), intervals.end());

        // Vector to store final merged intervals
        vector<vector<int>> merged;

        // Traverse each interval
        for (auto interval : intervals) {
            // If merged is empty or current interval does not overlap
            if (merged.empty() || merged.back()[1] < interval[0]) {
                // Add current interval as a new non-overlapping block
                merged.push_back(interval);
            } else {
                // Overlapping: merge by extending the end time
                merged.back()[1] = max(
                    merged.back()[1],
                    interval[1]
                );
            }
        }

        return merged;
    }
};
```

Q231. Non-overlapping Intervals

MEDIUM

Topic: Greedy Algorithms [Easy, Medium/Hard] > Medium/Hard

Practice Link

Input: Intervals = [[1, 2], [2, 3], [3, 4], [1, 3]]
Output: 1
Explanation: You can remove the interval [1, 3] to make the remaining intervals non-overlapping.
Input: Intervals = [[1, 3], [1, 4], [3, 5], [3, 4], [4, 5]]
Output: 2
Explanation: You can remove the intervals [1, 4] and [3, 5] to make the rest non-overlapping.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n log n) , This is because we sort the intervals by their end time, which takes O(n log n)

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the minimum number of intervals to remove to make all
      intervals non-overlapping
    int eraseOverlapIntervals(vector<vector<int>>& intervals) {
        // Sort intervals based on their end time (greedy strategy)
        sort(intervals.begin(), intervals.end(), [](auto& a, auto& b) {
            return a[1] < b[1];
        });

        // Count of intervals to remove
        int count = 0;

        // Store end time of the last non-overlapping interval
        int prevEnd = intervals[0][1];

        // Iterate through intervals starting from the second
        for (int i = 1; i < intervals.size(); i++) {

            // If current interval starts before the last accepted interval
              ends
            if (intervals[i][0] < prevEnd) {
                // Overlapping interval, increase removal count
                count++;
            } else {
                // No overlap, update the end of last accepted interval
                prevEnd = intervals[i][1];
            }
        }

        // Return minimum intervals to remove
        return count;
    }
};
// Driver Code
```

Q232. Binary Tree Representation in Java

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Refer to the code implementation and logic below.

[OPTIMAL SOLUTION] Snippet 1 — N/A (Conceptual Tutorial)

```
import java.util.*;

// Class representing a Node in the Binary
  Tree
class Node {
    int data;       // Stores the value of the
      node
    Node left;      // Reference to the left
      child node
    Node right;     // Reference to the right
      child node

    // Constructor to initialize a node with a
      given key
    public Node(int key) {
        data = key;
        left = null;
        right = null;
    }
}

// Class containing the logic for Binary Tree
  creation
class Solution {
    public Node createBinaryTree() {
        // Creates the root node with key
          value 1
        Node root = new Node(1);

        // Creates a left child node for the
          root with key value 2
        root.left = new Node(2);

        // Creates a right child node for the
          root with key value 3
        root.right = new Node(3);

        // Creates a left child node for the
          right child of root with key value 5
        root.right.left = new Node(5);

        return root;
    }
}

// Driver class
public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();
        Node root =
          solution.createBinaryTree();

        // Tree created, can be used for
          further operations
    }
}
```

Q233. Pre, Post, Inorder in one traversal

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Input: Binary Tree: 4 2 5 3 -1 7 6 -1 9 -1 -1 8 -1 1

Output: Preorder: [4, 2, 3, 9, 1, 5, 7, 6, 8], Inorder: [3, 1, 9, 2, 4, 7, 5, 8, 6], Postorder: [1, 9, 3, 2, 7, 8, 6, 5, 4]
Explanation:

Input: Binary Tree: 1 2 3 4 5 6 7 -1 -1 8 -1 -1 -1 9 10

Output: Preorder: [1, 2, 4, 5, 8, 3, 6, 7, 9, 10], Inorder: [4, 2, 8, 5, 1, 6, 3, 9, 7, 10], Postorder: [4, 8, 5, 2, 6, 9, 10, 7, 3, 1]
Explanation: The three traversal of the tree are given above.

[OPTIMAL SOLUTION] Approach

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    vector<vector<int>>
      preInPostTraversal(Node* root) {
        vector<int> pre, in, post;
        if (root == NULL) {
            return {};
        }
        stack<pair<Node*, int>> st;
        st.push({root, 1});
        while (!st.empty()) {
            auto it = st.top();
            st.pop();
            if (it.second == 1) {
                pre.push_back(it.first->data);
                it.second = 2;
                st.push(it);
                if (it.first->left != NULL) {
                    st.push({it.first->left,
                      1});
                }
            }
            else if (it.second == 2) {
                in.push_back(it.first->data);
                it.second = 3;
                st.push(it);
                if (it.first->right != NULL) {
                    st.push({it.first->right,
                      1});
                }
            }
            else {
post.push_back(it.first->data);
            }
        }
        vector<vector<int>> result;
        result.push_back(pre);
        result.push_back(in);
        result.push_back(post);
        return result;
    }
};
```

Q234. Preorder Traversal

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Input: Binary Tree: 4 2 5 3 -1 7 6 -1 9 -1 -1 8 -1 1

Output: Preorder: [4, 2, 3, 9, 1, 5, 7, 6, 8]
Explanation: We traverse the binary tree in the order of Root, Left then Right recursively resulting in the following traversal:

Input: Binary Tree: 1 2 3 4 5 6 7 -1 -1 8 -1 -1 -1 9 10

Output: Preorder: [1, 2, 4, 5, 8, 3, 6, 7, 9, 10]
Explanation: We traverse the binary tree in the order of Root, Left then Right recursively resulting in the following traversal:

[OPTIMAL SOLUTION] Approach

```cpp
#include <bits/stdc++.h>
using namespace std;
// Node structure for
// the binary tree
struct Node {
    int data;
    Node* left;
    Node* right;
    // Constructor to initialize
    // the node with a value
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
// Solution class containing traversal
  functions
class Solution {
public:
    // Function to perform preorder traversal
    // of the tree and store values in 'arr'
    void preorder(Node* root, vector<int>
      &arr){
        // If the current node is NULL
        // (base case for recursion), return
        if(root == nullptr){
            return;
        }
        // Push the current node's
        // value into the vector
        arr.push_back(root->data);
        // Recursively traverse
        // the left subtree
        preorder(root->left, arr);
        // Recursively traverse
        // the right subtree
        preorder(root->right, arr);
    }
    // Function to initiate preorder traversal
    // and return the resulting vector
    vector<int> preOrder(Node* root){
        // Create an empty vector to
        // store preorder traversal values
        vector<int> arr;
        // Call the preorder traversal
          function
        preorder(root, arr);
        // Return the resulting vector
        // containing preorder traversal
          values
        return arr;
    }
};
// Main function
```

Q235. Inorder Traversal of Binary Tree

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Example 1:
Input: root = [1, 4, null, 4, 2]
Output: [4, 4, 2, 1]
Explanation: In this example, the preorder traversal of the binary tree produces the sequence [4, 4, 2, 1].

Example 2:
Input: root = [1, null, 2, 3]
Output: [1, 3, 2]
Explanation: Th... (cont.)

[BETTER] Iterateive Approach

Complexity: *Time Complexity: O(n), where n is the number of nodes in the binary tree. Each node i*

```cpp
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : data(x), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
vector<int> inorder(TreeNode* root){
    stack<TreeNode*> st;
    TreeNode* node = root;
    vector<int> inorder;
    while(true){
        if(node != NULL){
            st.push(node);
            node = node->left;
        }
        else{
            if(st.empty()){
                break;
            }
            node = st.top();
            st.pop();
            inorder.push_back(node->data);
            node = node->right;
        }
    }
    return inorder;
}
};
```

[OPTIMAL] Recursive Approach

Complexity: *Time Complexity: O(N), where N is the number of nodes in the binary tree. Each node i*

```cpp
#include <bits/stdc++.h>
using namespace std;
// TreeNode structure for the binary tree
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    // Constructor to initialize
    // the TreeNode with a value
    TreeNode(int val) : data(val),
      left(nullptr), right(nullptr) {}
};
class Solution{
    private:
    // Function to perform inorder traversal
    // of the tree and store values in 'arr'
    void recursiveInorder(TreeNode* root,
      vector<int> &arr){
        // If the current Tree is NULL
        // (base case for recursion), return
        if(root == nullptr){
            return;
        }
        // Recursively traverse the left
          subtree
        recursiveInorder(root->left, arr);
        // Push the current TreeNode's
        // value into the vector
        arr.push_back(root->data);
        // Recursively traverse
        // the right subtree
        recursiveInorder(root->right, arr);
    }
    public:
    // Function to initiate inorder traversal
    // and return the resulting vector
    vector<int> inorder(TreeNode* root){
        // Create an empty vector to
        // store inorder traversal values
        vector<int> arr;
        // Call the inorder traversal function
        recursiveInorder(root, arr);
        // Return the resulting vector
        // containing inorder traversal values
        return arr;
    }
};
// Main function
```

Q236. Postorder Traversal

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Example 1:
Input: Binary Tree: 4 2 5 3 -1 7 6 -1 9 -1 -1 8 -1 1
Output: [1, 9, 3, 2, 7, 8, 6, 5, 4]
Explanation: We traverse the binary tree in the order of Left, Right, and then Root recursively. This results in the following traversal:
1 → 9 → 3 → 2 → 7 → 8 → 6 → 5 → 4.

Example 2:
Input: Binary Tree: 1 2 3 4 5 6 7 -1 -1 8 -1 -1 -1 9 10
Output: [1, 2, 4, 5, 8, 3, 6, 7, 9, 10]
Explanation: We traverse the binary tree in the order of Left, Right, and then Root recursively. This res... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the number of nodes in the binary tree. Each node is visited once during the

```cpp
#include <bits/stdc++.h>
using namespace std;
// Node structure for the binary tree
struct Node {
    int data;   // Value of the node
    Node* left; // Pointer to the left child
    Node* right; // Pointer to the right child
    // Constructor to initialize the node with
      a value
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
// Function to return the postOrder traversal
  of a binary tree using two stacks
vector<int> postOrder(Node* root) {
    vector<int> postorder; // Vector to store
      the postorder traversal
    // If the tree is empty, return an empty
      traversal
    if(root == NULL){
        return postorder;
    }
    // Two stacks for iterative traversal
    stack<Node*> st1, st2;
    // Push the root node onto the first stack
    st1.push(root);
    // Iterative traversal to populate st2
      with nodes in postorder
    while(!st1.empty()){
        // Get the top node from st1
        root = st1.top();
        st1.pop();
        // Push the node onto st2
        st2.push(root);
        // Push left child onto st1 if exists
        if(root->left != NULL){
            st1.push(root->left);
        }
        // Push right child onto st1 if exists
        if(root->right != NULL){
            st1.push(root->right);
        }
    }
    // Populate the postorder traversal vector
      by popping st2
    while(!st2.empty()){
        postorder.push_back(st2.top()->data);
        st2.pop();
    }
    // Return the postorder traversal
    return postorder;
}
// Function to print the elements of a vector
void printVector(const vector<int>& vec) {
    // Iterate through the vector and print
      each element
    for (int num : vec) {
        cout << num << " ";
    }
    cout << endl;
}
// Main function
```

Q237. Level Order Traversal

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Example 1:
Input: root = [3, 9, 20, null, null, 15, 7]
Output:
[[3], [9, 20], [15, 7]]

Example 2:
Input: root = [1, 4, null, 4, 2]
Output: [[1], [4], [4, 2]]

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the number of nodes in the binary tree. Each node is visited once during the

```cpp
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int val) : data(val),
      left(nullptr), right(nullptr) {}
};
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode*
      root) {
        vector<vector<int>> ans;
        if (root == nullptr) {
            return ans;
        }
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int size = q.size();
            vector<int> level;
            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();
                level.push_back(node->data);
                if (node->left != nullptr) {
                    q.push(node->left);
                }
                if (node->right != nullptr) {
                    q.push(node->right);
                }
            }
            ans.push_back(level);
        }
        return ans;
    }
};
void printVector(const vector<int>& vec) {
    for (int num : vec) {
        cout << num << " ";
    }
    cout << endl;
}
```

Q238. Iterative Preorder Traversal of Binary Tree

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Example 1:
Input: Binary Tree: 4 2 5 3 -1 7 6 -1 9 -1 -1 8 -1 1
Output: [4, 2, 3, 9, 1, 5, 7, 6, 8]
Explanation: We traverse the binary tree in the order of Root, Left, and then Right recursively. This results in the following traversal:
4 → 2 → 3 → 9 → 1 → 5 → 7 → 6 → 8.

Example 2:
Input: Binary Tree: 1 2 3 4 5 6 7 -1 -1 8 -1 -1 -1 9 10
Output: [1, 2, 4, 5, 8, 3, 6, 7, 9, 10]
Explanation: We traverse the binary tree in the order of Root, Left, and then Right recursively. This res... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the number of nodes in the binary tree. Each node is visited once during the

```cpp
#include <bits/stdc++.h>
using namespace std;

// Define the TreeNode structure
struct TreeNode {
    int val;  // Value of the node
    TreeNode* left;  // Pointer to the left
      child
    TreeNode* right;  // Pointer to the right
      child
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}  // Constructor to
      initialize a node
};

class Solution {
public:
    // Function to perform preorder traversal
      of a binary tree iteratively
    vector<int> preorderTraversal(TreeNode*
      root) {
        vector<int> preorder;  // Vector to
          store the preorder traversal result

        // If the root is null, return an
          empty traversal result
        if(root == nullptr) {
            return preorder;
        }

        stack<TreeNode*> st;  // Stack to
          store nodes during traversal
        st.push(root);  // Push the root node
          onto the stack

        // Perform iterative preorder
          traversal
        while(!st.empty()) {
            root = st.top();  // Get the
              current node from the top of the
              stack
            st.pop();  // Remove the node from
              the stack

            preorder.push_back(root->val);  //
              Add the node's value to the
              preorder result

            // Push the right child onto the
              stack if exists
            if(root->right != nullptr) {
                st.push(root->right);
            }

            // Push the left child onto the
              stack if exists
            if(root->left != nullptr) {
                st.push(root->left);
            }
        }

        // Return the preorder traversal
          result
        return preorder;
    }
};
```

Q239. Iterative Inorder Traversal of Binary Tree

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Example 1:
Input: root = [1, 4, null, 4, 2]
Output: [4, 4, 2, 1]
Explanation: In this example, the preorder traversal of the binary tree produces the sequence [4, 4, 2, 1].

Example 2:
Input: root = [1, null, 2, 3]
Output: [1, 3, 2]
Explanation: Th... (cont.)

[BETTER] Iterateive Approach

Complexity: *Time Complexity: O(n), where n is the number of nodes in the binary tree. Each node i*

```cpp
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : data(x), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
vector<int> inorder(TreeNode* root){
    stack<TreeNode*> st;
    TreeNode* node = root;
    vector<int> inorder;
    while(true){
        if(node != NULL){
            st.push(node);
            node = node->left;
        }
        else{
            if(st.empty()){
                break;
            }
            node = st.top();
            st.pop();
            inorder.push_back(node->data);
            node = node->right;
        }
    }
    return inorder;
}
};
```

[OPTIMAL] Recursive Approach

Complexity: *Time Complexity: O(N), where N is the number of nodes in the binary tree. Each node i*

```cpp
#include <bits/stdc++.h>
using namespace std;
// TreeNode structure for the binary tree
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    // Constructor to initialize
    // the TreeNode with a value
    TreeNode(int val) : data(val),
      left(nullptr), right(nullptr) {}
};
class Solution{
    private:
    // Function to perform inorder traversal
    // of the tree and store values in 'arr'
    void recursiveInorder(TreeNode* root,
      vector<int> &arr){
        // If the current Tree is NULL
        // (base case for recursion), return
        if(root == nullptr){
            return;
        }
        // Recursively traverse the left
          subtree
        recursiveInorder(root->left, arr);
        // Push the current TreeNode's
        // value into the vector
        arr.push_back(root->data);
        // Recursively traverse
        // the right subtree
        recursiveInorder(root->right, arr);
    }
    public:
    // Function to initiate inorder traversal
    // and return the resulting vector
    vector<int> inorder(TreeNode* root){
        // Create an empty vector to
        // store inorder traversal values
        vector<int> arr;
        // Call the inorder traversal function
        recursiveInorder(root, arr);
        // Return the resulting vector
        // containing inorder traversal values
        return arr;
    }
};
// Main function
```

Q240. Post-order Traversal of Binary Tree using 2 stack

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Example 1:
Input: Binary Tree: 4 2 5 3 -1 7 6 -1 9 -1 -1 8 -1 1
Output: [1, 9, 3, 2, 7, 8, 6, 5, 4]
Explanation: We traverse the binary tree in the order of Left, Right, and then Root recursively. This results in the following traversal:
1 → 9 → 3 → 2 → 7 → 8 → 6 → 5 → 4.

Example 2:
Input: Binary Tree: 1 2 3 4 5 6 7 -1 -1 8 -1 -1 -1 9 10
Output: [1, 2, 4, 5, 8, 3, 6, 7, 9, 10]
Explanation: We traverse the binary tree in the order of Left, Right, and then Root recursively. This res... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the number of nodes in the binary tree. Each node is visited once during the

```cpp
#include <bits/stdc++.h>
using namespace std;
// Node structure for the binary tree
struct Node {
    int data;   // Value of the node
    Node* left; // Pointer to the left child
    Node* right; // Pointer to the right child
    // Constructor to initialize the node with
      a value
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
// Function to return the postOrder traversal
  of a binary tree using two stacks
vector<int> postOrder(Node* root) {
    vector<int> postorder; // Vector to store
      the postorder traversal
    // If the tree is empty, return an empty
      traversal
    if(root == NULL){
        return postorder;
    }
    // Two stacks for iterative traversal
    stack<Node*> st1, st2;
    // Push the root node onto the first stack
    st1.push(root);
    // Iterative traversal to populate st2
      with nodes in postorder
    while(!st1.empty()){
        // Get the top node from st1
        root = st1.top();
        st1.pop();
        // Push the node onto st2
        st2.push(root);
        // Push left child onto st1 if exists
        if(root->left != NULL){
            st1.push(root->left);
        }
        // Push right child onto st1 if exists
        if(root->right != NULL){
            st1.push(root->right);
        }
    }
    // Populate the postorder traversal vector
      by popping st2
    while(!st2.empty()){
        postorder.push_back(st2.top()->data);
        st2.pop();
    }
    // Return the postorder traversal
    return postorder;
}
// Function to print the elements of a vector
void printVector(const vector<int>& vec) {
    // Iterate through the vector and print
      each element
    for (int num : vec) {
        cout << num << " ";
    }
    cout << endl;
}
// Main function
```

Q241. Post-order Traversal of Binary Tree using 1 stack

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Input: Binary Tree: 4 2 5 3 -1 7 6 -1 9 -1 -1 8 -1 1

Output: [1, 9, 3, 2, 7, 8, 6, 5, 4]
Explanation: We traverse the binary tree in the order of Left, Right then Root recursively resulting in the following traversal:

Input: Binary Tree: 1 2 3 4 5 6 7 -1 -1 8 -1 -1 -1 9 10

Output: [1 2 4 5 8 3 6 7 9 10]
Explanation: We traverse the binary tree in the order of Left, Right then Root recursively resulting in the following traversal:

[OPTIMAL SOLUTION] Approach

```cpp
#include <bits/stdc++.h>
using namespace std;

// Definition for a binary tree node
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

// Solution class with postorder function
class Solution {
public:
    void postorderTraversal(TreeNode* root, vector<int>& result) {
        // Base case: if node is null
        if (!root) return;

        // Traverse left subtree
        postorderTraversal(root->left, result);

        // Traverse right subtree
        postorderTraversal(root->right, result);

        // Add current node value
        result.push_back(root->val);
    }
};

// Main function
```

Q242. Preorder, Inorder, and Postorder Traversal in one Traversal

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Traversals

Practice Link

Input: Binary Tree: 4 2 5 3 -1 7 6 -1 9 -1 -1 8 -1 1

Output: Preorder: [4, 2, 3, 9, 1, 5, 7, 6, 8], Inorder: [3, 1, 9, 2, 4, 7, 5, 8, 6], Postorder: [1, 9, 3, 2, 7, 8, 6, 5, 4]
Explanation:

Input: Binary Tree: 1 2 3 4 5 6 7 -1 -1 8 -1 -1 -1 9 10

Output: Preorder: [1, 2, 4, 5, 8, 3, 6, 7, 9, 10], Inorder: [4, 2, 8, 5, 1, 6, 3, 9, 7, 10], Postorder: [4, 8, 5, 2, 6, 9, 10, 7, 3, 1]
Explanation: The three traversal of the tree are given above.

[OPTIMAL SOLUTION] Approach

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    vector<vector<int>>
      preInPostTraversal(Node* root) {
        vector<int> pre, in, post;
        if (root == NULL) {
            return {};
        }
        stack<pair<Node*, int>> st;
        st.push({root, 1});
        while (!st.empty()) {
            auto it = st.top();
            st.pop();
            if (it.second == 1) {
                pre.push_back(it.first->data);
                it.second = 2;
                st.push(it);
                if (it.first->left != NULL) {
                    st.push({it.first->left,
                      1});
                }
            }
            else if (it.second == 2) {
                in.push_back(it.first->data);
                it.second = 3;
                st.push(it);
                if (it.first->right != NULL) {
                    st.push({it.first->right,
                      1});
                }
            }
            else {
post.push_back(it.first->data);
            }
        }
        vector<vector<int>> result;
        result.push_back(pre);
        result.push_back(in);
        result.push_back(post);
        return result;
    }
};
```

Q243. Maximum Depth in BT

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Input: Binary Tree: 1 2 5 -1 -1 4 6 5

Output: 4
Explanation:
In the above example, the height of the binary tree is along the longest path from the root node 1 -> 5 -> 4 -> 5.

Input: Binary Tree: 3 1 2
Output: 2
Explanation:
We traverse the binary tree in the order of Left, RIght then Root recursively resulting in the following traversal:

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), each node is processed once in Level Order Traversal. Space Complexity: O(N), in wo

```cpp
#include <bits/stdc++.h>
using namespace std;

// Node structure for the binary tree
struct Node {
    int data;
    Node* left;
    Node* right;
    // Constructor to initialize
    // the node with a value
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};

class Solution{
public:
    // Function to find the
    // maximum depth of a binary tree
    int maxDepth(Node* root){
        // If the root is NULL
        // (empty tree), depth is 0
        if(root == NULL){
            return 0;
        }

        // Recursive call to find the
        // maximum depth of the left subtree
        int lh = maxDepth(root->left);

        // Recursive call to find the
        // maximum depth of the right subtree
        int rh = maxDepth(root->right);

        // Return the maximum depth of the
        // tree, adding 1 for the current node
        return 1 + max(lh, rh);
    }
};

// Main function
```

Q244. Check for balanced binary tree

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Example 1:
Input : [3, 9, 20, null, null, 15, 7]

Output : Yes
Explanation:

Example 2:
Input : [1, 2, null, null, 3]

Output : No
Explanation:

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the number of nodes in the Binary Tree. Each node is

```cpp
#include <bits/stdc++.h>
using namespace std;
// Node structure for the binary tree
struct Node {
    int data;       // Data stored in the node
    Node* left;     // Pointer to the left
      child
    Node* right;    // Pointer to the right
      child
    // Constructor to initialize the node with
      a value
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    // Function to check if a binary tree is
      balanced
    bool isBalanced(Node* root) {
        // Check if the tree's height
          difference between subtrees is less
          than 2
        // If not, return false; otherwise,
          return true
        return dfsHeight(root) != -1;
    }
    // Recursive function to calculate the
      height of the tree
    int dfsHeight(Node* root) {
        // Base case: if the current node is
          NULL, return 0 (height of an empty
          tree)
        if (root == NULL) return 0;
        // Recursively calculate the height of
          the left subtree
        int leftHeight =
          dfsHeight(root->left);
        // If the left subtree is unbalanced,
          propagate the unbalance status
        if (leftHeight == -1)
            return -1;
        // Recursively calculate the height of
          the right subtree
        int rightHeight =
          dfsHeight(root->right);
        // If the right subtree is unbalanced,
          propagate the unbalance status
        if (rightHeight == -1)
            return -1;
        // Check if the difference in height
          between left and right subtrees is
          greater than 1
        // If it's greater, the tree is
          unbalanced, return -1 to propagate
          the unbalance status
        if (abs(leftHeight - rightHeight) > 1)
            return -1;
        // Return the maximum height of left
          and right subtrees, adding 1 for the
          current node
        return max(leftHeight, rightHeight) +
          1;
    }
};
// Main function
```

Q245. Diameter of Binary Tree

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Example 1:
Input:

Output: 4
Explanation: The distance between the leftmost node 4 and the rightmost node 3 is 4, since this is the longest horizontal distance of the binary tree, hence its diameter.

Example 2:
Input:

Output : 7
Explanation: The distance between the leftmost node 4 and the rightmost node 3 is 4, since this is the longest horizontal distance of the binary tree, hence its diameter.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N) where N is the number of nodes in the Binary Tree. This complexity arises from visiting

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
// Node structure for the binary tree
struct Node {
    int data;
    Node* left;
    Node* right;
    // Constructor to initialize
    // the node with a value
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    // Function to find the
    // diameter of a binary tree
    int diameterOfBinaryTree(Node* root) {
        // Initialize the variable to
        // store the diameter of the tree
        int diameter = 0;
        // Call the height function to
          traverse
        // the tree and calculate diameter
        height(root, diameter);
        // Return the calculated diameter
        return diameter;
    }
private:
    // Function to calculate the height of
    // the tree and update the diameter
    int height(Node* node, int& diameter) {
        // Base case: If the node is null,
        // return 0 indicating the
        // height of an empty tree
        if (!node) {
            return 0;
        }
        // Recursively calculate the
        // height of left and right subtrees
        int lh = height(node->left, diameter);
        int rh = height(node->right,
          diameter);
        // Update the diameter with the
          maximum
        // of current diameter or sum of
        // left and right heights
        diameter = max(diameter, lh + rh);
        // Return the height of
        // the current node's subtree
        return 1 + max(lh, rh);
    }
};
// Main function
```

Q246. Maximum path sum

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Input: Binary Tree: -10 9 20 -1 -1 15 7

Output: 42
Explanation:
Out of all the paths possible in the Binary Tree, 15 -> 20 -> 7 has the greatest sum ie. 42.

Input: Binary Tree: -2 2 1

Output: 2
Explanation:
Out of all the paths possible in the Binary Tree, a path starting and ending at the node with value 2 has the greatest sum ie. 2.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), each node is processed once in DFS Traversal. Space Complexity: O(H), auxiliary sta

```cpp
#include <bits/stdc++.h>
using namespace std;

// Tree Node Definition
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int x) {
        val = x;
        left = NULL;
        right = NULL;
    }
};

class Solution {
public:
    // Function to find max path sum in the
      tree
    int maxPathSum(TreeNode* root) {
        int maxSum = INT_MIN;
        dfs(root, maxSum);
        return maxSum;
    }

    // Helper function for recursive DFS
    int dfs(TreeNode* node, int &maxSum) {
        if (!node) return 0;

        // Recursively find left and right max
          path sums
        int left = max(0, dfs(node->left,
          maxSum));
        int right = max(0, dfs(node->right,
          maxSum));

        // Update global max sum considering
          node as turning point
        maxSum = max(maxSum, left + right +
          node->val);

        // Return the best one-sided path sum
        return max(left, right) + node->val;
    }
};

// Driver Code
```

Q247. Check if two trees are identical or not

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Example 1:
Input: Binary Tree 1: 1 2 3 -1 -1 4 5, Binary Tree 2: 1 2 3 -1 -1 4 5

Output: True, these trees are identical.

Example 2:
Input: Binary Tree 1: 1 2 3 -1 -1 4 5, Binary Tree 2: 1 2 3 -1 -1 4

Output : False, these trees are not identical.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N + M) where N is the number of nodes in the first Binary Tree and M is

```cpp
#include <bits/stdc++.h>
using namespace std;

// Node structure for the binary tree
struct Node {
    int data;       // Data stored in the node
    Node* left;     // Pointer to the left child
    Node* right;    // Pointer to the right child

    // Constructor to initialize the node with a value
    Node(int val) : data(val), left(nullptr), right(nullptr) {}
};

class Solution {
public:
    // Function to check if two binary trees are identical
    bool isIdentical(Node* node1, Node* node2) {
        // Case 1: If both nodes are NULL, they are identical
        if (node1 == NULL && node2 == NULL) {
            return true;
        }

        // Case 2: If only one of the nodes is NULL, they are not identical
        if (node1 == NULL || node2 == NULL) {
            return false;
        }

        // Check if the current nodes have the same data value
        // and recursively check their left and right subtrees
        return ((node1->data == node2->data)
                && isIdentical(node1->left, node2->left)
                && isIdentical(node1->right, node2->right));
    }
};
```

Q248. Zig Zag or Spiral Traversal

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Input: Binary Tree: 1 2 3 4 5 -1 6

Output: [[1],[3, 2],[4, 5, 6]]
Explanation: Level 1 (Root): Visit the root node 1 from left to right. Zigzag Traversal: [1]Level 2: Visit nodes at this level in a right-to-left order. Zigzag Traversal: [1], [3, 2]Level 3: Visit nodes at this level in a left-to-right order. Zigzag Traversal: [1], [3, 2], [4, 5, 6]

Input: Binary Tree: 1 2 -1 4 5 -1 -1 7 8

Output : [[1], [2], [4, 5], [8, 7]]
Explanation: Level 1 (Root): Visit the root node 1 from left to right.... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N) where N is the number of nodes in the binary tree. Each node of the binary tree is enque

```cpp
#include <bits/stdc++.h>
using namespace std;
// Structure to represent a binary tree node
struct TreeNode {
    int val;                // Value of the
      node
    TreeNode* left;         // Pointer to left
      child
    TreeNode* right;        // Pointer to
      right child
    // Constructor to initialize node with a
      given value
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    // Function to perform zigzag (spiral)
      level order traversal of a binary tree
    vector<vector<int>>
      zigzagLevelOrder(TreeNode* root) {
        // Vector to store the final zigzag
          traversal result
        vector<vector<int>> result;
        // If the tree is empty, return an
          empty result
        if (!root) return result;
        // Queue to store nodes level by level
          (BFS)
        queue<TreeNode*> q;
        q.push(root);
        // Boolean flag to control traversal
          direction: true for left-to-right,
          false for right-to-left
        bool leftToRight = true;
        // Loop until all levels are processed
        while (!q.empty()) {
            // Get the number of nodes at the
              current level
            int size = q.size();
            // Temporary vector to store the
              current level's values in
              correct zigzag order
            vector<int> level(size);
            // Process each node in the
              current level
            for (int i = 0; i < size; i++) {
                // Get the front node from the
                  queue
                TreeNode* node = q.front();
                q.pop();
                // Determine index where this
                  node's value should be
                  placed based on traversal
                  direction
                int index = leftToRight ? i :
                  size - 1 - i;
                level[index] = node->val;
                // If left child exists, add
                  it to the queue
                if (node->left)
                  q.push(node->left);
                // If right child exists, add
                  it to the queue
                if (node->right)
                  q.push(node->right);
            }
            // After finishing this level,
              toggle the direction for the
              next level
            leftToRight = !leftToRight;
            // Add this level's data to the
              final result
            result.push_back(level);
        }
        // Return the zigzag traversal result
        return result;
    }
};
```

Q249. Boundary Traversal

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Example 1:
Input: Binary Tree: 1 2 7 3 -1 -1 8 -1 4 9 -1 5 6 10 11

Output: Boundary Traversal: [1, 2, 3, 4, 5, 6, 10, 11, 9, 8, 7]
Explanation: The boundary traversal of a binary tree involves visiting its boundary nodes in an anticlockwise direction:
Starting... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N) where N is the number of nodes in the Binary Tree. Space Complex

```
#include <iostream>
#include <vector>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    bool isLeaf(Node* root) {
        return !root->left && !root->right;
    }
    void addLeftBoundary(Node* root,
      vector<int>& res) {
        Node* curr = root->left;
        while (curr) {
            if (!isLeaf(curr)) {
                res.push_back(curr->data);
            }
            if (curr->left) {
                curr = curr->left;
            } else {
                curr = curr->right;
            }
        }
    }
    void addRightBoundary(Node* root,
      vector<int>& res) {
        Node* curr = root->right;
        vector<int> temp;
        while (curr) {
            if (!isLeaf(curr)) {
                temp.push_back(curr->data);
            }
            if (curr->right) {
                curr = curr->right;
            } else {
                curr = curr->left;
            }
        }
        for (int i = temp.size() - 1; i >= 0;
          --i) {
            res.push_back(temp[i]);
        }
    }
    void addLeaves(Node* root, vector<int>&
      res) {
        if (isLeaf(root)) {
            res.push_back(root->data);
            return;
        }
        if (root->left) {
            addLeaves(root->left, res);
        }
        if (root->right) {
            addLeaves(root->right, res);
        }
    }
    vector<int> printBoundary(Node* root) {
        vector<int> res;
        if (!root) {
            return res;
        }
        if (!isLeaf(root)) {
            res.push_back(root->data);
        }
        addLeftBoundary(root, res);
        addLeaves(root, res);
        addRightBoundary(root, res);
        return res;
    }
};
void printResult(const vector<int>& result) {
    for (int val : result) {
        cout << val << " ";
    }
    cout << endl;
}
```

Q250. Vertical Order Traversal

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Input: Binary Tree: 1 2 3 4 10 9 11 -1 5 -1 -1 -1 -1 -1 -1 -1 6

Output: Vertical Order Traversal: [[4],[2, 5], [1, 10, 9, 6],[3],[11]]
Explanation: Vertical Levels from left to right:Level -2: [4],Level -1: [2],Level 0: [1, 10, 9, 6] (Overlapping nodes are added in their level order sequence),Level 1: [3],Level 2: [11],
Input: Binary Tree: 2 7 5 2 6 -1 9 -1 -1 5 11 4 -1

Output : [[2],[7, 5],[2, 6], [5, 11, 4],[9]]
Explanation: Vertical Levels from left to right:Level -2: [2],Level -1: [7, 5],Lev... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N * log²N * log²N * log²N), where N represents the number of nodes in the Binary Tree.

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    vector<vector<int>> findVertical(Node*
      root) {
        map<int, map<int, multiset<int>>>
          nodes;
        queue<pair<Node*, pair<int, int>>>
          todo;
        todo.push({root, {0, 0}});
        while (!todo.empty()) {
            auto p = todo.front();
            todo.pop();
            Node* temp = p.first;
            int x = p.second.first;
            int y = p.second.second;
            nodes[x][y].insert(temp->data);
            if (temp->left) {
                todo.push({temp->left, {x - 1,
                  y + 1}});
            }
            if (temp->right) {
                todo.push({temp->right, {x +
                  1, y + 1}});
            }
        }
        vector<vector<int>> ans;
        for (auto p : nodes) {
            vector<int> col;
            for (auto q : p.second) {
                col.insert(col.end(),
                  q.second.begin(),
                  q.second.end());
            }
            ans.push_back(col);
        }
        return ans;
    }
};
void printResult(const vector<vector<int>>&
  result) {
    for (auto level : result) {
        for (auto node : level) {
            cout << node << " ";
        }
        cout << endl;
    }
    cout << endl;
}
```

Q251. Top View of BT

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Input: Binary Tree: 1 2 3 4 10 9 11 -1 5 -1 -1 -1 -1 -1 -1 -1 6

Output: Top View: [4, 2, 1, 3, 11]
Explanation:

Input: Binary Tree: 2 7 5 2 6 -1 9 -1 -1 5 11 4 -1

Output : Top View: [2, 7, 2, 5, 9]
Explanation:

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N) where N is the number of nodes in the Binary Tree. This complexity arises from visiting

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    vector<int> topView(Node* root) {
        vector<int> ans;
        if (root == NULL) {
            return ans;
        }
        map<int, int> mpp;
        queue<pair<Node*, int>> q;
        q.push({root, 0});
        while (!q.empty()) {
            auto it = q.front();
            q.pop();
            Node* node = it.first;
            int line = it.second;
            if (mpp.find(line) == mpp.end()) {
                mpp[line] = node->data;
            }
            if (node->left != NULL) {
                q.push({node->left, line -
                  1});
            }
            if (node->right != NULL) {
                q.push({node->right, line +
                  1});
            }
        }
        for (auto it : mpp) {
            ans.push_back(it.second);
        }
        return ans;
    }
};
```

Q252. Bottom view of BT

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Example 1:
Input: Binary Tree: 1 2 3 4 10 9 11 -1 5 -1 -1 -1 -1 -1 -1 -1 6

Output: Bottom View Traversal: [4, 5, 6, 3, 11]
Explanation: The bottom view of the binary tree would comprise of the nodes that are the last encountered nodes for each vertical index.

Example 2:
Input: Binary Tree: 2 7 5 2 6 -1 9 -1 -1 5 11 4 -1

Output : Bottom View: [2 5 6 11 4 9]
Explanation:

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N) where N is the number of nodes in the Binary Tree. This complexity arises from visiting

```cpp
#include <iostream>
#include <vector>
#include <set>
#include <queue>
#include <map>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
class Solution{
public:
    vector<int> bottomView(Node* root){
        vector<int> ans;
        if(root == NULL){
            return ans;
        }
        map<int, int> mpp;
        queue<pair<Node*, int>> q;
        q.push({root, 0});
        while(!q.empty()){
            auto it = q.front();
            q.pop();
            Node* node = it.first;
            int line = it.second;
            mpp[line] = node->data;
            if(node->left != NULL){
                q.push({node->left, line -
                  1});
            }
            if(node->right != NULL){
                q.push({node->right, line +
                  1});
            }
        }
        for(auto it : mpp){
            ans.push_back(it.second);
        }
        return ans;
    }
};
```

Q253. Right/Left View of Binary Tree

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Input : root = [1, 2, 3, null, 5, null, 4]
Output :[1, 3, 4]
Explanation :
Input :root = [1, 2, 3, 6, 5, 8, 4]
Output : [1, 3, 4]
Explanation :

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N) In the worst case, we may visit every node in the binary tree exactly once. This happens

```cpp
#include <bits/stdc++.h>
using namespace std;
// TreeNode definition
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int v) : val(v), left(NULL),
      right(NULL) {}
};
class Solution {
public:
    // Recursive function to get left view
    void leftDFS(TreeNode* node, int level,
      vector<int>& res) {
        // Base case: null node
        if (!node) return;
        // If this is the first node at the
          current level
        if (res.size() == level)
            res.push_back(node->val);
        // Explore left subtree first
        leftDFS(node->left, level + 1, res);
        // Then explore right subtree
        leftDFS(node->right, level + 1, res);
    }
    // Recursive function to get right view
    void rightDFS(TreeNode* node, int level,
      vector<int>& res) {
        if (!node) return;
        if (res.size() == level)
            res.push_back(node->val);
        // Explore right subtree first
        rightDFS(node->right, level + 1, res);
        // Then explore left subtree
        rightDFS(node->left, level + 1, res);
    }
    vector<int> leftView(TreeNode* root) {
        vector<int> res;
        leftDFS(root, 0, res);
        return res;
    }
    vector<int> rightView(TreeNode* root) {
        vector<int> res;
        rightDFS(root, 0, res);
        return res;
    }
};
// Driver code
```

Q254. Symmetric Binary Tree

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Medium Problems

Practice Link

Example 1:
Input: Binary Tree: 1 2 2 3 4 4 3

Output: True
Explanation: If we were to draw a vertical line through the centre of the tree, dividing it into left and right parts, we observe that the nodes on the left and right sides are mirror images of each other.

Example 2:
Input: Binary Tree: 1 2 2 -1 3 -1 3

Output : False
Explanation: While the tree has a symmetric structure at the first level with nodes 2 and 2, the subtrees under nodes 2 and 2 are not symmetric. If we were to draw a ver... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N) where N is the number of nodes in the Binary Tree. This complexity arises from visiting

```cpp
#include <iostream>
using namespace std;
// Node structure for the binary tree
struct Node {
    int data;
    Node* left;
    Node* right;
    // Constructor to initialize
    // the node with a value
    Node(int val) : data(val), left(nullptr),
      right(nullptr) {}
};
class Solution {
private:
    // Function to check if
    // two subtrees are symmetric
    bool isSymmetricUtil(Node* root1, Node*
      root2) {
        // Check if either subtree is NULL
        if (root1 == NULL || root2 == NULL) {
            // If one subtree is NULL, the
              other
            // must also be NULL for symmetry
            return root1 == root2;
        }
        // Check if the data in the current
          nodes is equal
        // and recursively check for symmetry
          in subtrees
        return (root1->data == root2->data)
            && isSymmetricUtil(root1->left,
              root2->right)
            && isSymmetricUtil(root1->right,
              root2->left);
    }
public:
    // Public function to check if the
    // entire binary tree is symmetric
    bool isSymmetric(Node* root) {
        // Check if the tree is empty
        if (!root) {
            // An empty tree is
            // considered symmetric
            return true;
        }
        // Call the utility function
        // to check symmetry of subtrees
        return isSymmetricUtil(root->left,
          root->right);
    }
};
// Function to print the Inorder
// Traversal of the Binary Tree
void printInorder(Node* root){
    if(!root){
        return;
    }
    printInorder(root->left);
    cout << root->data << " ";
    printInorder(root->right);
}
```

Q255. Print root to leaf path in BT

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Input: Binary Tree: 1 2 3 4 5 -1 -1 -1 -1, Node: 7

Output: [1, 2, 5, 7]
Explanation:

Input: Binary Tree: [1, 2, 3, -1, -1, 4, 5, -1, -1, 6], Node: 6

Output: [1, 3, 5, 6]
Explanation:

[OPTIMAL SOLUTION] Approach

```cpp
#include <bits/stdc++.h>
using namespace std;
// Structure for a binary tree node
struct TreeNode {
    // Value of the node
    int val;
    // Pointers to left and right children
    TreeNode* left;
    TreeNode* right;
    // Constructor to initialize the node
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
// Class containing the solution
class Solution {
public:
    // Function to find the path from root to
      node with value x
    bool getPath(TreeNode* root, vector<int>&
      arr, int x) {
        // If current node is NULL, return
          false
        if (!root) {
            return false;
        }
        // Add current node's value to the
          path
        arr.push_back(root->val);
        // If current node's value is equal to
          x, return true
        if (root->val == x) {
            return true;
        }
        // Recursively search in left or right
          subtree
        if (getPath(root->left, arr, x) ||
            getPath(root->right, arr, x)) {
            return true;
        }
        // If not found, backtrack and remove
          current node
        arr.pop_back();
        return false;
    }
    // Function to get the path from root to
      node with value B
    vector<int> solve(TreeNode* A, int B) {
        // Vector to store the path
        vector<int> arr;
        // If root is NULL, return empty path
        if (A == NULL) {
            return arr;
        }
        // Call helper function to fill the
          path
        getPath(A, arr, B);
        // Return the resulting path
        return arr;
    }
};
// Main function to run the code
```

Q256. LCA in BT

HARD

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Example 1:
Input: root = [3, 5, 1, 6, 2, 0, 8, null, null, 7, 4], p = 5, q = 1
Output: 3
Explanation: The lowest common ancestor (LCA) of nodes 5 and 1 is node 3, as it is the first common ancestor of both nodes in the binary tree.

Example 2:
Input: root = [3, 5, 1, 6, 2, 0, 8, null, null, 7, 4], p = 5, q = 4
Output: 5
Explanation: The lowest common ancestor (LCA) of nodes 5 and 4 is node 5, as node 5 is the ancestor of node 4 in the binary tree.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the number of nodes in the binary tree. In the worst case, we may need to tr

```cpp
#include <bits/stdc++.h>
using namespace std;

struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int val) : data(val), left(NULL), right(NULL) {}
};

class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        // Base case
        if (root == NULL || root == p || root == q) {
            return root;
        }

        // Search in left and right subtrees
        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);

        // Result
        if (left == NULL) {
            return right;
        } else if (right == NULL) {
            return left;
        } else { // Both left and right are not null, we found our result
            return root;
        }
    }
};
```

Q257. Maximum Width of BT

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Input: Binary Tree: 1 2 3 5 6 -1 9

Output: Maximum Width: 4
Explanation:
Level 3 is the widest level of the Binary Tree and whose end-to-end width is 4 comprising of nodes: {5, 6, null, 9}.

Input: Binary Tree: 1 2 3 5

Output: Maximum Width: 2
Explanation:
Level 2 is the widest level of the Binary Tree and whose end-to-end width is comprised of nodes: {2, 3}.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), each node is processed once in level order traversal. Space Complexity: O(N), in wo

```cpp
#include <bits/stdc++.h>
using namespace std;
class TreeNode {
public:
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) {
        val = x;
        left = nullptr;
        right = nullptr;
    }
};
class Solution {
public:
    int widthOfBinaryTree(TreeNode* root) {
        if (!root)
            return 0;
        int maxWidth = 0;
        queue<pair<TreeNode*, int>> q;
        q.push({root, 0});
        while (!q.empty()) {
            int size = q.size();
            int minIndex = q.front().second;
            int first = 0;
            int last = 0;
            for (int i = 0; i < size; i++) {
                int currIndex =
                  q.front().second - minIndex;
                TreeNode* node =
                  q.front().first;
                q.pop();
                if (i == 0)
                    first = currIndex;
                if (i == size - 1)
                    last = currIndex;
                if (node->left)
                    q.push({node->left, 2 *
                      currIndex + 1});
                if (node->right)
                    q.push({node->right, 2 *
                      currIndex + 2});
            }
            maxWidth = max(maxWidth, last -
              first + 1);
        }
        return maxWidth;
    }
};
```

Q258. Children Sum Property in Binary Tree

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Example 1:
Input: Binary Tree: 2 35 10 2 3 5 2

Output: Binary Tree: 45 35 10 30 5 8 2

Explanation: We cannot decrement the value of the node but only increment. There are many different ways to do this but we have to ensure that we are only increasing the values of the nodes in such a way that its value is equal to the sum of its left and right children.

Example 2:
Input: Binary Tree: 50 7 2 3 5 1 30

Output : Binary Tree: 50 55 5 86 1 31 30

Explanation: We modify the tree in such a way... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the number of nodes in the binary tree. This is because the algorithm traver

```cpp
#include <iostream>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    void changeTree(TreeNode* root) {
        if (root == NULL) {
            return;
        }
        int child = 0;
        if (root->left) {
            child += root->left->val;
        }
        if (root->right) {
            child += root->right->val;
        }
        if (child >= root->val) {
            root->val = child;
        } else {
            if (root->left) {
                root->left->val = root->val;
            } else if (root->right) {
                root->right->val = root->val;
            }
        }
        changeTree(root->left);
        changeTree(root->right);
        int tot = 0;
        if (root->left) {
            tot += root->left->val;
        }
        if (root->right) {
            tot += root->right->val;
        }
        if (root->left or root->right) {
            root->val = tot;
        }
    }
};
void inorderTraversal(TreeNode* root) {
    if (root == nullptr) {
        return;
    }
    inorderTraversal(root->left);
    cout << root->val << " ";
    inorderTraversal(root->right);
}
```

Q259. Print all nodes at a distance of K in BT

HARD

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Input: root = [3, 5, 1, 6, 2, 0, 8, N, N, 7, 4] , target = 5, k = 2
Output:[7,4,1]
Explanation: The nodes that are a distance 3 from the target node (with value 5) have values 7 , 4 and 1.

Input :root = [3, 5, 1, 6, 2, 0, 8, N, N, 7, 4] , target = 5, k = 3
O... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N) ,We visit each node exactly once when building the parent map using BFS ,O(N). We ag

```
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL),
      right(NULL) {}
};
class Solution {
public:
    vector<int> distanceK(TreeNode* root,
      TreeNode* target, int k) {
        if (!root) return {};
        unordered_map<TreeNode*, TreeNode*>
          parentMap;
        mapParentNodes(root, parentMap);
        return bfsFromTarget(target,
          parentMap, k);
    }
private:
    void mapParentNodes(TreeNode* root,
      unordered_map<TreeNode*, TreeNode*>&
      parentMap) {
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            TreeNode* node = q.front();
            q.pop();
            if (node->left) {
                parentMap[node->left] = node;
                q.push(node->left);
            }
            if (node->right) {
                parentMap[node->right] = node;
                q.push(node->right);
            }
        }
    }
    vector<int> bfsFromTarget(TreeNode*
      target, unordered_map<TreeNode*,
      TreeNode*>& parentMap, int k) {
        queue<TreeNode*> q;
        unordered_set<TreeNode*> visited;
        q.push(target);
        visited.insert(target);
        int currentLevel = 0;
        while (!q.empty()) {
            int size = q.size();
            if (currentLevel++ == k) break;
            for (int i = 0; i < size; ++i) {
                TreeNode* node = q.front();
                q.pop();
                if (node->left &&
                  visited.find(node->left) ==
                  visited.end()) {
visited.insert(node->left);
                    q.push(node->left);
                }
                if (node->right &&
                  visited.find(node->right) ==
                  visited.end()) {
visited.insert(node->right);
                    q.push(node->right);
                }
                if (parentMap.count(node) &&
                  visited.find(parentMap[node]
                  ) == visited.end()) {
visited.insert(parentMap[node]);
                    q.push(parentMap[node]);
                }
            }
        }
        vector<int> result;
        while (!q.empty()) {
            result.push_back(q.front()->val);
            q.pop();
        }
        return result;
    }
};
```

Q260. Minimum time taken to burn the BT from a given Node

HARD

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Input : root = [1, 2, 3, 4, null, 5, 6, null, 7]. target = 1
Output : 3
Explanation :The node with value 1 is set on fire.
In 1st second it burns node 2 and node 3.
In 2nd second it burns nodes 4, 5, 6.
In 3rd second it burns node 7.

Input : root = [1, 2, 3, null, 5, null, 4], target = 4
Output : 4
Explanation : node with value 4 is set on fire.
In 1st second it burns node 3.
In 2nd second it burns node 1.
In 3rd second it burns node 2.
In 4th second it burns node 5.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), We perform two full traversals of the tree. The first is a DFS to build the graph, whic

```
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL),
      right(NULL) {}
};
class Solution {
public:
    int minTime(TreeNode* root, int target) {
        unordered_map<int, vector<int>> graph;
        buildGraph(root, nullptr, graph);
        unordered_set<int> visited;
        queue<int> q;
        q.push(target);
        visited.insert(target);
        int time = 0;
        while (!q.empty()) {
            int size = q.size();
            bool burned = false;
            for (int i = 0; i < size; i++) {
                int node = q.front();
                q.pop();
                for (int neighbor :
                  graph[node]) {
                    if
                      (!visited.count(neighbor
                      )) {
visited.insert(neighbor);
                        q.push(neighbor);
                        burned = true;
                    }
                }
            }
            if (burned) time++;
        }
        return time;
    }
private:
    void buildGraph(TreeNode* node, TreeNode*
      parent, unordered_map<int, vector<int>>&
      graph) {
        if (!node) return;
        if (parent) {
graph[node->val].push_back(parent->val);
graph[parent->val].push_back(node->val);
        }
        buildGraph(node->left, node, graph);
        buildGraph(node->right, node, graph);
    }
};
```

Q261. Count total nodes in a complete BT

EASY

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Input: Binary Tree: 1 2 3 4 5 6

Output: 6
Explanation: There are 6 nodes in this Binary Tree.

Input: Binary Tree: 2 4 3 5 9 8 7 1 6
Output : 9
Explanation: There are 9 nodes in this Binary Tree.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(log N * log N) where N is the number of nodes in the Binary Tree. T

```cpp
#include <bits/stdc++.h>
using namespace std;
// Structure to represent a binary tree node
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    // Constructor to initialize node with a
      value
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    // Function to count the number of nodes
      in a complete binary tree
    int countNodes(TreeNode* root) {
        // If tree is empty, return 0
        if (root == NULL) {
            return 0;
        }
        // Get height from left side
        int lh = findHeightLeft(root);
        // Get height from right side
        int rh = findHeightRight(root);
        // If heights are equal, it's a
          perfect binary tree
        if (lh == rh) {
            // Use formula: 2^h - 1
            return (1 << lh) - 1;
        }
        // Otherwise, recursively count left
          and right subtrees
        return 1 + countNodes(root->left) +
          countNodes(root->right);
    }
    // Helper function to find height of
      leftmost path
    int findHeightLeft(TreeNode* node) {
        int height = 0;
        while (node) {
            height++;
            node = node->left;
        }
        return height;
    }
    // Helper function to find height of
      rightmost path
    int findHeightRight(TreeNode* node) {
        int height = 0;
        while (node) {
            height++;
            node = node->right;
        }
        return height;
    }
};
```

Q262. Construct a BT from Preorder and Inorder

HARD

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Input : preorder = [3, 9, 20, 15, 7] , inorder = [9, 3, 15, 20, 7]
Output : [3, 9, 20, null, null, 15, 7]
Explanation : The output tree is shown below.

Input : preorder = [3, 4, 5, 6, 2, 9] , inorder = [5, 4, 6, 3, 2, 9]
Output : [3, 4, 2, 5, 6, null, 9]
Explanation : The output tree is shown below.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), as ach node is visited once. Space Complexity: O(N), for the hashma

```cpp
#include <bits/stdc++.h>
using namespace std;
// TreeNode structure
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    // Constructor to initialize a node
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    // Function to build the binary tree from
      preorder and inorder
    TreeNode* buildTree(vector<int>& preorder,
      vector<int>& inorder) {
        // Map to store the index of each
          value in inorder
        map<int, int> inMap;
        // Fill the map with inorder values
          and their indices
        for (int i = 0; i < inorder.size();
          i++) {
            inMap[inorder[i]] = i;
        }
        // Call the recursive helper function
        return build(preorder, 0,
          preorder.size() - 1, inorder, 0,
          inorder.size() - 1, inMap);
    }
private:
    // Recursive function to build tree using
      preorder and inorder segments
    TreeNode* build(vector<int>& preorder, int
      preStart, int preEnd,
                    vector<int>& inorder, int
                      inStart, int inEnd,
                      map<int, int>& inMap) {
        // Base condition
        if (preStart > preEnd || inStart >
          inEnd) return nullptr;
        // The first element in preorder is
          root
        TreeNode* root = new
          TreeNode(preorder[preStart]);
        // Find the root index in inorder
        int inRoot = inMap[root->val];
        // Number of elements in left subtree
        int numsLeft = inRoot - inStart;
        // Recursively build left and right
          subtrees
        root->left = build(preorder, preStart
          + 1, preStart + numsLeft,
                           inorder, inStart,
                             inRoot - 1,
                             inMap);
        root->right = build(preorder, preStart
          + numsLeft + 1, preEnd,
                            inorder, inRoot +
                              1, inEnd,
                              inMap);
        return root;
    }
};
// Inorder traversal to print tree
void printInorder(TreeNode* root) {
    if (!root) return;
    printInorder(root->left);
    cout << root->val << " ";
    printInorder(root->right);
}
```

Q263. Construct the Binary Tree from Postorder and Inorder Traversal

HARD

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Example :
Input: Inorder: [40, 20 , 50, 10, 60, 30], Postorder: [40, 50, 20, 60, 30, 10]

Explanation: The unique Binary Tree hence created has inorder traversal: [40, 20, 50, 10, 60, 30] and postorder traversal: [40, 50, 20, 60, 30, 10].

Example :
Input: Inorder: [9, 3, 15, 20, 7], Postorder: [9, 15, 7, 20, 3]

Explanation: The unique Binary Tree hence created has inorder traversal: [9, 3, 15, 20, 7] and postorder traversal: [9, 15, 7, 20, 3].

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), Every node is visited once. Space Complexity: O(N), Due to hashmap

```cpp
#include <bits/stdc++.h>
using namespace std;
// TreeNode structure
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder,
      vector<int>& postorder) {
        if (inorder.size() !=
          postorder.size()) return nullptr;
        // Map each value in inorder to its
          index
        map<int, int> hm;
        for (int i = 0; i < inorder.size();
          i++) {
            hm[inorder[i]] = i;
        }
        // Call recursive builder
        return build(inorder, 0,
          inorder.size() - 1,
                     postorder, 0,
                       postorder.size() - 1,
                       hm);
    }
private:
    TreeNode* build(vector<int>& inorder, int
      is, int ie,
                    vector<int>& postorder,
                      int ps, int pe, map<int,
                      int>& hm) {
        if (ps > pe || is > ie) return
          nullptr;
        // Last element in postorder is root
        TreeNode* root = new
          TreeNode(postorder[pe]);
        // Find root index in inorder
        int inRoot = hm[postorder[pe]];
        int numsLeft = inRoot - is;
        // Recursively build left and right
          subtrees
        root->left = build(inorder, is, inRoot
          - 1,
                           postorder, ps, ps +
                             numsLeft - 1,
                             hm);
        root->right = build(inorder, inRoot +
          1, ie,
                            postorder, ps +
                              numsLeft, pe -
                              1, hm);
        return root;
    }
};
// Inorder printer
void printInorder(TreeNode* root) {
    if (!root) return;
    printInorder(root->left);
    cout << root->val << " ";
    printInorder(root->right);
}
// Driver
```

Q264. Serialize and De-serialize BT

HARD

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Input: Binary Tree: 1 2 3 -1 -1 4 5

Output: After Serialisation: 1,2,3,#,#,4,5,#,#,#,#, After Deserialization: (Original Tree Back)
Explanation: Any algorithm that compresses this binary tree to a string which can be transmitted and from which the binary tre... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N) serialize function: O(N), where N is the number of nodes in the tree

```
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) {
        val = x;
        left = nullptr;
        right = nullptr;
    }
};
void inorder(TreeNode* root) {
    if (!root) {
        return;
    }
    inorder(root->left);
    cout << root->val << " ";
    inorder(root->right);
}
class Solution {
public:
    string serialize(TreeNode* root) {
        if (!root) {
            return "";
        }
        string s = "";
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            TreeNode* curNode = q.front();
            q.pop();
            if (curNode == nullptr) {
                s += "#,";
            }
            else {
                s += to_string(curNode->val) +
                  ",";
                q.push(curNode->left);
                q.push(curNode->right);
            }
        }
        return s;
    }
    TreeNode* deserialize(string data) {
        if (data.empty()) {
            return nullptr;
        }
        stringstream s(data);
        string str;
        getline(s, str, ',');
        TreeNode* root = new
          TreeNode(stoi(str));
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            TreeNode* node = q.front();
            q.pop();
            getline(s, str, ',');
            if (str != "#") {
                TreeNode* leftNode = new
                  TreeNode(stoi(str));
                node->left = leftNode;
                q.push(leftNode);
            }
            getline(s, str, ',');
            if (str != "#") {
                TreeNode* rightNode = new
                  TreeNode(stoi(str));
                node->right = rightNode;
                q.push(rightNode);
            }
        }
        return root;
    }
};
```

Q265. Morris Preorder Traversal of a Binary Tree

HARD

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Input: Binary Tree: 4 2 5 3 -1 7 6 -1 9 -1 -1 8 -1 1

Output: [4 2 3 9 1 5 7 6 8]
Explanation: We traverse the binary tree in the order of Left, Root then Right.

Input : Binary Tree: 1 2 3 4 5 6 7 -1 -1 8 -1 -1 -1 9 10

Output: [1 2 4 5 8 3 6 7 9 10]
Explanation : We traverse the binary tree in the order of Left, Root then Right.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(2N), the time complexity is linear, as each node is visited at most twice (once for establi

```cpp
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    vector<int> getPreorder(TreeNode* root) {
        vector<int> preorder;
        TreeNode* cur = root;
        while (cur != NULL) {
            if (cur->left == NULL) {
                preorder.push_back(cur->val);
                cur = cur->right;
            } else {
                TreeNode* prev = cur->left;
                while (prev->right && prev->right != cur) {
                    prev = prev->right;
                }
                if (prev->right == NULL) {
                    prev->right = cur;
                    cur = cur->left;
                } else {
                    prev->right = NULL;
                    preorder.push_back(cur->val);
                    cur = cur->right;
                }
            }
        }
        return preorder;
    }
};
```

Q266. Morris Inorder Traversal of a Binary Tree

HARD

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Input: Binary Tree: 4 2 5 3 -1 7 6 -1 9 -1 -1 8 -1 1

Output: [3, 1, 9, 2, 4, 7, 5, 8, 6]
Explanation: We traverse the binary tree in the order of Left, Root then Right.

Input : Binary Tree: 1 2 3 4 5 6 7 -1 -1 8 -1 -1 -1 9 10

Output: [4,2,8,5,1,6,3,9,7,10]
Explanation : We traverse the binary tree in the order of Left, Root then Right.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(2N), the time complexity is linear, as each node is visited at most twice (once for establi

```cpp
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    vector<int> getInorder(TreeNode* root) {
        vector<int> inorder;
        TreeNode* cur = root;
        while (cur != NULL) {
            if (cur->left == NULL) {
                inorder.push_back(cur->val);
                cur = cur->right;
            } else {
                TreeNode* prev = cur->left;
                while (prev->right && prev->right != cur) {
                    prev = prev->right;
                }
                if (prev->right == NULL) {
                    prev->right = cur;
                    cur = cur->left;
                } else {
                    prev->right = NULL;
                    inorder.push_back(cur->val);
                    cur = cur->right;
                }
            }
        }
        return inorder;
    }
};
```

Q267. Flatten Binary Tree to Linked List

MEDIUM

Topic: Binary Trees [Traversals, Medium and Hard Problems] > Hard Problems

Practice Link

Example 1:
Input: Binary Tree: 1 2 5 3 4 -1 6 -1 -1 -1 -1 7

Output: 1 -1 2 -1 3 -1 4 -1 5 -1 6 -1 7

Explanation: The given binary tree has a preorder traversal order of [1, 2, 3, 4, 5, 6, 7]. To flatten the tree, we convert it into a linked list. In th... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(n), where n is the number of nodes in the binary tree. Each node i*

```cpp
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    TreeNode* prev = nullptr;
    void flatten(TreeNode* root) {
        if (root == nullptr) return;
        stack<TreeNode*> st;
        st.push(root);
        while (!st.empty()) {
            TreeNode* cur = st.top();
            st.pop();
            if (cur->right != nullptr)
              st.push(cur->right);
            if (cur->left != nullptr)
              st.push(cur->left);
            if (!st.empty()) cur->right =
              st.top();
            cur->left = nullptr;
        }
    }
};
void printPreorder(TreeNode* root) {
    if (!root) return;
    cout << root->val << " ";
    printPreorder(root->left);
    printPreorder(root->right);
}
void printFlattenTree(TreeNode* root) {
    if (!root) return;
    cout << root->val << " ";
    printFlattenTree(root->right);
}
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(2N) where N is the number of nodes in the Binary Tree. The time co*

```cpp
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
class Solution {
public:
    void flatten(TreeNode* root) {
        TreeNode* curr = root;
        while (curr) {
            if (curr->left) {
                TreeNode* pre = curr->left;
                while (pre->right) {
                    pre = pre->right;
                }
                pre->right = curr->right;
                curr->right = curr->left;
                curr->left = NULL;
            }
            curr = curr->right;
        }
    }
};
void printPreorder(TreeNode* root){
    if(!root){
        return;
    }
    cout << root->val << " ";
    printPreorder(root->left);
    printPreorder(root->right);
}
void printFlattenTree(TreeNode* root){
    if(!root){
        return;
    }
    cout << root->val << " ";
    printFlattenTree(root->right);
}
```

Q268. Introduction to BST

EASY

Topic: Binary Search Trees [Concept and Problems] > Concepts

Practice Link

Refer to the code implementation and logic below.

[OPTIMAL SOLUTION] Snippet 1 — N/A (Conceptual Tutorial)

```
Left Child < Node < Right Child
```

Q269. Search in a Binary Search Tree

EASY

Topic: Binary Search Trees [Concept and Problems] > Concepts

Practice Link

Input :8 5 12 4 7 10 14 -1 -1 6 -1 -1 -1 13 -1, Key = 10
Output : True

Input :4 2 6 1 3 5 7, Key = 3

Output :false
Explanation :

[OPTIMAL SOLUTION] Approach — Time Complexity: O(log N),Each step eliminates half of the tree, just like binary search. However, in the wors

```cpp
#include <bits/stdc++.h>
using namespace std;

// Definition of TreeNode
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int data) {
        val = data;
        left = right = nullptr;
    }
};

class Solution {
public:
    // Function to search for a node with
      given key in BST
    TreeNode* searchBST(TreeNode* root, int
      target) {
        // Traverse until we reach a null node
          or the target is found
        while (root != nullptr && root->val !=
          target) {

            // If target is less than current
              node, go left
            if (target < root->val) {
                root = root->left;
            }

            // If target is greater than
              current node, go right
            else {
                root = root->right;
            }
        }

        // Return node if found, else nullptr
        return root;
    }
};

// Driver code
```

Q270. Floor in a Binary Search Tree

EASY

Topic: Binary Search Trees [Concept and Problems] > Practice Problems

Practice Link

Input : Binary Search Tree: 10 5 15 2 6 -1 -1, Key = 7

Output : 6
Explanation : In the given BST, the largest value smaller or equal to 7 is 6.

Input : Binary Search Tree: 8 5 12 4 7 10 14 -1 -1 6 -1 -1 -1 13, Key = 9

Output : 8
Explanation : In the given BST, the largest value smaller or equal to 9 is 8.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(log2N), where N is the number of nodes in the Binary Search Tree. The complexity is equival

```cpp
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
class Solution {
public:
    int floorInBST(TreeNode* root, int key){
        int floor = -1;
        while(root){
            if(root->val == key){
                floor = root->val;
                return floor;
            }
            if(key > root->val){
                floor = root->val;
                root = root->right;
            }
            else{
                root = root->left;
            }
        }
        return floor;
    }
};
void printInOrder(TreeNode* root) {
    if (root == nullptr) return;
    printInOrder(root->left);
    cout << root->val << " ";
    printInOrder(root->right);
}
```

Q271. Kth Smallest and Largest element in BST

MEDIUM

Topic: Binary Search Trees [Concept and Problems] > Practice Problems

Practice Link

Example 1:
Input: root = [3, 1, 4, null, 2], k = 1
Output: [1, 4]
Explanation:
The 1st smallest value in the given BST is 1.
The 1st largest value in the given BST is 4.

Example 2:
Input: root = [5, 3, 6, 2, null, null, null, 1], k = 3
Output: [3, 3]
Explanation:
The 3rd smallest value in the given BST is 3.
The 3rd largest value in the given BST is 3.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N), where N is the number of nodes in the binary tree. The reason is that in the worst-case

```
#include <bits/stdc++.h>
using namespace std;
  struct TreeNode {
      int data;
      TreeNode *left;
      TreeNode *right;
      TreeNode(int val) : data(val),
        left(nullptr), right(nullptr) {}
 };
class Solution {
public:
    int kthSmallest(TreeNode* root, int k) {
        this->k = k;
        this->result = -1;
        inorder(root);
        return result;
    }
    int kthLargest(TreeNode* root, int k) {
        this->k = k;
        this->result = -1;
        reverse_inorder(root);
        return result;
    }
    vector<int> kLargesSmall(TreeNode* root,
      int k) {
        return {kthSmallest(root, k),
          kthLargest(root, k)};
    }
private:
    int k;
    int result;
    void inorder(TreeNode* node) {
        if (node != nullptr) {
            inorder(node->left);
            if (--k == 0) {
                result = node->data;
                return;
            }
            inorder(node->right);
        }
    }
    void reverse_inorder(TreeNode* node) {
        if (node != nullptr) {
            reverse_inorder(node->right);
            if (--k == 0) {
                result = node->data;
                return;
            }
            reverse_inorder(node->left);
        }
    }
};
```

Q272. Inorder Successor/Predecessor in BST

MEDIUM

Topic: Binary Search Trees [Concept and Problems] > Practice Problems

Practice Link

Example 1:
Input: Binary Search Tree: 5 3 7 2 4 6 9 1 -1 -1 -1 -1 -1 8 10, Key = 8
Output: Inorder Predecessor: 7, Inorder Successor: 9
Explanation:
1/2

Example 2:
Input: Bina... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N) where N is the number of nodes in the binary search tree. This*

```cpp
#include <bits/stdc++.h>
using namespace std;
// define tree node structure
struct TreeNode {
    // value of the node
    int val;
    // pointer to the left child
    TreeNode* left;
    // pointer to the right child
    TreeNode* right;
    // constructor
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
// solution class
class Solution {
public:
    // find inorder successor in BST by
      tracking candidate
    TreeNode* inorderSuccessor(TreeNode* root,
      TreeNode* p) {
        // initialize successor pointer
        TreeNode* successor = nullptr;
        // traverse until null
        while (root != nullptr) {
            // move left while updating
              successor when root > p
            if (root->val > p->val) {
                successor = root;
                root = root->left;
            }
            // otherwise move right
            else {
                root = root->right;
            }
        }
        // return final successor (or null)
        return successor;
    }
};
// print inorder traversal
void printInOrder(TreeNode* root) {
    // base case for recursion
    if (root == nullptr) return;
    // traverse left subtree
    printInOrder(root->left);
    // print current node
    cout << root->val << " ";
    // traverse right subtree
    printInOrder(root->right);
}
// program entry
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(H) where H is the height of the binary search tree as we are trave*

```cpp
#include <bits/stdc++.h>
using namespace std;
// define tree node structure
struct TreeNode {
    // value of the node
    int val;
    // pointer to left child
    TreeNode* left;
    // pointer to right child
    TreeNode* right;
    // constructor
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
// solution class
class Solution {
public:
    // find inorder successor iteratively
    TreeNode* inorderSuccessor(TreeNode* root,
      TreeNode* p) {
        // initialize successor
        TreeNode* successor = nullptr;
        // traverse until root becomes null
        while (root != nullptr) {
            // when p is greater or equal,
              move right
            if (p->val >= root->val) {
                root = root->right;
            }
            // otherwise update successor and
              move left
            else {
                successor = root;
                root = root->left;
            }
        }
        // return successor (may be null)
        return successor;
    }
};
// inorder print helper
void printInOrder(TreeNode* root) {
    // base case
    if (root == nullptr) return;
    // traverse left
    printInOrder(root->left);
    // print node
    cout << root->val << " ";
    // traverse right
    printInOrder(root->right);
}
// program entry
```

Q273. Merge 2 BST's

HARD

Topic: Binary Search Trees [Concept and Problems] > Practice Problems

Practice Link

Input: root1 = [5, 3, 6, 2, 4], root2 = [2, 1, 3, Null, Null, Null, 7, 6]
Output: [1, 2, 2, 3, 3, 4, 5, 6, 6, 7]
Explanation: After merging and sorting the two BST we get [1, 2, 2, 3, 3, 4, 5, 6, 6, 7].

Input: root1 = [12, 9, Null, 6, 11], root2 = [8, 5, 10, 2]
Output: [2, 5, 6, 8, 9, 10, 11, 12]
Explanation: After merging and sorting the two BST we get [2, 5, 6, 8, 9, 10, 11, 12].

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n+m), we traverse both the BSTs and merge two sorted lists. Space Complexity: O(m+n), a

```cpp
#include <bits/stdc++.h>
using namespace std;
// Structure for a binary tree node
struct Node {
    int data;
    Node* left;
    Node* right;
    // Constructor to initialize node
    Node(int val) : data(val), left(NULL),
      right(NULL) {}
};
class Solution {
public:
    // Function to perform inorder traversal
      and store elements
    void inorderTraversal(Node* root,
      vector<int>& arr) {
        // Base case
        if (!root) return;
        // Traverse left subtree
        inorderTraversal(root->left, arr);
        // Store current node data
        arr.push_back(root->data);
        // Traverse right subtree
        inorderTraversal(root->right, arr);
    }
    // Function to merge two sorted arrays
    vector<int> mergeArrays(vector<int>& arr1,
      vector<int>& arr2) {
        // Initialize result array
        vector<int> merged;
        // Initialize pointers
        int i = 0, j = 0;
        // Merge until one array ends
        while (i < arr1.size() && j <
          arr2.size()) {
            if (arr1[i] < arr2[j])
              merged.push_back(arr1[i++]);
            else merged.push_back(arr2[j++]);
        }
        // Add remaining elements
        while (i < arr1.size())
          merged.push_back(arr1[i++]);
        while (j < arr2.size())
          merged.push_back(arr2[j++]);
        return merged;
    }
    // Function to merge two BSTs in sorted
      order
    vector<int> mergeBSTs(Node* root1, Node*
      root2) {
        // Arrays to store inorder traversals
        vector<int> arr1, arr2;
        // Perform inorder traversal on first
          BST
        inorderTraversal(root1, arr1);
        // Perform inorder traversal on second
          BST
        inorderTraversal(root2, arr2);
        // Merge and return the result
        return mergeArrays(arr1, arr2);
    }
};
// Driver code
```

Q274. Two Sum In BST | Check if there exists a pair with Sum K

HARD

Topic: Binary Search Trees [Concept and Problems] > Practice Problems

Practice Link

Input: Binary Search Tree: 5 3 6 2 4 -1 7, K = 9

Output: True
Input: Binary Search Tree: 7 3 10 2 6 9 11 1 5 8 -1 -1 -1 -1 -1 4 -1 -1 -1

Output : True
Explanation:

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N) where N is the number of nodes in the BST as we have to traverse all the nodes using th

```cpp
#include <bits/stdc++.h>
using namespace std;
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr),
      right(nullptr) {}
};
class BSTIterator {
private:
    stack<TreeNode*> myStack;
    bool reverse;
public:
    BSTIterator(TreeNode* root, bool
      isReverse) : reverse(isReverse) {
        pushAll(root);
    }
    bool hasNext() {
        return !myStack.empty();
    }
    int next() {
        TreeNode* tmpNode = myStack.top();
        myStack.pop();
        if (!reverse) {
            pushAll(tmpNode->right);
        }
        else {
            pushAll(tmpNode->left);
        }
        return tmpNode->val;
    }
private:
    void pushAll(TreeNode* node) {
        while (node != nullptr) {
            myStack.push(node);
            if (reverse) {
                node = node->right;
            }
            else {
                node = node->left;
            }
        }
    }
};
class Solution {
public:
    bool findTarget(TreeNode* root, int k) {
        if (!root) return false;
        BSTIterator l(root, false);
        BSTIterator r(root, true);
        int i = l.next();
        int j = r.next();
        while (i < j) {
            if (i + j == k) return true;
            else if (i + j < k) i = l.next();
            else j = r.next();
        }
        return false;
    }
};
void printInOrder(TreeNode* root) {
    if (!root) return;
    printInOrder(root->left);
    cout << root->val << " ";
    printInOrder(root->right);
}
```

Q275. Introduction to Graph

EASY

Topic: Graphs [Concepts & Problems] > Learning

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // number of nodes
        int n = sc.nextInt();
        // number of edges
        int m = sc.nextInt();

        // adjacency matrix
        int[][] adj = new int[n + 1][n + 1];

        for (int i = 0; i < m; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            adj[u][v] = 1;
            // remove this line in case of
              directed graph
            adj[v][u] = 1;
        }

        sc.close();
    }
}
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```
Input:
5 6
1 2
1 3
2 4
3 4
3 5
4 5

Explanation:
Number of nodes, n = 5
Number of edges, m = 6
Next m lines represent the edges.
```

Q276. Graph Representation | C++

EASY

Topic: Graphs [Concepts & Problems] > Learning

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```
using namespace std;
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```
Input:
5 6
1 2
1 3
2 4
3 4
3 5
4 5

Explanation:
Number of nodes, n = 5
Number of edges, m = 6
Next m lines represent the edges.
```

Q277. Graph Representation | Java

EASY

Topic: Graphs [Concepts & Problems] > Learning

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // number of nodes
        int n = sc.nextInt();
        // number of edges
        int m = sc.nextInt();

        // adjacency matrix
        int[][] adj = new int[n + 1][n + 1];

        for (int i = 0; i < m; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            adj[u][v] = 1;
            // remove this line in case of
              directed graph
            adj[v][u] = 1;
        }

        sc.close();
    }
}
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```
Input:
5 6
1 2
1 3
2 4
3 4
3 5
4 5

Explanation:
Number of nodes, n = 5
Number of edges, m = 6
Next m lines represent the edges.
```

Q278. Connected Components

MEDIUM

Topic: Graphs [Concepts & Problems] > Learning

Practice Link

Input: V=4, edges=[[0,1],[1,2]]
Output: 2
Explanation: Vertices {0,1,2} forms the first component and vertex 3 forms the second component.

Input:V = 7, edges = [[0, 1], [1, 2], [2, 3], [4, 5]]
Output: 3
Explanation: The edges [0, 1], [1, 2], [2, 3] form a connected component with vertices {0, 1, 2, 3}
The edge [4, 5] forms another connected component with vertices {4, 5}.
Therefore, the graph has 3 connected components: {0, 1, 2, 3}, {4, 5}, and the isolated vertices {6}.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V+E),Each vertex is visited exactly once, and each edge is processed at most twice (once fr

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to count connected components
      in an undirected graph
    int countComponents(int V,
      vector<vector<int>>& edges) {
        // Create adjacency list from edge
          list
        vector<vector<int>> adj(V);
        for (auto &e : edges) {
            adj[e[0]].push_back(e[1]);
            adj[e[1]].push_back(e[0]);
        }
        // Vector to keep track of visited
          nodes
        vector<int> visited(V, 0);
        // Variable to count the number of
          connected components
        int components = 0;
        // Traverse all nodes in the graph
        for (int i = 0; i < V; ++i) {
            // If the node is not visited,
              it's a new component
            if (!visited[i]) {
                components++;
                // Start BFS from this node
                queue<int> q;
                q.push(i);
                visited[i] = 1;
                // Perform BFS traversal
                while (!q.empty()) {
                    int node = q.front();
                    q.pop();
                    // Visit all unvisited
                      neighbors
                    for (auto &nbr :
                      adj[node]) {
                        if (!visited[nbr]) {
                            visited[nbr] = 1;
                            q.push(nbr);
                        }
                    }
                }
            }
        }
        // Return the total number of
          connected components
        return components;
    }
};
```

Q279. Traversal Techniques

MEDIUM

Topic: Graphs [Concepts & Problems] > Learning

Practice Link

Input: V = 5, adj = [[2,3], [1,4,5], [1], [2,5], [4,5]]

Output: 1 2 4 5 3
Explanation: We traverse the entire graph using DFS technique.

Input: V = 4, adj = [[2,3], [1,4], [1], [2]]
Output: 1 2 4 3
Explanation: We traverse the entire graph using DFS technique.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V+E), each vertex is visited once and every edge is checked once in the adjacency list.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to perform DFS traversal
    void dfs(int v, vector<int> adj[],
             vector<int>& visited,
             vector<int>& result) {

        // Mark current node as visited
        visited[v] = 1;

        // Store node in result
        result.push_back(v);

        // Traverse all neighbours
        for (int u : adj[v]) {
            if (!visited[u]) {
                dfs(u, adj, visited, result);
            }
        }
    }
};
```

Q280. DFS

MEDIUM

Topic: Graphs [Concepts & Problems] > Learning

Practice Link

Input: V = 5, adj = [[2,3], [1,4,5], [1], [2,5], [4,5]]

Output: 1 2 4 5 3
Explanation: We traverse the entire graph using DFS technique.

Input: V = 4, adj = [[2,3], [1,4], [1], [2]]
Output: 1 2 4 3
Explanation: We traverse the entire graph using DFS technique.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V+E), each vertex is visited once and every edge is checked once in the adjacency list.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to perform DFS traversal
    void dfs(int v, vector<int> adj[],
             vector<int>& visited,
             vector<int>& result) {

        // Mark current node as visited
        visited[v] = 1;

        // Store node in result
        result.push_back(v);

        // Traverse all neighbours
        for (int u : adj[v]) {
            if (!visited[u]) {
                dfs(u, adj, visited, result);
            }
        }
    }
};
```

Q281. Number of provinces

MEDIUM

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Input: adj=[[1, 0, 0, 1], [0, 1, 1, 0], [0, 1, 1, 0], [1, 0, 0, 1]]
Output: 2
Explanation: In this graph, there are two provinces: [1, 4] and [2, 3]. City 1 and city 4 have a path between them, and city 2 and city 3 also have a path between them. There is no path between any city in province 1 and any city in province 2.

Input: adj= [[1, 0, 1], [0, 1, 0], [1, 0, 1]]
Output: 2
Explanation : The graph clearly has 2 Provinces [1,3] and [2]. As city 1 and city 3 has a path between them they be... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V+E), we visit every node and for every node we visit all of its neighbours in the DFS trav

```cpp
#include <bits/stdc++.h>
using namespace std;
// Solution class to hold the main logic
class Solution {
private:
    // Function to perform DFS traversal
    void dfs(int node, vector<int> adjList[],
      int visited[]) {
        // Mark current node as visited
        visited[node] = 1;
        // Visit all adjacent unvisited nodes
        for (auto neighbor : adjList[node]) {
            if (!visited[neighbor]) {
                dfs(neighbor, adjList,
                  visited);
            }
        }
    }
public:
    // Function to count number of connected
      components
    int numProvinces(vector<vector<int>> adj,
      int V) {
        // Create adjacency list from
          adjacency matrix
        vector<int> adjList[V];
        // Convert matrix to list
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                // Ignore self-loops
                if (adj[i][j] == 1 && i != j)
                  {
                    adjList[i].push_back(j);
                    adjList[j].push_back(i);
                }
            }
        }
        // Array to keep track of visited
          nodes
        int visited[V] = {0};
        // Count of connected components
        int count = 0;
        // Traverse all nodes
        for (int i = 0; i < V; i++) {
            // If node is not visited
            if (!visited[i]) {
                // Perform DFS and increment
                  count
                count++;
                dfs(i, adjList, visited);
            }
        }
        // Return final count
        return count;
    }
};
// Main function to drive the program
```

Q282. Connected Components Problem in Matrix

MEDIUM

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Input: V=4, edges=[[0,1],[1,2]]
Output: 2
Explanation: Vertices {0,1,2} forms the first component and vertex 3 forms the second component.

Input:V = 7, edges = [[0, 1], [1, 2], [2, 3], [4, 5]]
Output: 3
Explanation: The edges [0, 1], [1, 2], [2, 3] form a connected component with vertices {0, 1, 2, 3}
The edge [4, 5] forms another connected component with vertices {4, 5}.
Therefore, the graph has 3 connected components: {0, 1, 2, 3}, {4, 5}, and the isolated vertices {6}.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V+E),Each vertex is visited exactly once, and each edge is processed at most twice (once fr

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to count connected components
      in an undirected graph
    int countComponents(int V,
      vector<vector<int>>& edges) {
        // Create adjacency list from edge
          list
        vector<vector<int>> adj(V);
        for (auto &e : edges) {
            adj[e[0]].push_back(e[1]);
            adj[e[1]].push_back(e[0]);
        }
        // Vector to keep track of visited
          nodes
        vector<int> visited(V, 0);
        // Variable to count the number of
          connected components
        int components = 0;
        // Traverse all nodes in the graph
        for (int i = 0; i < V; ++i) {
            // If the node is not visited,
              it's a new component
            if (!visited[i]) {
                components++;
                // Start BFS from this node
                queue<int> q;
                q.push(i);
                visited[i] = 1;
                // Perform BFS traversal
                while (!q.empty()) {
                    int node = q.front();
                    q.pop();
                    // Visit all unvisited
                      neighbors
                    for (auto &nbr :
                      adj[node]) {
                        if (!visited[nbr]) {
                            visited[nbr] = 1;
                            q.push(nbr);
                        }
                    }
                }
            }
        }
        // Return the total number of
          connected components
        return components;
    }
};
```

Q283. Rotten Oranges

MEDIUM

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Input :grid = [[2, 1, 1] , [0, 1, 1] , [1, 0, 1]]
Output :-1
Explanation : Orange at (3,0) cannot be rotten.
Input :grid = [[2,1,1] , [1,1,0] , [0,1,1]]
Output :4
Explanation :

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n × n × 4) In the worst case, every cell in the grid may contain an orange, and for each ro

```cpp
#include <bits/stdc++.h>
using namespace std;
int orangesRotting(vector<vector<int>>& grid)
  {
    if (grid.empty()) return 0;
    int m = grid.size();
    int n = grid[0].size();
    int days = 0;
    int tot = 0;
    int cnt = 0;
    queue<pair<int, int>> rotten;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            if (grid[i][j] != 0)
                tot++;
            if (grid[i][j] == 2)
                rotten.push({i, j});
        }
    }
    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};
    while (!rotten.empty()) {
        int k = rotten.size();
        cnt += k;
        while (k--) {
            int x = rotten.front().first;
            int y = rotten.front().second;
            rotten.pop();
            for (int i = 0; i < 4; ++i) {
                int nx = x + dx[i];
                int ny = y + dy[i];
                if (nx < 0 || ny < 0 || nx >=
                  m || ny >= n || grid[nx][ny]
                  != 1)
                    continue;
                grid[nx][ny] = 2;
                rotten.push({nx, ny});
            }
        }
        if (!rotten.empty())
            days++;
    }
    return tot == cnt ? days : -1;
}
```

Q284. Cycle Detection in Undirected Graph (bfs)

HARD

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```
Example 2:
Input:
V = 8, E = 6

Output: 1

Explanation: 4->5->6->4 is a cycle.
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```
Example 1:
Input:
V = 8, E = 7

Output:  0

Explanation: No cycle in the given graph.
```

Q285. Detect a cycle in an undirected graph

HARD

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Input : V = 8, E = 7

Output : False
Explanation : No cycle is present in the given graph

Input : V = 8, E = 6

Output : True
Explanation : 4->5->6->4 is a cycle.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V+E), we build the adjacency list and explore all the edges and visit all the vertices once

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // DFS function to detect cycle
    bool dfs(int node, int parent, vector<int>
      adj[], vector<int>& visited) {
        // Mark current node visited
        visited[node] = 1;

        // Traverse neighbors
        for (int neighbor : adj[node]) {

            // If neighbor not visited,
              recurse
            if (!visited[neighbor]) {
                if (dfs(neighbor, node, adj,
                  visited)) return true;
            }

            // If neighbor visited and not
              parent, cycle exists
            else if (neighbor != parent) {
                return true;
            }
        }

        // No cycle found from this path
        return false;
    }

    // Function to check cycle in graph
    bool isCycle(int V, vector<int> adj[]) {
        vector<int> visited(V, 0);

        // Check all components
        for (int i = 0; i < V; i++) {
            if (!visited[i]) {
                if (dfs(i, -1, adj, visited))
                  return true;
            }
        }
        return false;
    }
};
```

Q286. Distance of nearest cell having one

MEDIUM

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Example 1:
Input:

Output:

Example 2:
Input:

Output:

[OPTIMAL SOLUTION] Approach — Time Complexity: O(NxM + NxMx4) ~ O(N x M), the BFS function will be called for (N x M) nodes, and for every n

```
#include<bits/stdc++.h>
using namespace std;
class Solution
{
    public:
    //Function to find the distance of nearest
      1 in the grid for each cell.
        vector<vector<int>>nearest(vector<vect
   or<int>>grid)
■{
■    int n = grid.size();
■    int m = grid[0].size();
■    // visited and distance matrix
           vector<vector<int>> vis(n,
      vector<int>(m, 0));
           vector<vector<int>> dist(n,
      vector<int>(m, 0));
■    // <coordinates, steps>
■    queue<pair<pair<int,int>, int>> q;
■    // traverse the matrix
■    for(int i = 0;i<n;i++) {
■        for(int j = 0;j<m;j++) {
■            // start BFS if cell contains 1
■            if(grid[i][j] == 1) {
■                q.push({{i,j}, 0});
■                vis[i][j] = 1;
■            }
■            else {
■                // mark unvisited
■                vis[i][j] = 0;
■            }
■        }
■    }
■    int delrow[] = {-1, 0, +1, 0};
■    int delcol[] = {0, +1, 0, -1};
■    // traverse till queue becomes empty
■    while(!q.empty()) {
■        int row = q.front().first.first;
■        int col = q.front().first.second;
■        int steps = q.front().second;
■        q.pop();
■        dist[row][col] = steps;
■        // for all 4 neighbours
■        for(int i = 0;i<4;i++) {
■            int nrow = row + delrow[i];
■            int ncol = col + delcol[i];
■            // check for valid unvisited cell
                    if(nrow >= 0 && nrow < n
                 && ncol >= 0 && ncol < m
■                && vis[nrow][ncol] == 0) {
■                    vis[nrow][ncol] = 1;
                            q.push({{nrow, ncol},
                        steps+1});
■            }
■        }
■    }
■    // return distance matrix
■    return dist;
■}
};
```

Q287. Surrounded Regions

MEDIUM

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Example 1:
Input: n = 5, m = 4

Output:

Example 2:
Input: n = 5, m = 4

Output:

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N × M), since each cell is visited at most once during DFS and once

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    void dfs(int row, int col, vector<vector<int>>& vis, vector<vector<char>>&
      mat, int delrow[], int delcol[]) {
        vis[row][col] = 1;
        int n = mat.size(), m = mat[0].size();
        for (int k = 0; k < 4; k++) {
            int nrow = row + delrow[k], ncol = col + delcol[k];
            if (nrow >= 0 && nrow < n && ncol >= 0 && ncol < m &&
              !vis[nrow][ncol] && mat[nrow][ncol] == 'O') {
                dfs(nrow, ncol, vis, mat, delrow, delcol);
            }
        }
    }
public:
    vector<vector<char>> fill(int n, int m, vector<vector<char>> mat) {
        if (n == 0 || m == 0) return mat;
        int delrow[4] = {-1, 0, 1, 0};
        int delcol[4] = {0, 1, 0, -1};
        vector<vector<int>> vis(n, vector<int>(m, 0));
        for (int j = 0; j < m; j++) {
            if (!vis[0][j] && mat[0][j] == 'O') dfs(0, j, vis, mat, delrow,
              delcol);
            if (!vis[n - 1][j] && mat[n - 1][j] == 'O') dfs(n - 1, j, vis,
              mat, delrow, delcol);
        }
        for (int i = 0; i < n; i++) {
            if (!vis[i][0] && mat[i][0] == 'O') dfs(i, 0, vis, mat, delrow,
              delcol);
            if (!vis[i][m - 1] && mat[i][m - 1] == 'O') dfs(i, m - 1, vis,
              mat, delrow, delcol);
        }
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (!vis[i][j] && mat[i][j] == 'O') mat[i][j] = 'X';
            }
        }
        return mat;
    }
};
```

Q288. Number of enclaves

MEDIUM

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Example 1:
Input:

Output:
Explanation: The highlighted cells represent the land cells.
The highlighted cells represent the land cells.

Example 2:
Input:

Output:

Explanation: The highlighted cells represent the land cells.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n × m), where n is the number of rows and m is the number of column

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    int numberOfEnclaves(vector<vector<int>>&
      grid) {
        if (grid.empty() || grid[0].empty())
          return 0;
        int n = (int)grid.size();
        int m = (int)grid[0].size();
        vector<vector<int>> vis(n,
          vector<int>(m, 0));
        queue<pair<int,int>> q;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (i == 0 || j == 0 || i == n
                  - 1 || j == m - 1) {
                    if (grid[i][j] == 1) {
                        vis[i][j] = 1;
                        q.push({i, j});
                    }
                }
            }
        }
        int delrow[] = {-1, 0, +1, 0};
        int delcol[] = {0, +1, 0, -1};
        while (!q.empty()) {
            auto [row, col] = q.front();
            q.pop();
            for (int k = 0; k < 4; k++) {
                int nrow = row + delrow[k];
                int ncol = col + delcol[k];
                if (nrow >= 0 && nrow < n &&
                  ncol >= 0 && ncol < m
                    && !vis[nrow][ncol] &&
                      grid[nrow][ncol] == 1) {
                    vis[nrow][ncol] = 1;
                    q.push({nrow, ncol});
                }
            }
        }
        int cnt = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == 1 &&
                  vis[i][j] == 0) cnt++;
            }
        }
        return cnt;
    }
};
```

Q289. Word ladder I

HARD

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Example 1:
Input:
wordList = {"des","der","dfr","dgt","dfs"}
startWord = "der", targetWord = "dfs"
Output:
3
Explanation:
The length of the smallest transformation sequence from "der" to
"dfs" is 3 i.e. "der" -> (replace ‘e’ by ‘f’) -> "dfr" -> (replace ‘r’ by ‘s’) -> "dfs". So, it takes 3 different strings for us to reach the targetWord. Each of these strings are present in the wordList.

Example 2:
Input:
wordList = {"geek", "gefk"}
startWord = "gedk", targetWord= "geek"
Output:
2
Explanati... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N * L * 26), where N is the number of words in the list and L is th

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to calculate the shortest
      transformation sequence length
    int wordLadderLength(string startWord,
      string targetWord, vector<string>&
      wordList) {
        // Queue for BFS storing {current
          word, steps taken}
        queue<pair<string, int>> q;
        q.push({startWord, 1});

        // Set for quick lookup and deletion
        unordered_set<string>
          st(wordList.begin(),
          wordList.end());
        st.erase(startWord);

        while (!q.empty()) {
            string word = q.front().first;
            int steps = q.front().second;
            q.pop();

            // If target word is found, return
              steps
            if (word == targetWord) return
              steps;

            // Try changing every character in
              the current word
            for (int i = 0; i < word.size();
              i++) {
                char original = word[i];
                for (char ch = 'a'; ch <= 'z';
                  ch++) {
                    word[i] = ch;
                    if (st.find(word) !=
                      st.end()) {
                        st.erase(word);
                        q.push({word, steps +
                          1});
                    }
                }
                word[i] = original;
            }
        }
        // If no sequence exists
        return 0;
    }
};
```

Q290. Word ladder II

HARD

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Example 1:
Input:
startWord = "der", targetWord = "dfs",
wordList = {"des","der","dfr","dgt","dfs"}
Output:
[[“der”, “dfr”, “dfs”], [“der”, “des”, “dfs”]]
Explanation:
The length of the smallest transformation sequence here is 3.
Following are the only t... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N × L × 26 + S × L) → dominated by generating all transformations (N = words, L = word leng

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution
{
public:
    vector<vector<string>>
      findSequences(string beginWord, string
      endWord,
vector<string> &wordList)
    {
        unordered_set<string>
          st(wordList.begin(),
          wordList.end());
        queue<vector<string>> q;
        q.push({beginWord});
        vector<string> usedOnLevel;
        usedOnLevel.push_back(beginWord);
        int level = 0;
        vector<vector<string>> ans;
        while (!q.empty())
        {
            vector<string> vec = q.front();
            q.pop();
            if (vec.size() > level)
            {
                level++;
                for (auto it : usedOnLevel)
                {
                    st.erase(it);
                }
            }
            string word = vec.back();
            if (word == endWord)
            {
                if (ans.size() == 0)
                {
                    ans.push_back(vec);
                }
                else if (ans[0].size() ==
                  vec.size())
                {
                    ans.push_back(vec);
                }
            }
            for (int i = 0; i < word.size();
              i++)
            {
                char original = word[i];
                for (char c = 'a'; c <= 'z';
                  c++)
                {
                    word[i] = c;
                    if (st.count(word) > 0)
                    {
                        vec.push_back(word);
                        q.push(vec);
usedOnLevel.push_back(word);
                        vec.pop_back();
                    }
                }
                word[i] = original;
            }
        }
        return ans;
    }
};
bool comp(vector<string> a, vector<string> b)
{
    string x = "", y = "";
    for (string i : a)
        x += i;
    for (string i : b)
        y += i;
    return x < y;
}
```

Q291. Number of islands

MEDIUM

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Input:

Output: 1
Explanation: Island at the top left corner is the same as the island at the bottom right corner.

Input :

Output: 3
Explanation : Island at the top right corner is the same as the island at the bottom left corner.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N*M), DFS traversal and marking visited cells dominate. Space C

```
#include <bits/stdc++.h>
using namespace std
class Solution {
public:
    // DFS traversal
    void dfs(int row, int col, int baseRow,
      int baseCol,
            vector<vector<int>>& grid,
              vector<vector<int>>& vis,
            vector<pair<int, int>>& shape) {
        // Mark as visited
        vis[row][col] = 1;
        // Store relative position
        shape.push_back({row - baseRow, col -
          baseCol});
        // All 4 directions
        int drow[] = {-1, 0, 1, 0};
        int dcol[] = {0, 1, 0, -1};
        // Explore all directions
        for (int i = 0; i < 4; i++) {
            int nrow = row + drow[i];
            int ncol = col + dcol[i];
            // Check bounds and if land and
              not visited
            if (nrow >= 0 && nrow <
              grid.size() &&
                ncol >= 0 && ncol <
                  grid[0].size() &&
                !vis[nrow][ncol] &&
                  grid[nrow][ncol] == 1) {
                dfs(nrow, ncol, baseRow,
                  baseCol, grid, vis, shape);
            }
        }
    }
    // Main function to return number of
      distinct islands
    int
      countDistinctIslands(vector<vector<int>>
      & grid) {
        int n = grid.size();
        int m = grid[0].size();
        vector<vector<int>> vis(n,
          vector<int>(m, 0));
        set<vector<pair<int, int>>> st;
        // Traverse grid
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                // If land and not visited
                if (grid[i][j] == 1 &&
                  !vis[i][j]) {
                    vector<pair<int, int>>
                      shape;
                    dfs(i, j, i, j, grid, vis,
                      shape);
                    st.insert(shape);
                }
            }
        }
        return st.size();
    }
};
// Driver code
```

Q292. Bipartite Graph (DFS)

HARD

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```java
import java.util.*;
class Solution
{
    private boolean dfs(int node, int col, int
      color[],
    ArrayList<ArrayList<Integer>>adj) {
        color[node] = col;
        // traverse adjacent nodes
        for(int it : adj.get(node)) {
            // if uncoloured
            if(color[it] == -1) {
                if(dfs(it, 1 - col, color,
                  adj) == false) return false;
            }
            // if previously coloured and have
              the same colour
            else if(color[it] == col) {
                return false;
            }
        }
        return true;
    }
    public boolean isBipartite(int V,
      ArrayList<ArrayList<Integer>>adj)
    {
        int color[] = new int[V];
■    for(int i = 0;i<V;i++) color[i] = -1;
■    // for connected components
■    for(int i = 0;i<V;i++) {
■        if(color[i] == -1) {
                    if(dfs(i, 0, color, adj)
                == false) return false;
■        }
■    }
■    return true;
    }
     public static void main(String[] args)
    {
        // V = 4, E = 4
        ArrayList < ArrayList < Integer >> adj
          = new ArrayList < > ();
        for (int i = 0; i < 4; i++) {
            adj.add(new ArrayList < > ());
        }
        adj.get(0).add(2);
        adj.get(2).add(0);
        adj.get(0).add(3);
        adj.get(3).add(0);
        adj.get(1).add(3);
        adj.get(3).add(1);
        adj.get(2).add(3);
        adj.get(3).add(2);
        Solution obj = new Solution();
        boolean ans = obj.isBipartite(4, adj);
        if(ans)
            System.out.println("1");
        else System.out.println("0");
    }
}
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
private:
    bool dfs(int node, int col, int color[],
      vector<int> adj[]) {
        color[node] = col;

        // traverse adjacent nodes
        for(auto it : adj[node]) {
            // if uncoloured
            if(color[it] == -1) {
                if(dfs(it, !col, color, adj)
                  == false) return false;
            }
            // if previously coloured and have
              the same colour
            else if(color[it] == col) {
                return false;
            }
        }

        return true;
    }
public:
■bool isBipartite(int V, vector<int>adj[]){
■    int color[V];
■    for(int i = 0;i<V;i++) color[i] = -1;

■    // for connected components
■    for(int i = 0;i<V;i++) {
■        if(color[i] == -1) {
                    if(dfs(i, 0, color, adj)
                == false)
■                return false;
■        }
■    }
■    return true;
■}
};

void addEdge(vector <int> adj[], int u, int v)
  {
    adj[u].push_back(v);
    adj[v].push_back(u);
}
```

Q293. Cycle Detection in Directed Graph (DFS)

HARD

Topic: Graphs [Concepts & Problems] > Problems on BFS/DFS

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```
Input Format: N = 10, E = 10

Result: false

Explanation: No cycle detected.
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```
Input: N = 10, E = 11

Output: true

Explanation: 8->9->10 is a cycle.
```

Q294. Topo Sort

HARD

Topic: Graphs [Concepts & Problems] > Topo Sort and Problems

Practice Link

Input: V = 6, adj = [[], [], [3], [1], [0, 1], [0, 2]]
Output: [5, 4, 2, 3, 1, 0]
Explanation: A graph may have multiple topological sortings. The result is one of them. The necessary conditions for the ordering are:
According to edge 5 -> 0, node 5 must appear before node 0 in the ordering.
According to edge 4 -> 0, node 4 must appear before node 0 in the ordering.
According to edge 5 -> 2, node 5 must appear before node 2 in the ordering.
According to edge 2 -> 3, node 2 must appea... (cont.)

[OPTIMAL SOLUTION] General — Time Complexity: O(V + E) because we visit each vertex exactly once and process all outgoing edges from each v

```cpp
#include <bits/stdc++.h>
using namespace std;
// Creating a class named Solution
class Solution {
public:
    // Function to perform BFS-based
      Topological Sort
    vector<int> topologicalSort(int V,
      vector<int> adj[]) {
        // Create a vector to store the
          in-degree of each vertex
        vector<int> indegree(V, 0);
        // Loop through all vertices to
          calculate in-degree
        for (int i = 0; i < V; i++) {
            // Traverse all adjacent vertices
              of the current vertex
            for (auto it : adj[i]) {
                // Increase in-degree of the
                  connected vertex
                indegree[it]++;
            }
        }
        // Create a queue to store vertices
          with in-degree zero
        queue<int> q;
        // Loop through all vertices
        for (int i = 0; i < V; i++) {
            // If in-degree is zero, add to
              queue
            if (indegree[i] == 0) {
                q.push(i);
            }
        }
        // Vector to store the topological
          order
        vector<int> topo;
        // Process until queue is empty
        while (!q.empty()) {
            // Get the front vertex from the
              queue
            int node = q.front();
            q.pop();
            // Add it to the topological order
            topo.push_back(node);
            // Reduce in-degree of its
              adjacent vertices
            for (auto it : adj[node]) {
                indegree[it]--;
                // If in-degree becomes zero,
                  push it into queue
                if (indegree[it] == 0) {
                    q.push(it);
                }
            }
        }
        // Return the topological ordering
        return topo;
    }
};
// Driver code
```

Q295. Topological sort or Kahn's algorithm

HARD

Topic: Graphs [Concepts & Problems] > Topo Sort and Problems

Practice Link

Input: V = 6, adj = [[], [], [3], [1], [0, 1], [0, 2]]
Output: [5, 4, 2, 3, 1, 0]
Explanation: A graph may have multiple topological sortings. The result is one of them. The necessary conditions for the ordering are:
According to edge 5 -> 0, node 5 must appear before node 0 in the ordering.
According to edge 4 -> 0, node 4 must appear before node 0 in the ordering.
According to edge 5 -> 2, node 5 must appear before node 2 in the ordering.
According to edge 2 -> 3, node 2 must appea... (cont.)

[OPTIMAL SOLUTION] General — Time Complexity: O(V + E) because we visit each vertex exactly once and process all outgoing edges from each v

```cpp
#include <bits/stdc++.h>
using namespace std;
// Creating a class named Solution
class Solution {
public:
    // Function to perform BFS-based
      Topological Sort
    vector<int> topologicalSort(int V,
      vector<int> adj[]) {
        // Create a vector to store the
          in-degree of each vertex
        vector<int> indegree(V, 0);
        // Loop through all vertices to
          calculate in-degree
        for (int i = 0; i < V; i++) {
            // Traverse all adjacent vertices
              of the current vertex
            for (auto it : adj[i]) {
                // Increase in-degree of the
                  connected vertex
                indegree[it]++;
            }
        }
        // Create a queue to store vertices
          with in-degree zero
        queue<int> q;
        // Loop through all vertices
        for (int i = 0; i < V; i++) {
            // If in-degree is zero, add to
              queue
            if (indegree[i] == 0) {
                q.push(i);
            }
        }
        // Vector to store the topological
          order
        vector<int> topo;
        // Process until queue is empty
        while (!q.empty()) {
            // Get the front vertex from the
              queue
            int node = q.front();
            q.pop();
            // Add it to the topological order
            topo.push_back(node);
            // Reduce in-degree of its
              adjacent vertices
            for (auto it : adj[node]) {
                indegree[it]--;
                // If in-degree becomes zero,
                  push it into queue
                if (indegree[it] == 0) {
                    q.push(it);
                }
            }
        }
        // Return the topological ordering
        return topo;
    }
};
// Driver code
```

Q296. Detect a cycle in a directed graph

HARD

Topic: Graphs [Concepts & Problems] > Topo Sort and Problems

Practice Link

Input : V = 5, E = 5

Output : True
Explanation : 2->3->4->2 is a cycle.

Input : V = 4, E = 3

Output : False
Explanation : There is no cycle in the graph. This is a directed acyclic graph.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V+E), we traverse all the edges and visit all the vertices once.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to detect cycle in a directed
      graph using Kahn's Algorithm
    bool hasCycle(int V, vector<vector<int>>&
      adj) {
        // Store in-degrees of all nodes
        vector<int> indegree(V, 0);
        for (int u = 0; u < V; u++) {
            for (int v : adj[u]) {
                indegree[v]++;
            }
        }

        // Queue for nodes with 0 in-degree
        queue<int> q;
        for (int i = 0; i < V; i++) {
            if (indegree[i] == 0) q.push(i);
        }

        // Count visited nodes
        int count = 0;
        while (!q.empty()) {
            int node = q.front();
            q.pop();
            count++;

            // Reduce in-degree of neighbors
            for (int neighbor : adj[node]) {
                indegree[neighbor]--;
                if (indegree[neighbor] == 0) {
                    q.push(neighbor);
                }
            }
        }

        // If count != V, cycle exists
        return count != V;
    }
};
```

Q297. Course Schedule I

HARD

Topic: Graphs [Concepts & Problems] > Topo Sort and Problems

Practice Link

Input: N = 4, P = 3, array[] = {{1,0},{2,1},{3,2}}
Output: Yes
Explanation: It is possible to finish all the tasks in the order : 3 2 1 0.
First, we will finish task 3. Then we will finish task 2, task 1, and task 0.

Input: N = 4, P = 4, array[] = {{1,2},{4,3},{2,4},{4,1}}
Output: No
Explanation: It is impossible to finish all the tasks. Let's analyze the pairs:
For pair {1, 2} -> we need to finish task 1 first and then task 2. (order : 1 2).
For pair{4, 3} -> we need to finish task 4 first... (cont.)

[OPTIMAL SOLUTION] General — Time Complexity: O(V+E), each course and pre-requisite edge is processed once. Spa

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find ordering of courses
    vector<int> findOrder(int numCourses,
                          vector<vector<int>>&
                            prerequisites) {
        // Build adjacency list
        vector<vector<int>> adj(numCourses);
        // Build in-degree array
        vector<int> inDegree(numCourses, 0);
        // Fill adjacency and in-degree
        for (auto& pre : prerequisites) {
            int a = pre[0], b = pre[1];
            adj[b].push_back(a);
            inDegree[a]++;
        }
        // Initialize queue with zero
          in-degree nodes
        queue<int> q;
        for (int i = 0; i < numCourses; i++) {
            if (inDegree[i] == 0) {
                q.push(i);
            }
        }
        // Create result list
        vector<int> order;
        // Process queue
        while (!q.empty()) {
            int node = q.front();
            q.pop();
            order.push_back(node);
            // Reduce in-degree of neighbors
            for (int nei : adj[node]) {
                inDegree[nei]--;
                if (inDegree[nei] == 0) {
                    q.push(nei);
                }
            }
        }
        // Return order if valid, else empty
        if ((int)order.size() == numCourses) {
            return order;
        }
        return {};
    }
};
```

Q298. Course Schedule II

MEDIUM

Topic: Graphs [Concepts & Problems] > Topo Sort and Problems

Practice Link

Input: N = 4, P = 3, array[] = {{1,0},{2,1},{3,2}}
Output: Yes
Explanation: It is possible to finish all the tasks in the order : 3 2 1 0.
First, we will finish task 3. Then we will finish task 2, task 1, and task 0.

Input: N = 4, P = 4, array[] = {{1,2},{4,3},{2,4},{4,1}}
Output: No
Explanation: It is impossible to finish all the tasks. Let's analyze the pairs:
For pair {1, 2} -> we need to finish task 1 first and then task 2. (order : 1 2).
For pair{4, 3} -> we need to finish task 4 first... (cont.)

[OPTIMAL SOLUTION] General — Time Complexity: O(V+E), each course and pre-requisite edge is processed once. Spa

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find ordering of courses
    vector<int> findOrder(int numCourses,
                          vector<vector<int>>&
                            prerequisites) {
        // Build adjacency list
        vector<vector<int>> adj(numCourses);
        // Build in-degree array
        vector<int> inDegree(numCourses, 0);
        // Fill adjacency and in-degree
        for (auto& pre : prerequisites) {
            int a = pre[0], b = pre[1];
            adj[b].push_back(a);
            inDegree[a]++;
        }
        // Initialize queue with zero
          in-degree nodes
        queue<int> q;
        for (int i = 0; i < numCourses; i++) {
            if (inDegree[i] == 0) {
                q.push(i);
            }
        }
        // Create result list
        vector<int> order;
        // Process queue
        while (!q.empty()) {
            int node = q.front();
            q.pop();
            order.push_back(node);
            // Reduce in-degree of neighbors
            for (int nei : adj[node]) {
                inDegree[nei]--;
                if (inDegree[nei] == 0) {
                    q.push(nei);
                }
            }
        }
        // Return order if valid, else empty
        if ((int)order.size() == numCourses) {
            return order;
        }
        return {};
    }
};
```

Q299. Find eventual safe states

HARD

Topic: Graphs [Concepts & Problems] > Topo Sort and Problems

Practice Link

Example 1:
Input Format: N = 7, E = 7

Result: {2 4 5 6}

Example 2:
Input Format: N = 4, E = 4

Result: {3}

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V + E) + O(N log N), O(V + E) for performing BFS on th

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find all eventual safe
      nodes in a directed graph
    vector<int> eventualSafeNodes(int V,
      vector<int> adj[]) {
        vector<int> adjRev[V];  // Reverse
          adjacency list
        int indegree[V] = {0};  // Indegree
          array to track nodes with no
          outgoing edges

        // Build the reverse graph and
          calculate indegrees
        for (int i = 0; i < V; i++) {
            for (auto it : adj[i]) {
                adjRev[it].push_back(i);  //
                  Reverse the direction of
                  edges
                indegree[i]++;  // Increment
                  indegree for the current
                  node
            }
        }

        queue<int> q;  // Queue to store nodes
          with no outgoing edges (safe nodes)
        vector<int> safeNodes;

        // Add all nodes with 0 indegree to
          the queue
        for (int i = 0; i < V; i++) {
            if (indegree[i] == 0) {
                q.push(i);
            }
        }

        // Process the queue to find all safe
          nodes
        while (!q.empty()) {
            int node = q.front();
            q.pop();
            safeNodes.push_back(node);  //
              This node is safe
            for (auto it : adjRev[node]) {
                indegree[it]--;  // Decrease
                  indegree of the parent nodes
                if (indegree[it] == 0) {
                    q.push(it);  // If
                      indegree becomes 0, it
                      is a safe node
                }
            }
        }

        sort(safeNodes.begin(),
          safeNodes.end());  // Sort the safe
          nodes
        return safeNodes;
    }
};
```

Q300. Alien Dictionary

HARD

Topic: Graphs [Concepts & Problems] > Topo Sort and Problems

Practice Link

Example 1:
Input: N = 5, K = 4
dict = {"baa","abcd","abca","cab","cad"}
Output: b d a c
Explanation:
We will analyze every consecutive pair to find out the order of the characters. The pair “baa” and “abcd” suggests ‘b’ appears before ‘a’ in the alien dictionary.The pair “abcd” and “abca” suggests ‘d’ appears before ‘a’ in the alien dictionary. The pair “abca” and “cab” suggests ‘a’ appears before ‘c’ in the alien dictionary. The pair “cab” and “cad” suggests ‘b’ appears before ‘d’ in the alien... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N*len)+O(K+E), where N is the number of words in the dictionary, ‘len’ is the length up to

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    vector<int> topoSort(int V, vector<int>
      adj[]) {
        vector<int> indegree(V, 0);
        for (int i = 0; i < V; i++) {
            for (auto neighbor : adj[i]) {
                indegree[neighbor]++;
            }
        }
        queue<int> q;
        for (int i = 0; i < V; i++) {
            if (indegree[i] == 0) {
                q.push(i);
            }
        }
        vector<int> topo;
        while (!q.empty()) {
            int node = q.front();
            q.pop();
            topo.push_back(node);
            for (auto neighbor : adj[node]) {
                indegree[neighbor]--;
                if (indegree[neighbor] == 0) {
                    q.push(neighbor);
                }
            }
        }
        return topo;
    }
public:
    string findOrder(string dict[], int N, int
      K) {
        vector<int> adj[K];
        for (int i = 0; i < N - 1; i++) {
            string s1 = dict[i];
            string s2 = dict[i + 1];
            int len = min(s1.size(),
              s2.size());
            for (int ptr = 0; ptr < len;
              ptr++) {
                if (s1[ptr] != s2[ptr]) {
                    adj[s1[ptr] -
                      'a'].push_back(s2[ptr] -
                      'a');
                    break;
                }
            }
        }
        vector<int> topo = topoSort(K, adj);
        string ans = "";
        for (auto node : topo) {
            ans += char(node + 'a');
        }
        return ans;
    }
};
```

Q301. Shortest path in undirected graph with unit weights

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Input: n = 9, m = 10 edges = [[0,1],[0,3],[3,4],[4 ,5],[5,6],[1,2],[2,6],[6,7],[7,8],[6,8]] src = 0
Output: 0 1 2 1 2 3 3 4 4
Explanation: The above output array shows the shortest path to all the nodes from the source vertex (0). Dist[0] = 0, Dist[1] = 1, Dist[2] = 2, … Dist[8] = 4 Where Dist[node] is the shortest distance from the source to that node. If Dist[node] = -1, then that node is unreachable from the source.

Input: n = 8, m = 10 edges = [[1,0],[2,1],[0,3],[3,7],[3,4],[7,4],[7,6],... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(M) { for creating the adjacency list from given list 'edges'} + O(N + 2M) { for the BFS Alg

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find the shortest path from
      source to all vertices
    vector<int>
      shortestPath(vector<vector<int>>& edges,
      int N, int M, int src) {
        // Create an adjacency list of size N
          to store the undirected graph
        vector<int> adj[N];
        // Build the graph by adding edges to
          the adjacency list
        for (auto it : edges) {
            adj[it[0]].push_back(it[1]);
            adj[it[1]].push_back(it[0]);
        }
        // Initialize the distance array with
          a large value (treated as infinity)
        int dist[N];
        for (int i = 0; i < N; i++)
            dist[i] = 1e9;
        // Set the source node distance to 0
        dist[src] = 0;
        // Use a queue to perform BFS
          traversal
        queue<int> q;
        // Start BFS from the source node
        q.push(src);
        // Loop until queue is empty
        while (!q.empty()) {
            // Get the front node from the
              queue
            int node = q.front();
            q.pop();
            // Traverse all adjacent nodes
            for (auto it : adj[node]) {
                // If a shorter path to
                  neighbor is found
                if (dist[node] + 1 < dist[it])
                  {
                    dist[it] = 1 + dist[node];
                    q.push(it);
                }
            }
        }
        // Initialize the result array with -1
          for unreachable nodes
        vector<int> ans(N, -1);
        // Fill in the result array with
          distances where reachable
        for (int i = 0; i < N; i++) {
            if (dist[i] != 1e9) {
                ans[i] = dist[i];
            }
        }
        // Return the final shortest distances
        return ans;
    }
};
// Driver code
```

Q302. Shortest path in DAG

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Input:n = 7, m = 8 ,Edges = [[0,4,2], [0,5,3], [5,4,1], [4,6,3], [4,2,1], [6,1,2], [2,3,3], [1,3,1]]
Output:0 7 3 6 2 3 5
Explanation: The output represents the shortest distance from the source node (0) to all other nodes.
Dist[0] = 0
Dist[1] = 7
Dist[2] = 3
Dist[3] = 6
Dist[4] = 2
Dist[5] = 3
Dist[6] = 5

Input:n = 6, m = 7, edges = [[0,1,2], [0,4,1], [4,5,4], [4,2,2], [1,2,3], [2,3,6], [5,3,1]]
Output:0 2 3 6 1 5
Explanation: The output represents the shortest distances from the source node (0)... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N+M) {for the topological sort} + O(N+M) {for relaxation of vertices, each node and its adj

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
  private:
    void topoSort(int node, vector<pair<int,
      int>> adj[], int vis[], stack<int> &st)
      {
      vis[node] = 1;
      for (auto it : adj[node]) {
        int v = it.first;
        if (!vis[v]) {
          topoSort(v, adj, vis, st);
        }
      }
      st.push(node);
    }
  public:
    vector<int> shortestPath(int N, int M,
      vector<vector<int>> &edges) {
      vector<pair<int, int>> adj[N];
      for (int i = 0; i < M; i++) {
        int u = edges[i][0];
        int v = edges[i][1];
        int wt = edges[i][2];
        adj[u].push_back({v, wt});
      }
      int vis[N] = {0};
      stack<int> st;
      for (int i = 0; i < N; i++) {
        if (!vis[i]) {
          topoSort(i, adj, vis, st);
        }
      }
      vector<int> dist(N);
      for (int i = 0; i < N; i++) {
        dist[i] = 1e9;
      }
      dist[0] = 0;
      while (!st.empty()) {
        int node = st.top();
        st.pop();
        for (auto it : adj[node]) {
          int v = it.first;
          int wt = it.second;
          if (dist[node] + wt < dist[v]) {
            dist[v] = wt + dist[node];
          }
        }
      }
      for (int i = 0; i < N; i++) {
        if (dist[i] == 1e9) {
          dist[i] = -1;
        }
      }
      return dist;
    }
};
```

Q303. Djisktra's Algorithm

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Example 1:
Input:

V = 2
adj [] = {{{1, 9}}, {{0, 9}}}
S = 0
Output: 0 9
Explanation:
The source vertex is 0. Hence, the shortest distance of node 0 from the source is 0 and the shortest distance of node 1 from source will be 9.

Example 2:
Input:

V = 3, E = 3
adj = {{{1, 1}, {2, 6}}, {{2, 3}, {0, 1}}, {{1, 3}, {0, 6}}}
S = 2
Output:
4 3 0
Explanation:
For nodes 2 to 0, we can follow the path 2-1-0.
This has a distance of 1+3 = 4, whereas the path 2-0 has a distance of 6. So, the Shortest... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(E log V), as each edge leads to at most one insertion in the priority q

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution
{
    public:
        // Function to find the shortest
          distance of all the vertices
        // from the source vertex S.
        vector<int> dijkstra(int V,
          vector<vector<int>> adj[], int S)
        {
            // Create a set ds for storing the
              nodes as a pair {dist,node}
            // where dist is the distance from
              source to the node.
            // set stores the nodes in
              ascending order of the
              distances.
            set<pair<int, int>> st;
            // Initialising dist list with a
              large number to
            // indicate the nodes are
              unvisited initially.
            // This list contains distance
              from source to the nodes.
            vector<int> dist(V, 1e9);
            // Insert the source node with a
              distance of 0.
            st.insert({0, S});
            // Source initialised with dist =
              0
            dist[S] = 0;
            // Traverse the graph until the
              set is empty
            while(!st.empty()) {
                // Extract the node with the
                  minimum distance
                auto it = *(st.begin());
                int node = it.second;
                int dis = it.first;
                st.erase(it);
                // Check for all adjacent
                  nodes of the extracted node
                for(auto it : adj[node]) {
                    int adjNode = it[0];  //
                      Adjacent node
                    int edgW = it[1];     //
                      Weight of the edge
                    // If the new distance is
                      smaller, update it
                    if(dis + edgW <
                      dist[adjNode]) {
                        // Erase the previous
                          entry of the
                          adjacent node
                        // if it was visited
                          previously with a
                          larger cost.
                        if(dist[adjNode] !=
                          1e9)
st.erase({dist[adjNode], adjNode});
                        // Update the distance
                          for the adjacent
                          node
                        dist[adjNode] = dis +
                          edgW;
                        // Insert the adjacent
                          node with the
                          updated distance
                          into the set
st.insert({dist[adjNode], adjNode});
                    }
                }
            }
            // Return the list containing
              shortest distances
            // from source to all the nodes.
            return dist;
        }
};
```

Q304. Why priority Queue is used in Djisktra's Algorithm

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Input : V = 2, E = 1

Output : [0, 9]
Explanation : The source vertex is 0. Hence, the shortest distance of node 0 from the source is 0 and the shortest distance of node 1 from source will be 9.

Input : V = 3, E = 3

Output : [4, 3, 0]
Explanation : For nodes 2 to 0, we can follow the path 2-1-0. This has a distance of 1+3 = 4, whereas the path 2-0 has a distance of 6. So, the Shortest path from 2 to 0 is 4.

[OPTIMAL SOLUTION] Approach — Time Complexity: O((V+E)logV), each edge is relaxed at most once, and for each relaxation we may push into the

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to implement Dijkstra's
      Algorithm
    vector<int> dijkstra(int V,
      vector<vector<pair<int,int>>>& adj, int
      src) {
        // Distance array initialized to large
          value
        vector<int> dist(V, 1e9);

        // Min-heap storing {distance, node}
        priority_queue<pair<int,int>,
          vector<pair<int,int>>,
                       greater<pair<int,int>>>
                         pq;

        // Distance to source is 0
        dist[src] = 0;

        // Push source into heap
        pq.push({0, src});

        // Process nodes until heap is empty
        while (!pq.empty()) {
            // Extract node with minimum
              distance
            int d = pq.top().first;
            int node = pq.top().second;
            pq.pop();

            // Skip if this distance is
              outdated
            if (d > dist[node]) continue;

            // Traverse all adjacent neighbors
            for (auto it : adj[node]) {
                int next = it.first;
                int wt = it.second;

                // Relaxation check
                if (dist[node] + wt <
                  dist[next]) {
                    // Update distance
                    dist[next] = dist[node] +
                      wt;

                    // Push updated distance
                      into heap
                    pq.push({dist[next],
                      next});
                }
            }
        }
        return dist;
    }
};
```

Q305. Shortest Distance in a Binary Maze

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Example 1:
Input:
grid[][] = {{1, 1, 1, 1},
{1, 1, 0, 1},
{1, 1, 1, 1},
{1, 1, 0, 0},
{1, 0, 0, 1}}
source = {0, 1}, destination = {2, 2}
Output: 3
Explanation: The shortest path from source to destination goes through cells (0,1) → (1,1) → (2,1) → (2,2), total 3 steps.

Example 2:
Input:
grid[][] = {{1, 1, 1, 1, 1},
{1, 1, 1, 1, 1},
{1, 1, 1, 1, 0},
{1, 0,... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(4 * N * M), where N * M are the total cells, and for each cell, we check 4 adjacent nodes f

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Method to find the shortest path in a
      grid using BFS
    int shortestPath(vector<vector<int>>
      &grid, pair<int, int> source, pair<int,
      int> destination) {
        // Edge Case: if the source is the
          same as the destination
        if (source.first == destination.first
          && source.second ==
          destination.second)
            return 0;
        // Create a queue for BFS, storing
          cells with their distances from
          source
        queue<pair<int, pair<int, int>>> q;
        int n = grid.size();
        int m = grid[0].size();
        // Initialize the distance matrix,
          marking all cells as unvisited
          initially
        vector<vector<int>> dist(n,
          vector<int>(m, 1e9));
        dist[source.first][source.second] = 0;
        // Push the source cell into the queue
          with distance 0
        q.push({0, {source.first,
          source.second}});
        // Define the possible directions (up,
          right, down, left)
        int dr[] = {-1, 0, 1, 0};
        int dc[] = {0, 1, 0, -1};
        // Start BFS traversal
        while (!q.empty()) {
            auto it = q.front();
            q.pop();
            int dis = it.first;
            int r = it.second.first;
            int c = it.second.second;
            // Check all 4 adjacent directions
            for (int i = 0; i < 4; i++) {
                int newr = r + dr[i];
                int newc = c + dc[i];
                // Check if the new cell is
                  within bounds and is a valid
                  cell (i.e., 1)
                if (newr >= 0 && newr < n &&
                  newc >= 0 && newc < m &&
                  grid[newr][newc] == 1 && dis
                  + 1 < dist[newr][newc]) {
                    dist[newr][newc] = 1 +
                      dis;  // Update the
                      distance
                    // If destination is
                      reached, return the
                      distance
                    if (newr ==
                      destination.first &&
                      newc ==
                      destination.second)
                        return dis + 1;
                    // Add the new cell to the
                      queue for further
                      processing
                    q.push({1 + dis, {newr,
                      newc}});
                }
            }
        }
        // If no path is found from source to
          destination
        return -1;
    }
};
```

Q306. Path with minimum effort

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Example 1:
Input:
heights = [[1, 2, 2], [3, 8, 2], [5, 3, 5]]
Output: 2
Explanation: The route [1, 3, 5, 3, 5] has a maximum absolute difference of 2 in consecutive cells.
This is better than the route [1, 2, 2, 2, 5], where the maximum absolute difference is 3.

Example 2:
Input:
heights = [[1, 2, 1, 1, 1], [1, 2, 1, 2, 1], [1, 2, 1, 2, 1], [1, 1, 1, 2, 1]]
Output: 0
Explanation: The route [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] has a maximum absolute difference of 0 in c... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(4 * N * M * log(N * M)), where N * M are the total cells, for each of which we check 4 adja

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Method to calculate the minimum effort
      path using Dijkstra's algorithm
    int MinimumEffort(vector<vector<int>>
      &heights) {
        // Create a priority queue to store
          the cells and their respective
          distance from the source
        priority_queue<pair<int, pair<int,
          int>>,
                       vector<pair<int,
                         pair<int, int>>>,
                       greater<pair<int,
                         pair<int, int>>>>
            pq;
        int n = heights.size();  // Number of
          rows
        int m = heights[0].size();  // Number
          of columns
        // Create a distance matrix,
          initialized with a large value
          (unvisited)
        vector<vector<int>> dist(n,
          vector<int>(m, 1e9));
        dist[0][0] = 0;  // Distance for the
          source cell (0, 0) is 0
        pq.push({0, {0, 0}});  // Push source
          cell to the priority queue
        // Define the possible directions (up,
          right, down, left)
        int dr[] = {-1, 0, 1, 0};
        int dc[] = {0, 1, 0, -1};
        // Start the Dijkstra algorithm
        while (!pq.empty()) {
            auto it = pq.top();  // Get the
              cell with the minimum distance
            pq.pop();
            int diff = it.first;  // The
              current effort
            int row = it.second.first;
            int col = it.second.second;
            // If we reach the destination
              cell, return the current effort
            if (row == n - 1 && col == m - 1)
                return diff;
            // Check all 4 possible adjacent
              cells
            for (int i = 0; i < 4; i++) {
                int newr = row + dr[i];
                int newc = col + dc[i];
                // Check if the new cell is
                  within bounds
                if (newr >= 0 && newc >= 0 &&
                  newr < n && newc < m) {
                    // Calculate the effort
                      required to move to the
                      new cell
                    int newEffort =
                      max(abs(heights[row][col
                      ] -
                      heights[newr][newc]),
                      diff);
                    // If the calculated
                      effort is less, update
                      and push to the queue
                    if (newEffort <
                      dist[newr][newc]) {
                        dist[newr][newc] =
                          newEffort;
                        pq.push({newEffort,
                          {newr, newc}});
                    }
                }
            }
        }
        return 0;  // If unreachable (although
          it should not reach here)
    }
};
```

Q307. Cheapest flight within K stops

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Example 1:
Input: n = 4
flights = [[0, 1, 100], [1, 2, 100], [2, 0, 100], [1, 3, 600], [2, 3, 200]]
src = 0
dst = 3
k = 1
Output: 700
Explanation: The optimal path with at most 1 stop from city 0 to 3 is through cities [0, 1, 3] with a total cost of 100 + 600 = 700.
Note that the path [0, 1, 2, 3] is cheaper but is invalid because it uses 2 stops.

Example 2:
Input: n = 3
flights = [[0, 1, 100], [1, 2, 100], [0, 2, 500]]
src = 0
dst = 2
k = 1
Output: 200
Explanati... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where the additional log(N) time is eliminated by using a simple queue rather than a pr

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution
{
public:
    // Method to find the cheapest flight
      within K stops using BFS
    int CheapestFLight(int n,
      vector<vector<int>> &flights,
                       int src, int dst, int
                         K)
    {
        // Create the adjacency list to
          represent airports and flights as a
          graph
        vector<pair<int, int>> adj[n];
        for (auto it : flights)
        {
            adj[it[0]].push_back({it[1],
              it[2]});  // Add an edge from
              source to destination with the
              weight (cost)
        }
        // Create a queue to store the node,
          its distance from the source, and
          the number of stops
        queue<pair<int, pair<int, int>>> q;
        q.push({0, {src, 0}});  // Push the
          source node with 0 stops and 0 cost
        // Create a distance array to store
          the minimum cost to reach each node
        vector<int> dist(n, 1e9);
        dist[src] = 0;  // The distance from
          source to itself is 0
        // BFS traversal with a queue to
          process the nodes
        while (!q.empty())
        {
            auto it = q.front();
            q.pop();
            int stops = it.first;  // Number
              of stops so far
            int node = it.second.first;  //
              Current node
            int cost = it.second.second;  //
              Cost to reach the current node
            // If the number of stops exceeds
              K, continue to the next
              iteration
            if (stops > K)
                continue;
            // Iterate over all the adjacent
              nodes (next destinations)
            for (auto iter : adj[node])
            {
                int adjNode = iter.first;  //
                  Next destination node
                int edW = iter.second;  //
                  Cost of the flight to the
                  next destination
                // If a shorter path to the
                  adjacent node is found,
                  update the distance
                if (cost + edW < dist[adjNode]
                  && stops <= K)
                {
                    dist[adjNode] = cost +
                      edW;  // Update the
                      distance
                    q.push({stops + 1,
                      {adjNode, cost + edW}});
                      // Push the new node
                      with updated stops and
                      cost
                }
            }
        }
        // If destination node is unreachable,
          return -1
        if (dist[dst] == 1e9)
            return -1;
        return dist[dst];  // Return the
          minimum cost to reach the
          destination
    }
};
```

Q308. Network Delay Time

MEDIUM

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Input: times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2
Output: 2
Explanation: 2 -> 1 (1 unit) and 2 -> 3 -> 4 (2 unit)
The last node (4) gets the signal at time 2.

Input: times = [[1,2,1]], n = 2, k = 1
Output: 1
Explanation: Direct edge 1 -> 2 delivers signal in 1 unit of time.

[OPTIMAL SOLUTION] Approach — Time Complexity: O((E + V) * log V), Each edge is relaxed atmost once. Priority queue operations take log V ti

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find minimum time for every
      node to receive signal
    int networkDelayTime(vector<vector<int>>&
      times, int n, int k) {
        // Create adjacency list
        vector<vector<pair<int, int>>> adj(n +
          1);
        for (auto& time : times) {
            int u = time[0], v = time[1], w =
              time[2];
            adj[u].push_back({v, w});
        }

        // Initialize min-heap
        priority_queue<pair<int, int>,
          vector<pair<int, int>>, greater<>>
          pq;
        pq.push({0, k});

        // Initialize distance array
        vector<int> dist(n + 1, INT_MAX);
        dist[k] = 0;

        // Process nodes
        while (!pq.empty()) {
            // Get the node with the smallest
              time
            int time = pq.top().first;
            int node = pq.top().second;
            pq.pop();

            // Traverse all neighbors of the
              current node
            for (auto& [nbr, wt] : adj[node])
              {
                // If shorter path is found to
                  neighbor
                if (dist[nbr] > time + wt) {
                    // Update distance
                    dist[nbr] = time + wt;
                    // Push updated time to
                      the priority queue
                    pq.push({dist[nbr], nbr});
                }
            }
        }

        // Get maximum time
        int ans = *max_element(dist.begin() +
          1, dist.end());
        return ans == INT_MAX ? -1 : ans;
    }
};
```

Q309. Number of ways to arrive at destination

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Example 1:
Input: n = 7, m = 10
edges = [[0, 6, 7], [0, 1, 2], [1, 2, 3], [1, 3, 3], [6, 3, 3], [3, 5, 1], [6, 5, 1], [2, 5, 1], [0, 4, 5], [4, 6, 2]]
Output: 4
Explanation: The four ways to get from city 0 to city 6 in 7 minutes (the shortest time) are:
- 0 → 6
- 0 → 4 → 6
- 0 → 1 → 2 → 5 → 6
- 0 → 1 → 3 → 5 → 6

Example 2:
Input: n = 6, m = 8
edges = [[0, 5, 8], [0, 2, 2], [0, 1, 1], [1, 3, 3], [1, 2, 3], [2, 5, 6], [3, 4, 2], [4, 5, 2]]
Output: 3
Explanation: The... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(E * log(V)), as we are using simple Dijkstra's algorithm here, the time complexity will be

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    int CheapestFLight(int n,
      vector<vector<int>> &flights, int src,
      int dst, int K) {
        // Create the adjacency list to
          represent airports and flights as a
          graph
        vector<pair<int, int>> adj[n];
        for (auto it : flights) {
            adj[it[0]].push_back({it[1],
              it[2]});
            adj[it[1]].push_back({it[0],
              it[2]});
        }
        // Create a priority queue (min heap)
          for Dijkstra's algorithm
        priority_queue<pair<int, int>,
                       vector<pair<int,
                         int>>,
                       greater<pair<int,
                       int>>> pq;
        // Initialize the distance array and
          ways array
        vector<int> dist(n, INT_MAX), ways(n,
          0);
        // Distance for the source node is 0
        dist[src] = 0;
        // There's 1 way to reach the source
          (itself)
        ways[src] = 1;
        // Push the source node with distance
          0
        pq.push({0, src});
        // Define the modulo value for large
          numbers
        int mod = (int)(1e9 + 7);
        // Perform Dijkstra's algorithm
        while (!pq.empty()) {
            int dis = pq.top().first;
            int node = pq.top().second;
            pq.pop();
            // Iterate through adjacent nodes
            for (auto it : adj[node]) {
                // Next destination node
                int adjNode = it.first;
                // Cost of the flight to the
                  next destination
                int edW = it.second;
                // If a shorter path is found,
                  update the distance and
                  number of ways
                if (dis + edW < dist[adjNode])
                  {
                    // Update the distance
                    dist[adjNode] = dis + edW;
                    // Push the new node with
                      updated distance
                    pq.push({dis + edW,
                      adjNode});
                    // Copy the number of ways
                      to the new node
                    ways[adjNode] =
                      ways[node];
                }
                // If the same shortest path
                  is found, update the number
                  of ways
                else if (dis + edW ==
                  dist[adjNode]) {
                    // Increment the number of
                      ways
                    ways[adjNode] =
                      (ways[adjNode] +
                      ways[node]) % mod;
                }
            }
        }
        // Return the number of ways to reach
          the last node modulo 10^9 + 7
        return ways[dst] % mod;
    }
};
```

Q310. Minimum multiplications to reach end

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Example 1:
Input:
arr[] = {2, 5, 7}
start = 3
end = 30
Output:
2
Explanation:
Step 1: 3*2 = 6 % 100000 = 6
Step 2: 6*5 = 30 % 100000 = 30
Therefore, in minimum 2 multiplications we reach the end number which is treated as a destination node of a graph here.

Example 2:
Input:
arr[] = {3, 4, 65}
start = 7
end = 66175
Output:
4
Explanation:
Step 1: 7*3 = 21 % 100000 = 21
Step 2: 21*3 = 6 % 100000 = 63
Step 3: 63*65 = 4095 % 100000 = 4095
Step 4: 4095*65 = 266175 % 100000 = 66175
Therefore, i... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity : O(100000 * N), Where '100000' are the total possible numbers generated by multiplication (hy

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int minimumMultiplications(vector<int>
      &arr,
                               int start, int
                                 end)
    {
        // Create a queue for storing the
          numbers as a result of
          multiplication
        // of the numbers in the array and the
          start number.
        queue<pair<int, int>> q;
        q.push({start, 0});
        // Create a dist array to store the
          no. of multiplications to reach
        // a particular number from the start
          number.
        vector<int> dist(100000, 1e9);
        dist[start] = 0;
        int mod = 100000;
        // Multiply the start no. with each of
          numbers in the arr
        // until we get the end no.
        while (!q.empty())
        {
            int node = q.front().first;
            int steps = q.front().second;
            q.pop();
            for (auto it : arr)
            {
                int num = (it * node) % mod;
                // If the no. of
                  multiplications are less
                  than before
                // in order to reach a number,
                  we update the dist array.
                if (steps + 1 < dist[num])
                {
                    dist[num] = steps + 1;
                    // Whenever we reach the
                      end number
                    // return the calculated
                      steps
                    if (num == end)
                        return steps + 1;
                    q.push({num, steps + 1});
                }
            }
        }
        // If the end no. is unattainable.
        return -1;
    }
};
```

Q311. Bellman Ford Algorithm

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Example 1:
Input Format:
V = 6,
E = [[3, 2, 6], [5, 3, 1], [0, 1, 5], [1, 5, -3], [1, 2, -2], [3, 4, -2], [2, 4, 3]],
S = 0

Result: 0 5 3 3 1 2

Example 2:
Input Format:V = 2, E = [[0,1,9]], S = 0

Result: 0 9
Explanation: Shortest distance of all nodes from the source node is returned.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V*E), where V = no. of vertices and E = no. of Edges. Space Complexity:

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    /*  Function to implement Bellman Ford
    *   edges: vector of vectors which represents the graph
    *   S: source vertex to start traversing graph with
    *   V: number of vertices
    */
    vector<int> bellman_ford(int V, vector<vector<int>>& edges, int S) {
        vector<int> dist(V, 1e8);
        dist[S] = 0;
        for (int i = 0; i < V - 1; i++) {
            for (auto it : edges) {
                int u = it[0]; //Starting point of the edge
                int v = it[1]; //Ending point of the edge
                int wt = it[2]; //Edge weight
                if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {
                    dist[v] = dist[u] + wt;
                }
            }
        }
        // Nth relaxation to check negative cycle
        for (auto it : edges) {
            int u = it[0];
            int v = it[1];
            int wt = it[2];
            if (dist[u] != 1e8 && dist[u] + wt < dist[v]) {
                return { -1};
            }
        }

        return dist;
    }
};
```

Q312. Floyd warshall algorithm

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Example 1:
Input: matrix = [[0, 2, -1, -1], [1, 0, 3, -1], [-1, -1, 0, 1], [3, 5, 4, 0]]
Output: [[0, 2, 5, 6], [1, 0, 3, 4], [4, 6, 0, 1], [3, 5, 4, 0]]
Explanation: The matrix stores the shortest distances between vertices. After applying the algorithm (like Floyd-Warshall), the shortest distances between all pairs of vertices are computed. The updated matrix shows the shortest distances, e.g., distance from vertex 0 to vertex 2 is 5, from vertex 1 to vertex 3 is 4, and so on.

Example 2... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V^3), where V is the number of vertices in the graph. This is because we have three nested

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    /* Function to find the shortest distance
    between every pair of vertices. */
    void shortest_distance(vector<vector<int>>
      &matrix){

        // Getting the number of nodes
    int n = matrix.size();

    // For each intermediate node k
    for(int k=0; k<n; k++) {

                // Check for every (i, j) pair
            of nodes
        for(int i=0; i<n; i++) {
            for(int j=0; j<n; j++) {

                        /* If k is not an
                    intermediate
                node, skip the iteration */
                if(matrix[i][k] == -1 ||
                   matrix[k][j] == -1)
                        continue;
                /* If no direct edge from
                i to v is present */
                if(matrix[i][j] == -1) {

                    // Update the distance
                    matrix[i][j] =
                                matrix[i][k] +
                          matrix[k][j];
                }

                            /* Else update the
                      distance to
                minimum of both paths */
                else {
                    matrix[i][j] =
                          min(matrix[i][j] ,
matrix[i][k] + matrix[k][j]
                                );
                }
            }
        }
    }
}
};
```

Q313. Find the city with the smallest number of neighbors

HARD

Topic: Graphs [Concepts & Problems] > Shortest Path Algorithms and Problems

Practice Link

Example 1:
Input: N=4, M=4,
edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]],
distanceThreshold = 4
Result: 3
Explanation: The adjacent cities for each city at a distanceThreshold are:
City 0 → [City 1, City 2]
City 1 → [City 0, City 2, City 3]
City 2 → [City 0, City 1, City 3]
City 3 → [City 1, City 2]
Here, City 0 and City 3 have a minimum number of cities (2) within the distanceThreshold. The city with the largest number is City 3. So, the result is City 3.

Example 2:
Input:... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V3) , as we have three nested loops each running for V times, where V = number of vertices.

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Method to find the city with the least
      reachable cities within the threshold
      distance
    int findCity(int n, int m,
      vector<vector<int>>& edges, int
      distanceThreshold) {
        // Initialize the distance matrix with
          a large value (INT_MAX)
        vector<vector<int>> dist(n,
          vector<int> (n, INT_MAX));
        // Set the distance for each edge
        for (auto it : edges) {
            dist[it[0]][it[1]] = it[2];  //
              Set the distance from city1 to
              city2
            dist[it[1]][it[0]] = it[2];  //
              Set the distance from city2 to
              city1
        }
        // Set the diagonal to 0, as the
          distance from a city to itself is 0
        for (int i = 0; i < n; i++) dist[i][i]
          = 0;
        // Apply Floyd-Warshall Algorithm to
          find the shortest paths between all
          pairs of cities
        for (int k = 0; k < n; k++) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    // Update the distance if
                      a shorter path is found
                    if (dist[i][k] == INT_MAX
                      || dist[k][j] ==
                      INT_MAX)
                        continue;
                    dist[i][j] =
                      min(dist[i][j],
                      dist[i][k] +
                      dist[k][j]);
                }
            }
        }
        // Initialize variables to track the
          city with the least reachable cities
        int cntCity = n;
        int cityNo = -1;
        // Check each city and count the
          number of cities within the
          threshold distance
        for (int city = 0; city < n; city++) {
            int cnt = 0;
            for (int adjCity = 0; adjCity < n;
              adjCity++) {
                // If the distance to the
                  adjacent city is within the
                  threshold, increment count
                if (dist[city][adjCity] <=
                  distanceThreshold)
                    cnt++;
            }
            // Update the city with the least
              number of reachable cities
            if (cnt <= cntCity) {
                cntCity = cnt;
                cityNo = city;
            }
        }
        return cityNo;
    }
};
```

Q314. Prim's Algorithm

HARD

Topic: Graphs [Concepts & Problems] > MinimumSpanningTree/Disjoint Set and Problems

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```
Input Format:
V = 5, edges = { {0, 1, 2}, {0, 2, 1}, {1, 2,
  1}, {2, 3, 2}, {3, 4, 1}, {4, 2, 2}}

Result: 5
Explanation:
The minimum spanning tree is drawn below:

MST = {(0, 2), (1, 2), (2, 3), (3, 4)}
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```
Input Format:
V = 5, edges = { {0, 1, 2}, {0, 3, 6}, {1, 2,
  3}, {1, 3, 8}, {1, 4, 5}, {4, 2, 7}}

Result: 16
Explanation:
The minimum spanning tree for the given graph
  is drawn below:
MST = {(0, 1), (0, 3), (1, 2), (1, 4)}
```

Q315. Disjoint Set

HARD

Topic: Graphs [Concepts & Problems] > MinimumSpanningTree/Disjoint Set and Problems

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```java
import java.io.*;
import java.util.*;
class DisjointSet {
    List<Integer> rank = new ArrayList<>();
    List<Integer> parent = new ArrayList<>();
    public DisjointSet(int n) {
        for (int i = 0; i <= n; i++) {
            rank.add(0);
            parent.add(i);
        }
    }
    public int findUPar(int node) {
        if (node == parent.get(node)) {
            return node;
        }
        int ulp = findUPar(parent.get(node));
        parent.set(node, ulp);
        return parent.get(node);
    }
    public void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (rank.get(ulp_u) < rank.get(ulp_v))
          {
            parent.set(ulp_u, ulp_v);
        } else if (rank.get(ulp_v) <
          rank.get(ulp_u)) {
            parent.set(ulp_v, ulp_u);
        } else {
            parent.set(ulp_v, ulp_u);
            int rankU = rank.get(ulp_u);
            rank.set(ulp_u, rankU + 1);
        }
    }
}
class Main {
    public static void main (String[] args) {
        DisjointSet ds = new DisjointSet(7);
        ds.unionByRank(1, 2);
        ds.unionByRank(2, 3);
        ds.unionByRank(4, 5);
        ds.unionByRank(6, 7);
        ds.unionByRank(5, 6);
        // if 3 and 7 same or not
        if (ds.findUPar(3) == ds.findUPar(7))
          {
            System.out.println("Same");
        } else
            System.out.println("Not Same");
        ds.unionByRank(3, 7);
        if (ds.findUPar(3) == ds.findUPar(7))
          {
            System.out.println("Same");
        } else
            System.out.println("Not Same");
    }
}
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```cpp
#include <bits/stdc++.h>
using namespace std;
class DisjointSet {
    vector<int> rank, parent;
public:
    DisjointSet(int n) {
        rank.resize(n + 1, 0);
        parent.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            parent[i] = i;
        }
    }

    int findUPar(int node) {
        if (node == parent[node])
            return node;
        return parent[node] =
          findUPar(parent[node]);
    }

    void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (rank[ulp_u] < rank[ulp_v]) {
            parent[ulp_u] = ulp_v;
        }
        else if (rank[ulp_v] < rank[ulp_u]) {
            parent[ulp_v] = ulp_u;
        }
        else {
            parent[ulp_v] = ulp_u;
            rank[ulp_u]++;
        }
    }
};
```

Q316. Find the MST weight

HARD

Topic: Graphs [Concepts & Problems] > MinimumSpanningTree/Disjoint Set and Problems

Practice Link

Refer to the code implementation and logic below.

[BETTER] Snippet 2

Complexity: *N/A (Conceptual Tutorial)*

```
Input Format:
V = 5, edges = { {0, 1, 2}, {0, 2, 1}, {1, 2,
  1}, {2, 3, 2}, {3, 4, 1}, {4, 2, 2}}

Result: 5
Explanation:
The minimum spanning tree is drawn below:

MST = {(0, 2), (1, 2), (2, 3), (3, 4)}
```

[OPTIMAL] Snippet 1

Complexity: *N/A (Conceptual Tutorial)*

```
Input Format:
V = 5, edges = { {0, 1, 2}, {0, 3, 6}, {1, 2,
  3}, {1, 3, 8}, {1, 4, 5}, {4, 2, 7}}

Result: 16
Explanation:
The minimum spanning tree for the given graph
  is drawn below:
MST = {(0, 1), (0, 3), (1, 2), (1, 4)}
```

Q317. Number of operations to make network connected

HARD

Topic: Graphs [Concepts & Problems] > MinimumSpanningTree/Disjoint Set and Problems

Practice Link

Input: N = 4, M = 3, Edge[] =[[0, 1], [0, 2], [1, 2]]

Output: 1
Explanation: We need a minimum of 1 operation to make the two components connected. We can remove the edge (1,2) and add the edge between node 2 and node 3 like the following:

Input: N = 9, M = 8, Edge[] = [[0,1],[0,2],[0,3],[1,2],[2,3],[4,5],[5,6],[7,8]]

Output: 1
Explanation : We need a minimum of 2 operations to make the two components connected. We can remove the edge (0,2) and add the edge between node 3 and node 4 an... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N + M × α(N)), we visit every node and edge once to find number of components.

```cpp
#include <bits/stdc++.h>
using namespace std;
class DSU {
public:
    vector<int> parent, rank;
    DSU(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for(int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }
    int find(int x) {
        if(parent[x] != x)
            parent[x] = find(parent[x]);
        return parent[x];
    }
    void unite(int x, int y) {
        int px = find(x);
        int py = find(y);
        if(px == py) return;
        if(rank[px] < rank[py])
            parent[px] = py;
        else if(rank[px] > rank[py])
            parent[py] = px;
        else {
            parent[py] = px;
            rank[px]++;
        }
    }
};
class Solution {
public:
    int makeConnected(int n,
      vector<vector<int>>& connections) {
        if(connections.size() < n - 1) return
          -1;
        DSU dsu(n);
        for(auto& edge : connections) {
            dsu.unite(edge[0], edge[1]);
        }
        unordered_set<int> components;
        for(int i = 0; i < n; i++) {
            components.insert(dsu.find(i));
        }
        return components.size() - 1;
    }
};
```

Q318. Most stones removed with same row or column

MEDIUM

Topic: Graphs [Concepts & Problems] > MinimumSpanningTree/Disjoint Set and Problems

Practice Link

Input: n=6 stones = [[0, 0],[0, 1], [1, 0],[1, 2],[2, 1],[2, 2]]

Output: 5
Explanation: One of the many ways to remove 5 stones is to remove the following stones: [0,0], [1,0], [0,1], [2,1], [1,2]

Input: N = 6, stones = [[0, 0], [0, 2], [1, 3], [3, 1], [3, 2], [4, 3]]

Output: 4
Explanation : We can remove the following stones: [0,0], [0,2], [1,3], [3,1]

[OPTIMAL SOLUTION] Approach

```cpp
#include <bits/stdc++.h>
using namespace std;

class DSU {
public:
    unordered_map<int, int> parent;

    // Function to find the ultimate parent of
      node x
    int find(int x) {
        // If x is not present, make it its
          own parent
        if (parent.find(x) == parent.end())
            parent[x] = x;

        // Path compression to optimize future
          finds
        if (x != parent[x])
            parent[x] = find(parent[x]);

        return parent[x];
    }

    // Function to connect two nodes
    void unite(int x, int y) {
        // Set parent of x's root to y's root
        parent[find(x)] = find(y);
    }
};

class Solution {
public:
    int removeStones(vector<vector<int>>&
      stones) {
        DSU dsu;

        // Connect each stone's row and column
          (offset to avoid collision)
        for (auto& stone : stones) {
            dsu.unite(stone[0], stone[1] +
              10001);
        }

        unordered_set<int> components;

        // Count unique connected components
        for (auto& stone : stones) {
components.insert(dsu.find(stone[0]));
        }

        // Max stones removed = total stones -
          number of components
        return stones.size() -
          components.size();
    }
};
```

Q319. Accounts merge

HARD

Topic: Graphs [Concepts & Problems] > MinimumSpanningTree/Disjoint Set and Problems

Practice Link

Example 1:
Input: N = 4
accounts [ ] =
[["John","johnsmith@mail.com","john_newyork@mail.com"],
["John","johnsmith@mail.com","john00@mail.com"],
"Mary","mary@mail.com"]
["John","johnnybravo@mail.com"]]
Output: [["John","john00@mail.com","john_newyork@mail.com",... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N+E) + O(E*4■) + O(N*(ElogE + E)) where N = no. of indices or nodes and E = no. of emails.

```cpp
#include <bits/stdc++.h>
using namespace std;
class DisjointSet {
    vector<int> rank, parent, size;
public:
    DisjointSet(int n) {
        rank.resize(n + 1, 0);
        parent.resize(n + 1);
        size.resize(n + 1);
        for (int i = 0; i <= n; i++)
          {
            parent[i] = i;
            size[i] = 1;
        }
    }
    int findUPar(int node) {
        if (node == parent[node])
            return node;
        parent[node] =
          findUPar(parent[node]);
        return parent[node];
    }
    void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (rank[ulp_u] <
          rank[ulp_v]) {
            parent[ulp_u] = ulp_v;
        }
        else if (rank[ulp_v] <
          rank[ulp_u]) {
            parent[ulp_v] = ulp_u;
        }
        else {
            parent[ulp_v] = ulp_u;
            rank[ulp_u]++;
        }
    }
    void unionBySize(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (size[ulp_u] <
          size[ulp_v]) {
            parent[ulp_u] = ulp_v;
            size[ulp_v] +=
              size[ulp_u];
        }
        else {
            parent[ulp_v] = ulp_u;
            size[ulp_u] +=
              size[ulp_v];
        }
    }
};
class Solution {
public:
    vector<vector<string>>
      accountsMerge(
        vector<vector<string>>
          &details)
    {
        int n = details.size();
        DisjointSet ds(n);
        unordered_map<string, int>
          mapMailNode;
        for (int i = 0; i < n; i++)
          {
            for (int j = 1; j <
              details[i].size();
              j++) {
                string mail =
                  details[i][j];
                if
                  (mapMailNode.find(
                  mail) ==
                  mapMailNode.end())
                  {
mapMailNode[mail] = i;
                }
                else {
ds.unionBySize(i,
                    mapMailNode[ma
                    il]);
                }
            }
        }
        vector<string>
          mergedMail[n];
        for (auto it : mapMailNode)
          {
            string mail = it.first;
            int node =
              ds.findUPar(it.second)
              ;
mergedMail[node].push_back(mail);
        }
        vector<vector<string>> ans;
        for (int i = 0; i < n; i++)
          {
            if
              (mergedMail[i].empty()
              ) continue;
sort(mergedMail[i].begin(),
              mergedMail[i].end());
            vector<string> temp;
temp.push_back(details[i][0]);
            for (auto &mail :
              mergedMail[i]) {
temp.push_back(mail);
            }
            ans.push_back(temp);
        }
        sort(ans.begin(),
          ans.end());
        return ans;
    }
};
```

Q320. Number of islands II

HARD

Topic: Graphs [Concepts & Problems] > MinimumSpanningTree/Disjoint Set and Problems

Practice Link

Example 1:
Input Format: n = 4 m = 5 k = 4 A = {{1,1},{0,1},{3,3},{3,4}}
Output: 1 1 2 2
Explanation: The following illustration is the representation of the operation:

Example 2:
Input Format: n = 4 m = 5 k = 12 A = {{0,0},{0,0},{1,1},{1,0},{0,1},{0,3},{1,... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N × M), Each cell is visited at most once in BFS/DFS traversal.

```
#include <bits/stdc++.h>
using namespace std;
class DisjointSet {
    vector<int> rank, parent, size;
public:
    DisjointSet(int n) {
        rank.resize(n + 1, 0);
        parent.resize(n + 1);
        size.resize(n + 1, 1);
        for (int i = 0; i <= n; i++)
          {
            parent[i] = i;
        }
    }
    int findUPar(int node) {
        if (node == parent[node])
            return node;
        return parent[node] =
          findUPar(parent[node]);
    }
    void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (rank[ulp_u] <
          rank[ulp_v]) {
            parent[ulp_u] = ulp_v;
        } else if (rank[ulp_v] <
          rank[ulp_u]) {
            parent[ulp_v] = ulp_u;
        } else {
            parent[ulp_v] = ulp_u;
            rank[ulp_u]++;
        }
    }
    void unionBySize(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (size[ulp_u] <
          size[ulp_v]) {
            parent[ulp_u] = ulp_v;
            size[ulp_v] +=
              size[ulp_u];
        } else {
            parent[ulp_v] = ulp_u;
            size[ulp_u] +=
              size[ulp_v];
        }
    }
};
class Solution {
private:
    bool isValid(int adjr, int adjc,
      int n, int m) {
        return adjr >= 0 && adjr < n
          && adjc >= 0 && adjc < m;
    }
public:
    vector<int> numOfIslands(int n,
      int m, vector<vector<int>>&
      operators) {
        DisjointSet ds(n * m);
        int vis[n][m];
        memset(vis, 0, sizeof vis);
        int cnt = 0;
        vector<int> ans;
        for (auto it : operators) {
            int row = it[0], col =
              it[1];
            if (vis[row][col] == 1)
              {
                ans.push_back(cnt);
                continue;
            }
            vis[row][col] = 1;
            cnt++;
            int dr[] = {-1, 0, 1,
              0};
            int dc[] = {0, 1, 0,
              -1};
            for (int ind = 0; ind <
              4; ind++) {
                int adjr = row +
                  dr[ind], adjc =
                  col + dc[ind];
                if (isValid(adjr,
                  adjc, n, m)) {
                    if
                      (vis[adjr][adj
                      c] == 1) {
                        int nodeNo =
                          row * m +
                          col;
                        int
                          adjNodeNo
                          = adjr * m
                          + adjc;
                        if
                          (ds.findUP
                          ar(nodeNo)
                          != ds.find
                          UPar(adjNo
                          deNo)) {
                            cnt--;
ds.unionBySize(nodeNo, adjNodeNo);
                        }
                    }
                }
            }
            ans.push_back(cnt);
        }
        return ans;
    }
};
```

Q321. Making a large island

HARD

Topic: Graphs [Concepts & Problems] > MinimumSpanningTree/Disjoint Set and Problems

Practice Link

Example 1:
Input: grid = [[1,0], [0,1]]
Output: 3
Explanation: We change one of the 0s to 1, connecting the two existing 1s, which results in an island with a maximum area of 3.

Example 2:
Input: grid = [[1, 1], [1, 1]]
Output: 4
Explanation: The... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N2) using nested loops, and within the loops, all the operations take constant time. Space

```cpp
#include <bits/stdc++.h>
using namespace std;
class DisjointSet {
public:
    vector<int> rank, parent, size;
    DisjointSet(int n) {
        rank.resize(n + 1, 0);
        parent.resize(n + 1);
        size.resize(n + 1);
        for (int i = 0; i <= n; i++)
          {
            parent[i] = i;
            size[i] = 1;
        }
    }
    int findUPar(int node) {
        if (node == parent[node])
            return node;
        return parent[node] =
          findUPar(parent[node]);
    }
    void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (rank[ulp_u] <
          rank[ulp_v]) {
            parent[ulp_u] = ulp_v;
        }
        else if (rank[ulp_v] <
          rank[ulp_u]) {
            parent[ulp_v] = ulp_u;
        }
        else {
            parent[ulp_v] = ulp_u;
            rank[ulp_u]++;
        }
    }
    void unionBySize(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (size[ulp_u] <
          size[ulp_v]) {
            parent[ulp_u] = ulp_v;
            size[ulp_v] +=
              size[ulp_u];
        }
        else {
            parent[ulp_v] = ulp_u;
            size[ulp_u] +=
              size[ulp_v];
        }
    }
};
class Solution{
private:
    vector<int> delRow = {-1, 0, 1,
      0};
    vector<int> delCol = {0, 1, 0,
      -1};
    bool isValid(int &i, int &j, int
      &n) {
        if(i < 0 || i >= n) return
          false;
        if(j < 0 || j >= n) return
          false;
        return true;
    }
    void addInitialIslands(vector<ve
      ctor<int>> grid,
DisjointSet &ds, int n) {
        for (int row = 0; row < n ;
          row++) {
            for (int col = 0; col <
              n ; col++) {
                if (grid[row][col]
                  == 0) continue;
                for (int ind = 0;
                  ind < 4; ind++) {
                    int newRow = row
                      + delRow[ind];
                    int newCol = col
                      + delCol[ind];
                    if
                      (isValid(newRo
                      w, newCol, n)
                      &&
grid[newRow][newCol] == 1) {
                        int nodeNo =
                          row * n +
                          col;
                        int
                          adjNodeNo
                          = newRow *
                          n +
                          newCol;
ds.unionBySize(nodeNo, adjNodeNo);
                    }
                }
            }
        }
public:
    int
      largestIsland(vector<vector<in
      t>>& grid) {
        int n = grid.size();
        DisjointSet ds(n*n);
        addInitialIslands(grid, ds,
          n);
        int ans = 0;
        for (int row = 0; row < n;
          row++) {
            for (int col = 0; col <
              n; col++) {
                if (grid[row][col]
                  == 1) continue;
                set<int> components;
                for (int ind = 0;
                  ind < 4; ind++) {
                    int newRow = row
                      + delRow[ind];
                    int newCol = col
                      + delCol[ind];
                    if
                      (isValid(newRo
                      w, newCol, n)
                      &&
grid[newRow][newCol] == 1) {
                        int
                          nodeNumber
                          = newRow *
                          n +
                          newCol;
                        components.i
                          nsert(ds.f
                          indUPar(no
                          deNumber))
                          ;
                    }
                }
                int sizeTotal = 0;
                for (auto it :
                  components) {
                    sizeTotal +=
                      ds.size[it];
                }
                ans = max(ans,
                  sizeTotal + 1);
            }
        }
        for (int cellNo = 0; cellNo
          < n * n; cellNo++) {
            ans = max(ans,
              ds.size[ds.findUPar(ce
              llNo)]);
        }
        return ans;
    }
};
```

Q322. Swim in Rising Water

MEDIUM

Topic: Graphs [Concepts & Problems] > MinimumSpanningTree/Disjoint Set and Problems

Practice Link

Input: grid = [[0,2],[1,3]]
Output: 3
Explanation:
t = 0, only (0,0) is reachable.
t = 1, cells {0,0},{1,0} reachable, but (1,1) still blocked (elevation 3).
t = 2, cells {0,0},{0,1},{1,0} reachable.
t = 3, all cells ≤ 3 are flooded, so we can swim to (1,1). Minimum t = 3.

Input: grid = [[0, 1, 2, 3, 4],[24,23,22,21, 5],[12,13,14,15,16],[11,17,18,19,20],[10, 9, 8, 7, 6]]
Output: 16
Explanation: The earliest moment the start and end cells belong to the same flooded component is t = 16.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N^2 * log(N)), Each cell is inserted into minHeap atmost once. Priority queue operations ta

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Minimum time to reach bottom right cell
    int swimInWater(vector<vector<int>>& grid)
      {
        // Get grid size
        int n = grid.size();

        // Create a min-heap for cells based
          on elevation
        priority_queue<vector<int>,
          vector<vector<int>>,
          greater<vector<int>>> minHeap;

        // Create visited matrix
        vector<vector<int>> visited(n,
          vector<int>(n, 0));

        // Push starting cell to heap
        minHeap.push({grid[0][0], 0, 0});
        visited[0][0] = 1;

        // Direction vectors for movement
        vector<pair<int, int>> dirs = {{0,1},
          {1,0}, {0,-1}, {-1,0}};

        // Process heap until destination is
          reached
        while (!minHeap.empty()) {
            // Extract cell with least
              elevation
            auto curr = minHeap.top();
              minHeap.pop();
            int elevation = curr[0], r =
              curr[1], c = curr[2];

            // If destination reached, return
              elevation
            if (r == n - 1 && c == n - 1)
              return elevation;

            // Explore neighboring cells
            for (auto& dir : dirs) {
                int nr = r + dir.first;
                int nc = c + dir.second;

                // Check bounds and if not
                  visited
                if (nr >= 0 && nc >= 0 && nr <
                  n && nc < n &&
                  !visited[nr][nc]) {
                    visited[nr][nc] = 1;
minHeap.push({max(elevation, grid[nr][nc]),
                      nr, nc});
                }
            }
        }
        return -1;
    }
};
```

Q323. Bridges in graph

HARD

Topic: Graphs [Concepts & Problems] > Other Algorithms

Practice Link

Example 1:
Input Format: N = 4, connections = [[0,1],[1,2],[2,0],[1,3]]
Result: [[1, 3]]
Explanation: The edge [1, 3] is the critical edge because if we remove the edge the graph will be divided into 2 components.

Example 2:
Input Format N = 12, connection=[[1,2], [1,4], [2,3], [3, 4], [4, 5], [5, 6], [6, 7], [7, 8], [6, 9], [8, 9], [8, 10], [10, 11], [10, 12]]
Result: [[4, 5], [5, 6], [8, 10]]
Explanation: If we remove any of the three edges, the graph will be divided into 2 or more component... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V+2E), where V = no. of vertices, E = no. of edges. It is because the algorithm is just a s

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    int timer = 1; // Global timer to assign
      discovery times
    // Depth-First Search (DFS) to find
      bridges
    void dfs(int node, int parent, vector<int>
      &vis,
             vector<int> adj[], int tin[], int
               low[],
             vector<vector<int>> &bridges) {
        vis[node] = 1;                  // Mark
          current node as visited
        tin[node] = low[node] = timer; // Set
          discovery time and low-link value
        timer++;
        for (auto it : adj[node]) {    //
          Explore all adjacent nodes
            if (it == parent) continue; //
              Skip the edge to parent
            if (vis[it] == 0) {
                // If neighbor is unvisited,
                  recurse
                dfs(it, node, vis, adj, tin,
                  low, bridges);
                // Update low-link value of
                  current node
                low[node] = min(low[node],
                  low[it]);
                // Check if the edge is a
                  bridge
                if (low[it] > tin[node]) {
                    bridges.push_back({it,
                      node});
                }
            } else {
                // Back edge: update low-link
                  value
                low[node] = min(low[node],
                  low[it]);
            }
        }
    }
public:
    vector<vector<int>>
      criticalConnections(int n,
        vector<vector<int>>& connections) {
        // Step 1: Build adjacency list
        vector<int> adj[n];
        for (auto it : connections) {
            int u = it[0], v = it[1];
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        // Step 2: Initialize helper arrays
        vector<int> vis(n, 0);
        int tin[n]; // Discovery time
        int low[n]; // Lowest reachable time
        vector<vector<int>> bridges;
        // Step 3: Run DFS (assuming graph is
          connected)
        dfs(0, -1, vis, adj, tin, low,
          bridges);
        return bridges;
    }
};
```

Q324. Articulation point in graph

HARD

Topic: Graphs [Concepts & Problems] > Other Algorithms

Practice Link

Example-1
Result-{0,2}
Explanation: If we remove node 0 or node 2, the graph will be divided into 2 or more components.

Example 2:
Input Format:

Result={1,4}
Explanation: If we remove either node 1 or node 4, the graph breaks into multiple components.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V+2E), where V = no. of vertices, E = no. of edges. It is because the algorithm is just a s

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    int timer = 1;
    // DFS to find articulation points
    void dfs(int node, int parent, vector<int>
      &vis, int tin[], int low[],
             vector<int> &mark, vector<int>
               adj[]) {
        vis[node] = 1;
        tin[node] = low[node] = timer++; //
          set discovery and low time
        int child = 0;
        for (auto it : adj[node]) {
            if (it == parent) continue; //
              skip the edge to parent
            if (!vis[it]) {
                dfs(it, node, vis, tin, low,
                  mark, adj); // recursive DFS
                low[node] = min(low[node],
                  low[it]);     // update low
                  time
                // Check articulation
                  condition (excluding root)
                if (low[it] >= tin[node] &&
                  parent != -1) {
                    mark[node] = 1;
                }
                child++;
            }
            else {
                // back edge case
                low[node] = min(low[node],
                  tin[it]);
            }
        }
        // If root node has more than one
          child
        if (parent == -1 && child > 1) {
            mark[node] = 1;
        }
    }
public:
    vector<int> articulationPoints(int n,
      vector<int> adj[]) {
        vector<int> vis(n, 0), mark(n, 0);
        int tin[n], low[n];
        for (int i = 0; i < n; i++) {
            if (!vis[i]) {
                dfs(i, -1, vis, tin, low,
                  mark, adj);
            }
        }
        vector<int> ans;
        for (int i = 0; i < n; i++) {
            if (mark[i]) ans.push_back(i);
        }
        return ans.empty() ? vector<int>{-1} :
          ans;
    }
};
```

Q325. Kosaraju's algorithm

HARD

Topic: Graphs [Concepts & Problems] > Other Algorithms

Practice Link

Input Format:

Result: 3
Explanation: Three strongly connected components are marked below:

Input Format:

Result: 4
Explanation: Four strongly connected components are marked below:

[OPTIMAL SOLUTION] Approach — Time Complexity: O(V + E), since each node and edge is visited at most twice (once in

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    void dfs(int node, vector<int> &vis,
      vector<int> adj[], stack<int> &st) {
        vis[node] = 1;
        for (auto it : adj[node]) {
            if (!vis[it]) {
                dfs(it, vis, adj, st);
            }
        }
        st.push(node);
    }
    void dfs3(int node, vector<int> &vis,
      vector<int> adjT[]) {
        vis[node] = 1;
        for (auto it : adjT[node]) {
            if (!vis[it]) {
                dfs3(it, vis, adjT);
            }
        }
    }
public:
    int kosaraju(int V, vector<int> adj[]) {
        vector<int> vis(V, 0);
        stack<int> st;
        for (int i = 0; i < V; i++) {
            if (!vis[i]) {
                dfs(i, vis, adj, st);
            }
        }
        vector<int> adjT[V];
        for (int i = 0; i < V; i++) {
            vis[i] = 0;
            for (auto it : adj[i]) {
                adjT[it].push_back(i);
            }
        }
        int scc = 0;
        while (!st.empty()) {
            int node = st.top();
            st.pop();
            if (!vis[node]) {
                scc++;
                dfs3(node, vis, adjT);
            }
        }
        return scc;
    }
};
```

Q326. Introduction to DP

EASY

Topic: Dynamic Programming [Patterns and Problems] > Introduction to DP

Practice Link

Refer to the code implementation and logic below.

[OPTIMAL SOLUTION] General

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int fib(int n) {
        // If n is 0 return 0
        if(n == 0) return 0;
        // If n is 1 return 1
        if(n == 1) return 1;

        // prev2 stores fib(n-2)
        int prev2 = 0;
        // prev stores fib(n-1)
        int prev = 1;
        // curr stores current fib
        int curr;

        // Loop from 2 to n
        for(int i = 2; i <= n; i++) {
            // Calculate current fib
            curr = prev + prev2;
            // Update prev2
            prev2 = prev;
            // Update prev
            prev = curr;
        }
        // Return final answer
        return prev;
    }
};
```

Q327. Climbing stairs

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 1D DP

Practice Link

Input: n = 2
Output: 2
Explanation: There are 2 unique ways to climb to the 2nd step:
1. 1 step + 1 step
2. 2 steps

Input: n = 3
Output: 3
Explanation: There are 3 unique ways to climb to the 3rd step:
1. 1 step + 1 step + 1 step
2. 2 steps + 1 step
3. 1 step + 2 steps

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), since we compute each Fibonacci number from 2 to n exactly once

```cpp
#include <bits/stdc++.h>
using namespace std;
```

Q328. Frog Jump

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 1D DP

Practice Link

Example 1:
Input: heights = [2, 1, 3, 5, 4]
Output: 2
Explanation: One possible route can be,
0th step -> 2nd Step = abs(2 - 3) = 1
2nd step -> 4th step = abs(3 - 4) = 1
Total = 1 + 1 = 2.

Example 2:
Input: heights = [7, 5, 1, 2, 6]
Output: 9
Explanation: One... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(n), since we fill the DP array once with at most two*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Computes minimum energy to reach last
      index using bottom-up DP
    int frogJump(const vector<int>& height) {
        // Handle empty input
        if (height.empty()) return 0;

        // Fetch size of the input
        int n = (int)height.size();

        // Create dp array where dp[i] = min
          energy to reach i
        vector<int> dp(n, INT_MAX);

        // Base case: cost to stand on first
          stone is zero
        dp[0] = 0;

        // Iterate over stones from index 1 to
          n-1
        for (int ind = 1; ind < n; ind++) {
            // Compute cost for a jump from
              ind-1
            int jumpOne = dp[ind - 1] +
              abs(height[ind] - height[ind -
              1]);

            // Initialize jumpTwo with large
              value
            int jumpTwo = INT_MAX;

            // If possible, compute cost for a
              jump from ind-2
            if (ind > 1) {
                jumpTwo = dp[ind - 2] +
                  abs(height[ind] - height[ind
                  - 2]);
            }

            // Take the minimum of the two
              options
            dp[ind] = min(jumpOne, jumpTwo);
        }

        // Return min energy to reach last
          stone
        return dp[n - 1];
    }
};
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(n), since each state (index) is computed once and stored in*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Solve function using recursion with
      memoization
    // ind -> current index the frog needs to
      reach
    // height -> vector of heights
    // dp -> memo table where dp[i] stores min
      cost to reach i
    int solve(int ind, const vector<int>&
      height, vector<int>& dp) {
        // If at the first stone, cost is 0
        if (ind == 0) return 0;

        // Return memoized result if already
          computed
        if (dp[ind] != -1) return dp[ind];

        // Initialize jumpTwo with a large
          value
        int jumpTwo = INT_MAX;

        // Compute cost when jumping from
          previous stone (ind - 1)
        int jumpOne = solve(ind - 1, height,
          dp) + abs(height[ind] - height[ind -
          1]);

        // Compute cost when jumping from two
          stones back (ind - 2) if possible
        if (ind > 1) {
            jumpTwo = solve(ind - 2, height,
              dp) + abs(height[ind] -
              height[ind - 2]);
        }

        // Memoize and return the minimum of
          the two choices
        return dp[ind] = min(jumpOne,
          jumpTwo);
    }

    // Helper to handle edge cases and start
      recursion
    int frogJump(const vector<int>& height) {
        // Handle empty input
        if (height.empty()) return 0;

        // Prepare dp with -1 indicating
          uncomputed states
        int n = (int)height.size();
        vector<int> dp(n, -1);

        // Start from the last index
        return solve(n - 1, height, dp);
    }
};
```

Q329. Frog jump with K distances

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 1D DP

Practice Link

Example 1:
Input: heights = [10, 5, 20, 0, 15], k = 2
Output: 15
Explanation:
0th step -> 2nd step, cost = abs(10 - 20) = 10
2nd step -> 4th step, cost = abs(20 - 15) = 5
Total cost = 10 + 5 = 15.

Example 2:
Input: heights = [15, 4, 1, 14, 15], k = 3
Output:... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(n × k), where n is the number of stones and*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to compute the minimum cost to
  reach the end using at most 'k' jumps
int solveUtil(int n, vector<int>& height,
  vector<int>& dp, int k) {
    // Base case: cost to reach the first
      stone is 0
    dp[0] = 0;

    // Iterate over each stone
    for (int i = 1; i < n; i++) {
        // Initialize the minimum cost for
          this stone as large value
        int mmSteps = INT_MAX;

        // Try all possible jump lengths from
          1 to k
        for (int j = 1; j <= k; j++) {
            // Ensure jump doesn't go out of
              bounds
            if (i - j >= 0) {
                // Cost of jumping from (i -
                  j) to i
                int jump = dp[i - j] +
                  abs(height[i] - height[i -
                  j]);
                // Keep track of the minimum
                  cost
                mmSteps = min(mmSteps, jump);
            }
        }

        // Store the computed minimum cost for
          this stone
        dp[i] = mmSteps;
    }

    // The last element of dp stores the
      answer
    return dp[n - 1];
}

// Main function to solve the problem
int solve(int n, vector<int>& height, int k) {
    // Initialize DP array with -1
    vector<int> dp(n, -1);
    // Fill DP array iteratively
    return solveUtil(n, height, dp, k);
}

// Driver code
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(n * k), where n is the number of stairs and*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to find the minimum cost to reach
  index 'ind' using at most 'k' jumps
int solveUtil(int ind, vector<int>& height,
  vector<int>& dp, int k) {
    // Base case: starting point has zero cost
    if (ind == 0) return 0;

    // Return already computed result
    if (dp[ind] != -1) return dp[ind];

    // Initialize minimum steps as large value
    int mmSteps = INT_MAX;

    // Try all possible jumps from 1 to k
    for (int j = 1; j <= k; j++) {
        // Ensure jump does not go out of
          bounds
        if (ind - j >= 0) {
            // Cost of taking the jump
            int jump = solveUtil(ind - j,
              height, dp, k) + abs(height[ind]
              - height[ind - j]);
            // Store the minimum cost
            mmSteps = min(jump, mmSteps);
        }
    }
    // Save the result in dp array
    return dp[ind] = mmSteps;
}

// Function to get minimum cost to reach end
int solve(int n, vector<int>& height, int k) {
    // DP array initialized to -1
    vector<int> dp(n, -1);
    // Start recursion from last index
    return solveUtil(n - 1, height, dp, k);
}

// Driver code
```

Q330. Maximum sum of non adjacent elements

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 1D DP

Practice Link

Input: nums = [1, 2, 4]
Output: 5
Explanation:
Subsequence {1,4} gives maximum sum.

Input: [2, 1, 4, 9]
Output: 11
Explanation:
Subsequence {2,9} gives maximum sum

[OPTIMAL SOLUTION] General — Time Complexity: O(N), every element of array is processed once. Space Complexity: O(1) , only constant va

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to return maximum sum of non-adjacent elements
    int maxSum(vector<int>& nums) {
        // Handle edge case for empty input
        if (nums.empty()) return 0;

        // Initialize two tracking sums
        int prev2 = 0;
        int prev = nums[0];

        // Traverse through the array
        for (int i = 1; i < nums.size(); i++) {
            // Include current by adding it to two steps back
            int include = nums[i] + prev2;

            // Exclude current by taking previous best
            int exclude = prev;

            // Choose max of include and exclude
            int curr = max(include, exclude);

            // Update tracking variables
            prev2 = prev;
            prev = curr;
        }

        // Final result is stored in prev
        return prev;
    }
};
```

Q331. House robber

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 1D DP

Practice Link

Example 1:
Input: money = [2, 1, 4, 9]
Output: 10
Explanation: [2, 1, 4, 9] The underlined houses would give the maximum loot.
Note that we cannot loot the 1st and 4th houses together.

Example 2:
Input: money = [1, 5, 2, 1, 6]
Output: 11
Explanation: [1, 5, 2, 1, 6] The underlined houses would give the maximum loot.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n), where n is the number of houses. We process each house once in

```cpp
#include <bits/stdc++.h>
using namespace std;
// Function to solve the linear house robber
  problem (no circular arrangement)
long long int solve(vector<int>& arr) {
    // Get the size of the array
    int n = arr.size();
    // If there's only one house, return its
      value
    if (n == 1) return arr[0];
    // prev stores the maximum sum till the
      previous index
    long long int prev = arr[0];
    // prev2 stores the maximum sum till the
      index before previous
    long long int prev2 = 0;
    // Loop through houses starting from index
      1
    for (int i = 1; i < n; i++) {
        // Option 1: Pick the current house
          and add the value from prev2
        long long int pick = arr[i];
        if (i > 1) pick += prev2;
        // Option 2: Skip the current house,
          take prev
        long long int nonPick = prev;
        // Choose the maximum of pick and
          nonPick
        long long int cur_i = max(pick,
          nonPick);
        // Update prev2 and prev for the next
          iteration
        prev2 = prev;
        prev = cur_i;
    }
    // prev will contain the maximum loot
      possible
    return prev;
}
// Function to solve the circular house robber
  problem
long long int robStreet(int n, vector<int>
  &arr) {
    // If there are no houses, return 0
    if (n == 0) return 0;
    // If there is only one house, return its
      value
    if (n == 1) return arr[0];
    // Create two arrays:
    // arr1 excludes the first house
    // arr2 excludes the last house
    vector<int> arr1, arr2;
    for (int i = 0; i < n; i++) {
        if (i != 0) arr1.push_back(arr[i]);
        if (i != n - 1)
          arr2.push_back(arr[i]);
    }
    // Compute maximum loot for both cases and
      return the maximum
    long long int ans1 = solve(arr1);
    long long int ans2 = solve(arr2);
    return max(ans1, ans2);
}
// Driver code
```

Q332. Ninja's training

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 2D/3D DP and DP on Grids

Practice Link

Example 1:
Input: matrix = [[10, 40, 70], [20, 50, 80], [30, 60, 90]]
Output: 210
Explanation:
Day 1: fighting practice = 70
Day 2: stealth training = 50
Day 3: fighting practice = 90
Total = 70 + 50 + 90 = 210
This gives the optimal points.

Example 2:
Input:... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(n × 4 × 3) = O(n) For*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to find the maximum points for
  ninja training
int ninjaTraining(int n, vector<vector<int>>&
  points) {
  // Create a 2D DP (Dynamic Programming)
    table to store the maximum points
  // dp[i][j] represents the maximum points at
    day i, considering the last activity as j
  vector<vector<int>> dp(n, vector<int>(4,
    0));

  // Initialize the DP table for the first day
    (day 0)
  dp[0][0] = max(points[0][1], points[0][2]);
  dp[0][1] = max(points[0][0], points[0][2]);
  dp[0][2] = max(points[0][0], points[0][1]);
  dp[0][3] = max(points[0][0],
    max(points[0][1], points[0][2]));  // Max
    points for day 0 considering any activity

  // Iterate through the days starting from
    day 1
  for (int day = 1; day < n; day++) {
    // For each day, check all possible "last
      activities" (0, 1, 2, or no previous
      activity 3)
    for (int last = 0; last < 4; last++) {
      dp[day][last] = 0; // Initialize
        dp[day][last] to 0
      // Iterate through the tasks
        (activities) for the current day
      for (int task = 0; task <= 2; task++) {
        // If the current task is different
          from the last task, we can perform
          this task
        if (task != last) {
          // Calculate the points for the
            current activity and add it to the
            maximum points obtained
          // on the previous day with the
            previous task
          int activity = points[day][task] +
            dp[day - 1][task];
          // Update the maximum points for the
            current day and last activity
          dp[day][last] = max(dp[day][last],
            activity);
        }
      }
    }
  }

  // The maximum points for the last day with
    any activity can be found in dp[n-1][3]
  return dp[n - 1][3];
}
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(n × 4 × 3) = O(n) The*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Recursive function to calculate the maximum
  points for the ninja training
int f(int day, int last, vector<vector<int>>
  &points, vector<vector<int>> &dp) {
    // If the result for this day and last
      activity is already calculated, return
      it
    if (dp[day][last] != -1) return
      dp[day][last];

    // Base case: When we reach the first day
      (day == 0)
    if (day == 0) {
        int maxi = 0;
        // Calculate the maximum points for
          the first day by choosing an
          activity
        // different from the last one
        for (int i = 0; i <= 2; i++) {
            if (i != last)
                maxi = max(maxi,
                  points[0][i]);
        }
        // Store the result in dp array and
          return it
        return dp[day][last] = maxi;
    }

    int maxi = 0;
    // Iterate through the activities for the
      current day
    for (int i = 0; i <= 2; i++) {
        if (i != last) {
            // Calculate the points for the
              current activity and add it to
              the
            // maximum points obtained so far
              (recursively calculated)
            int activity = points[day][i] +
              f(day - 1, i, points, dp);
            maxi = max(maxi, activity);
        }
    }

    // Store the result in dp array and return
      it
    return dp[day][last] = maxi;
}

// Function to find the maximum points for
  ninja training
int ninjaTraining(int n, vector<vector<int>>
  &points) {
    // Create a memoization table (dp) to
      store intermediate results
    vector<vector<int>> dp(n, vector<int>(4,
      -1));
    // Start the recursive calculation from
      the last day with no previous activity
    return f(n - 1, 3, points, dp);
}
```

Q333. Grid Unique Paths : DP on Grids (DP8)

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 2D/3D DP and DP on Grids

Practice Link

Example 1:
Input: m = 3, n = 2
Output: 3
Explanation: There are 3 unique ways to go from the top-left to the bottom-right cell:
1) right → down → down
2) down → right → down
3) down → down → right

Example 2:
Input: m = 2, n = 4
Output: 4
Exp... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(m * n), where m is the number of rows and n is the number of colum*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    //Function to solve the problem using
      tabulation
    int func(int m, int n,
      vector<vector<int>>& dp){
        // Loop through the grid using two
          nested loops
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
            // Base condition
            if (i == 0 && j == 0) {
                dp[i][j] = 1;
                continue;
            }
            int up = 0;
            int left = 0;
            if (i > 0)
                up = dp[i - 1][j];
            if (j > 0)
                left = dp[i][j - 1];
            dp[i][j] = up + left;
        }
    }
    // The result is stored in bottom-right
      cell (m-1, n-1).
    return dp[m - 1][n - 1];
}
public:
    int uniquePaths(int m, int n) {
        vector<vector<int>> dp(m,
          vector<int>(n, -1));
        //Return the total count(0 based
          indexing)
        return func(m, n, dp);
    }
};
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(m * n), where m is the number of rows and n is the number of colum*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
private:
    //Function to solve the problem using
      recursion
    int func(int i, int j,
      vector<vector<int>>& dp){
        // Base case
        if (i == 0 && j == 0)  return 1;

        /* If we go out of bounds or reach
        a blocked cell, there are no ways.*/
        if (i < 0 || j < 0)  return 0;

        /* If we have already computed the
          number
        of ways for this cell, return it.*/
        if (dp[i][j] != -1)  return dp[i][j];

        /* Calculate the number of ways by
        moving up and left recursively.*/
        int up = func(i - 1, j, dp);
        int left = func(i, j - 1, dp);

        // Store the result in dp table and
          return it.
        return dp[i][j] = up + left;
    }
public:
    /*Function to count the total ways
    to reach (0,0) from (m-1,n-1)*/
    int uniquePaths(int m, int n) {
        /* Initialize a memoization table (dp)
          to
        store the results of subproblems.*/
        vector<vector<int>> dp(m,
          vector<int>(n, -1));

        //Return the total count(0 based
          indexing)
        return func(m-1,n-1, dp);
    }
};
```

Q334. Unique paths II

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 2D/3D DP and DP on Grids

Practice Link

Example 1:
Input: matrix = [[0, 0, 0], [0, 1, 0], [0, 0, 0]]
Output: 2
Explanation:
The two possible paths to reach the bottom-right cell are:
1) down → down → right → right
2) right → right → down → down
Thus, the number of paths is 2.

Example... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(m * n), where m is the number of rows and n is the number of colum*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
private:
    // Function to solve the problem using
      tabulation
    int func(int m, int n,
      vector<vector<int>>& matrix,
      vector<vector<int>>& dp) {
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                // Base conditions
                if (matrix[i][j] == 1) {
                    dp[i][j] = 0;
                    continue;
                }
                if (i == 0 && j == 0) {
                    dp[i][j] = 1;
                    continue;
                }
                int up = 0;
                int left = 0;
                if (i > 0)
                    up = dp[i - 1][j];
                if (j > 0)
                    left = dp[i][j - 1];
                dp[i][j] = up + left;
            }
        }
        return dp[m - 1][n - 1];
    }
public:
    int
      uniquePathsWithObstacles(vector<vector<i
      nt>>& matrix) {
        int m = matrix.size();
        int n = matrix[0].size();
        // Initialize DP table to memoize
          results
        vector<vector<int>> dp(m,
          vector<int>(n, 0));
        // Return the total number of paths
        return func(m, n, matrix, dp);
    }
};
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(m * n), where m is the number of rows and n is the number of colum*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
private:

    //Function to solve the problem using
      memoization
    int func(int i, int j,
      vector<vector<int>>& matrix,
      vector<vector<int>> &dp){
        // Base cases
        if (i < 0 || j < 0 || matrix[i][j] ==
          1) return 0;
        else if(i == 0 && j == 0) return 1;

        // If the result is already computed,
          return it
        if(dp[i][j] != -1) return dp[i][j];

        /* Calculate the number of ways by
        moving up and left recursively.*/
        int up = func(i - 1, j, matrix, dp);
        int left = func(i, j - 1, matrix, dp);

        // Return the total ways
        return dp[i][j] = up + left;
    }

public:

    /*Function to find all unique paths to
      reach
    matrix[m-1][n-1] form matrix[0][0] with
      obstacles*/
    int
      uniquePathsWithObstacles(vector<vector<i
      nt>>& matrix) {
        int m = matrix.size();
        int n = matrix[0].size();

        // Initialize DP table to memoize
          results
        vector<vector<int>> dp(m,
          vector<int>(n, -1));

        //Return the total number of paths
        return func(m-1, n-1, matrix, dp);
    }
};
```

Q335. Minimum Falling Path Sum

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 2D/3D DP and DP on Grids

Practice Link

Input: grid = [[5,9,6],[11,5,2]]
Output: 21
Explanation: Minimum sum is achieved via path 5->9->5->2 i.e. 21.

Input : grid = [[1,2,3],[4,5,6]]
Output: 12
Explanation : Minimum sum is achieved via path 1->2->3->6 i.e. 12.

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*M), entire grid is visited atleast once.*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to solve the problem
class Solution {
public:
    // Function to calculate minimum path sum
    int minPathSum(vector<vector<int>>
      &matrix) {
        int n = matrix.size();
        int m = matrix[0].size();

        // Create DP table
        vector<vector<int>> dp(n,
          vector<int>(m, 0));

        // Fill the table
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {

                // First cell initialization
                if (i == 0 && j == 0)
                    dp[i][j] = matrix[i][j];
                else {
                    // Calculate from top
                    int up = matrix[i][j];
                    if (i > 0) up += dp[i -
                      1][j];
                    else up += 1e9;

                    // Calculate from left
                    int left = matrix[i][j];
                    if (j > 0) left += dp[i][j
                      - 1];
                    else left += 1e9;

                    // Take minimum
                    dp[i][j] = min(up, left);
                }
            }
        }

        // Return result
        return dp[n - 1][m - 1];
    }
};

// Driver function
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*M), at max, there will be N*M calls of recursion..*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class containing the solution logic
class Solution {
public:
    // Function to calculate minimum path sum
      with memoization
    int minPath(int i, int j,
                vector<vector<int>> &grid,
                vector<vector<int>> &dp) {
        // If we are at (0,0), return that
          cell's value
        if (i == 0 && j == 0)
            return grid[0][0];

        // If out of bounds, return large
          number
        if (i < 0 || j < 0)
            return 1e9;

        // If already computed, return from dp
        if (dp[i][j] != -1)
            return dp[i][j];

        // Compute path by going up
        int up = grid[i][j] +
                 minPath(i - 1, j, grid, dp);

        // Compute path by going left
        int left = grid[i][j] +
                   minPath(i, j - 1, grid,
                     dp);

        // Store the minimum in dp and return
        return dp[i][j] = min(up, left);
    }

    // Main function that initializes dp and
      calls helper
    int minPathSum(vector<vector<int>> &grid)
      {
        int n = grid.size();
        int m = grid[0].size();

        // Create dp table initialized with -1
        vector<vector<int>> dp(n,
                vector<int>(m, -1));

        // Start from bottom-right corner
        return minPath(n - 1, m - 1, grid,
          dp);
    }
};

// Driver function
```

Q336. Triangle

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 2D/3D DP and DP on Grids

Practice Link

Input: triangle = [[1], [1, 2], [1, 2, 4]]
Output: 3
Explanation: One possible route can be:
Start at 1st row -> bottom -> bottom.

Input : triangle = [[1], [4, 7], [4,10, 50], [-50, 5, 6, -100]]
Output: -42
Explanation : One possible route can be:
Start at 1... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*N), entire triangular grid is visited atleast once.*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to solve the triangle minimum path
  sum
class Solution {
public:
    // Function to compute the minimum path
      sum using tabulation
    int minimumPathSum(vector<vector<int>>
      &triangle, int n) {
        // Create a 2D dp array to store
          intermediate results
        vector<vector<int>> dp(n,
          vector<int>(n, 0));

        // Initialize the last row of dp with
          triangle values
        for (int j = 0; j < n; j++) {
            dp[n - 1][j] = triangle[n - 1][j];
        }

        // Traverse from second-last row to
          the top
        for (int i = n - 2; i >= 0; i--) {
            for (int j = i; j >= 0; j--) {
                // Calculate sum from down and
                  diagonal paths
                int down = triangle[i][j] +
                  dp[i + 1][j];
                int diag = triangle[i][j] +
                  dp[i + 1][j + 1];

                // Store the minimum of the
                  two paths
                dp[i][j] = min(down, diag);
            }
        }

        // Return the minimum path sum from
          top
        return dp[0][0];
    }
};

// Main function to run the code
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*N), we fill our complete 2D DP table one by one.*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class to solve the problem
class Solution {
public:
    // Recursive function with memoization
    int solve(int i, int j,
      vector<vector<int>> &triangle, int n,
      vector<vector<int>> &dp) {
        // If value already computed
        if (dp[i][j] != -1)
            return dp[i][j];

        // If at bottom row
        if (i == n - 1)
            return triangle[i][j];

        // Compute both downward and diagonal
          moves
        int down = triangle[i][j] + solve(i +
          1, j, triangle, n, dp);
        int diag = triangle[i][j] + solve(i +
          1, j + 1, triangle, n, dp);

        // Store and return min path sum
        return dp[i][j] = min(down, diag);
    }

    // Function to start the process
    int minimumPathSum(vector<vector<int>>
      &triangle) {
        int n = triangle.size();
        vector<vector<int>> dp(n,
          vector<int>(n, -1));
        return solve(0, 0, triangle, n, dp);
    }
};

// Main function
```

Q337. Ninja and his Friends

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > 2D/3D DP and DP on Grids

Practice Link

Example 1:
Input: 'R' = 3, 'C' = 4
'GRID' = [[2, 3, 1, 2], [3, 4, 2, 2], [5, 6, 3, 5]]
Output: 21

Example 2:
Input: 'R' = 2, 'C' = 3
'GRID' = [[4, 1, 2], [7, 3, 5]]
Output: 22

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*M*M)*9, The outer nested loops run for (N*M*M) times and the inn*

```
#include <bits/stdc++.h>
using namespace std;
// Class to solve Ninja and his friends using
  tabulation
class Solution {
public:
    int maximumChocolates(int n, int m,
      vector<vector<int>>& grid) {
        // 3D DP table
        vector<vector<vector<int>>> dp(n,
            vector<vector<int>>(m,
              vector<int>(m, 0)));
        // Base case: last row
        for (int j1 = 0; j1 < m; j1++) {
            for (int j2 = 0; j2 < m; j2++) {
                if (j1 == j2) dp[n-1][j1][j2]
                  = grid[n-1][j1];
                else dp[n-1][j1][j2] =
                  grid[n-1][j1] +
                  grid[n-1][j2];
            }
        }
        // Fill DP table bottom-up
        for (int i = n - 2; i >= 0; i--) {
            for (int j1 = 0; j1 < m; j1++) {
                for (int j2 = 0; j2 < m; j2++)
                  {
                    int maxi = -1e9;
                    int curr = (j1 == j2) ?
                      grid[i][j1]
                                          :
                                            gr
                                            id
                                            [i
                                            ][
                                            j1
                                            ]
                                            +
                                            gr
                                            id
                                            [i
                                            ][
                                            j2
                                            ];
                    // Try all 9 moves
                    for (int dj1 = -1; dj1 <=
                      1; dj1++) {
                        for (int dj2 = -1; dj2
                          <= 1; dj2++) {
                            int newJ1 = j1 +
                              dj1;
                            int newJ2 = j2 +
                              dj2;
                            if (newJ1 >= 0 &&
                              newJ1 < m &&
                                newJ2 >= 0 &&
                                  newJ2 < m) {
                                maxi =
                                  max(maxi,
                                  curr +
dp[i+1][newJ1][newJ2]);
                            } else {
                                maxi =
                                  max(maxi,
                                  (int)-1e9);
                            }
                        }
                    }
                    dp[i][j1][j2] = maxi;
                }
            }
        }
        return dp[0][0][m-1];
    }
};
// Driver code
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*M*M) * 9, At max, there will be N*M*M calls of recursion to solv*

```
#include <bits/stdc++.h>
using namespace std;
// Class to solve Ninja and his friends using
  memoization
class Solution {
public:
    // Recursive function with memoization
    int solve(int i, int j1, int j2, int n,
      int m,
              vector<vector<int>>& grid,
              vector<vector<vector<int>>>& dp)
                {
        // Out of boundary check
        if (j1 < 0 || j1 >= m || j2 < 0 || j2
          >= m)
            return -1e9;
        // Base case: last row
        if (i == n - 1) {
            if (j1 == j2) return grid[i][j1];
            else return grid[i][j1] +
              grid[i][j2];
        }
        // If already computed return it
        if (dp[i][j1][j2] != -1) return
          dp[i][j1][j2];
        // Take chocolates from current
          cell(s)
        int maxi = -1e9;
        int curr = (j1 == j2) ? grid[i][j1] :
          grid[i][j1] + grid[i][j2];
        // Try all 9 moves
        for (int dj1 = -1; dj1 <= 1; dj1++) {
            for (int dj2 = -1; dj2 <= 1;
              dj2++) {
                int ans = curr + solve(i + 1,
                  j1 + dj1, j2 + dj2,
                                    n, m,
                                      grid,
                                      dp);
                maxi = max(maxi, ans);
            }
        }
        // Store result
        return dp[i][j1][j2] = maxi;
    }
    // Main function to call
    int maximumChocolates(int n, int m,
      vector<vector<int>>& grid) {
        vector<vector<vector<int>>> dp(n,
            vector<vector<int>>(m,
              vector<int>(m, -1)));
        return solve(0, 0, m - 1, n, m, grid,
          dp);
    }
};
// Driver code
```

Q338. Subset sum equal to target (DP- 14)

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input : N = 4, ARR = [4, 3, 5, 2], K = 6
Output : true
Explanation : One possible subset with sum = 6 is [4, 2]. There's also [3, 3] but that doesn't exist in the array. As soon as we find one subset whose sum is equal to K, the answer is true.
Input : N = 3... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*K),There are two nested loops Space Complexity: O(N*K), We are u*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to check if there is a subset
      of 'arr' with a sum equal to 'k'
    bool subsetSumToK(int n, int k,
      vector<int> &arr) {
        // Initialize a 2D DP array with
          dimensions (n x k+1) to store
          subproblem results
        vector<vector<bool>> dp(n,
          vector<bool>(k + 1, false));

        // Base case: If the target sum is 0,
          we can always achieve it by taking
          no elements
        for (int i = 0; i < n; i++) {
            dp[i][0] = true;
        }

        // Base case: If the first element of
          'arr' is less than or equal to 'k',
          set dp[0][arr[0]] to true
        if (arr[0] <= k) {
            dp[0][arr[0]] = true;
        }

        // Fill the DP array iteratively
        for (int ind = 1; ind < n; ind++) {
            for (int target = 1; target <= k;
              target++) {
                // If we don't take the
                  current element, the result
                  is the same as the previous
                  row
                bool notTaken = dp[ind -
                  1][target];

                // If we take the current
                  element, subtract its value
                  from the target and check
                  the previous row
                bool taken = false;
                if (arr[ind] <= target) {
                    taken = dp[ind - 1][target
                      - arr[ind]];
                }

                // Store the result in the DP
                  array for the current
                  subproblem
                dp[ind][target] = notTaken ||
                  taken;
            }
        }

        // The final result is stored in
          dp[n-1][k]
        return dp[n - 1][k];
    }
};

// Driver code
```

[OPTIMAL] Memorization Approach

Complexity: *Time Complexity: O(N*K),There are N*K states therefore at max 'N*K' new problems will*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Recursive helper function to check if a
      subset sum equals target
    bool subsetSumUtil(int ind, int target,
      vector<int>& arr, vector<vector<int>>&
      dp) {
        // Base case: target achieved
        if (target == 0) return true;

        // Base case: at the first index,
          check if it equals the target
        if (ind == 0) return arr[0] == target;

        // Check memoization table
        if (dp[ind][target] != -1) return
          dp[ind][target];

        // Choice 1: do not take the current
          element
        bool notTaken = subsetSumUtil(ind - 1,
          target, arr, dp);

        // Choice 2: take the current element
          if possible
        bool taken = false;
        if (arr[ind] <= target) {
            taken = subsetSumUtil(ind - 1,
              target - arr[ind], arr, dp);
        }

        // Store result in DP table
        return dp[ind][target] = notTaken ||
          taken;
    }

    // Main function to check if subset with
      sum = k exists
    bool subsetSumToK(int n, int k,
      vector<int>& arr) {
        vector<vector<int>> dp(n,
          vector<int>(k + 1, -1));
        return subsetSumUtil(n - 1, k, arr,
          dp);
    }
};

// Driver code
```

Q339. Partition equal subset sum

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input: nums = [2, 3, 3, 3, 4, 5]
Output: True
Explanation: Nums can be partitioned into two subsets of sum 10.

Input: nums = [1, 2, 3, 5]
Output: False
Explanation: The array cannot be partitioned into equal sum subsets.

[OPTIMAL SOLUTION] General — Time Complexity: O(N*K), there are total N*K states, where N is the length of array and K is the target sum i.

```cpp
#include <bits/stdc++.h>
using namespace std;
// Function to check if it's possible to
  partition the array into two subsets with
  equal sum
bool canPartition(int n, vector<int>& arr) {
    int totSum = 0;
    // Calculate the total sum of the array
    for (int i = 0; i < n; i++) {
        totSum += arr[i];
    }
    // If the total sum is odd, it cannot be
      partitioned into two equal subsets
    if (totSum % 2 == 1)
        return false;
    else {
        int k = totSum / 2;
        // Create a vector to represent the
          previous row of the DP table
        vector<bool> prev(k + 1, false);
        // Base case: If the target sum is 0,
          it can be achieved by not selecting
          any elements
        prev[0] = true;
        // Initialize the first row based on
          the first element of the array
        if (arr[0] <= k)
            prev[arr[0]] = true;
        // Fill in the DP table using a
          bottom-up approach
        for (int ind = 1; ind < n; ind++) {
            // Create a vector to represent
              the current row of the DP table
            vector<bool> cur(k + 1, false);
            cur[0] = true;
            for (int target = 1; target <= k;
              target++) {
                // Exclude the current element
                bool notTaken = prev[target];
                // Include the current element
                  if it doesn't exceed the
                  target
                bool taken = false;
                if (arr[ind] <= target)
                    taken = prev[target -
                      arr[ind]];
                // Update the current row of
                  the DP table
                cur[target] = notTaken ||
                  taken;
            }
            // Set the current row as the
              previous row for the next
              iteration
            prev = cur;
        }
        // The final result is in the last
          cell of the previous row of the DP
          table
        return prev[k];
    }
}
```

Q340. Partition a set into two subsets with minimum absolute sum difference

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input: nums = [1, 2, 3, 4]
Output: 0
Explanation: Two subsets can be [1,4] and [2,3].

Input: nums = [8, 6, 5]
Output: 3
Explanation: Two subsets can be [8] and [6, 5].

[OPTIMAL SOLUTION] General — Time Complexity: O(N*K), there are total N*K states, where N is the length of array and K is the total sum of

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find the minimum absolute
      difference between two subset sums
    int minSubsetSumDifference(vector<int>&
      arr, int n) {
        int totSum = 0;
        // Calculate the total sum of the
          array
        for (int i = 0; i < n; i++) {
            totSum += arr[i];
        }
        // Initialize a boolean vector 'prev'
          to represent the previous row of the
          DP table
        vector<bool> prev(totSum + 1, false);
        // Base case: If no elements are
          selected (sum is 0), it's a valid
          subset
        prev[0] = true;
        // Initialize the first row based on
          the first element of the array
        if (arr[0] <= totSum)
            prev[arr[0]] = true;
        // Fill in the DP table using a
          bottom-up approach
        for (int ind = 1; ind < n; ind++) {
            // Create a boolean vector 'cur'
              to represent the current row of
              the DP table
            vector<bool> cur(totSum + 1,
              false);
            cur[0] = true;
            for (int target = 1; target <=
              totSum; target++) {
                // Exclude the current element
                bool notTaken = prev[target];
                // Include the current element
                  if it doesn't exceed the
                  target
                bool taken = false;
                if (arr[ind] <= target)
                    taken = prev[target -
                      arr[ind]];
                // Mark current target as
                  achievable if either
                  excluding or including
                  current element works
                cur[target] = notTaken ||
                  taken;
            }
            // Set 'cur' as the 'prev' for the
              next iteration
            prev = cur;
        }
        int mini = INT_MAX;
        for (int i = 0; i <= totSum; i++) {
            if (prev[i]) {
                // Calculate the absolute
                  difference between two
                  subset sums
                int diff = abs(i - (totSum -
                  i));
                mini = min(mini, diff);
            }
        }
        return mini;
    }
};
```

Q341. Count subsets with sum K

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input: arr = [1, 2, 2, 3], K = 3
Output: 3

Explanation: The subarrays [1,2], [1,2] and [3] have a sum of 3.

Input: arr = [1, 2, 3, 4, 5], K = 5
Output: 3
Explanation: The subsets are [5], [2, 3], and [1, 4].

[OPTIMAL SOLUTION] General — Time Complexity: O(N * K), each state defined by index and target is computed once. Space Complexity: O(K), ex

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int countSubsets(vector<int>& arr, int K)
    {
        // Create a dp array of size K+1
          initialized to 0
        vector<int> dp(K + 1, 0);

        // Base case: One subset (empty set)
          makes sum 0
        dp[0] = 1;

        // If first element <= K, mark it as a
          subset
        if (arr[0] <= K) dp[arr[0]] += 1;

        // Loop through elements starting from
          index 1
        for (int i = 1; i < arr.size(); i++) {
            // Create a temporary dp array for
              current element
            vector<int> curr(K + 1, 0);

            // Empty set always makes sum 0
            curr[0] = 1;

            // Iterate over all possible
              targets
            for (int t = 0; t <= K; t++) {
                // Exclude current element
                int notTake = dp[t];

                // Include current element if
                  possible
                int take = 0;
                if (arr[i] <= t) {
                    take = dp[t - arr[i]];
                }

                // Total ways = include +
                  exclude
                curr[t] = take + notTake;
            }

            // Update dp for next iteration
            dp = curr;
        }

        // Return answer for sum K
        return dp[K];
    }
};
```

Q342. Count partitions with given difference

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input: arr = [1, 1, 2, 3], diff = 1
Output: 3
Explanation: The subsets are [1, 2] and [1, 3], [1, 3] and [1, 2], [1, 1, 2] and [3].

Input: arr = [1, 2, 3, 4], diff = 2
Output: 2
Explanation: The subsets are [1, 3] and [2, 4], [1, 2, 3] and [4].

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N * K), each state defined by index and target is computed once. Space Complexity: O(K), ex

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to count partitions with given
      difference
    int countPartitions(vector<int>& arr, int
      d) {
        // Calculate total sum of array
        int totalSum = accumulate(arr.begin(),
          arr.end(), 0);

        // Check if solution is possible
        if ((totalSum + d) % 2 != 0 || d >
          totalSum) return 0;

        // Calculate target sum
        int K = (totalSum + d) / 2;

        // Create dp array of size K+1
        vector<int> dp(K + 1, 0);

        // Base case: empty set makes sum 0
        dp[0] = 1;

        // If first element <= K, mark it
        if (arr[0] <= K) dp[arr[0]] += 1;

        // Process remaining elements
        for (int i = 1; i < arr.size(); i++) {
            vector<int> curr(K + 1, 0);
            curr[0] = 1;

            for (int t = 0; t <= K; t++) {
                int notTake = dp[t];
                int take = 0;
                if (arr[i] <= t) {
                    take = dp[t - arr[i]];
                }
                curr[t] = take + notTake;
            }
            dp = curr;
        }
        return dp[K];
    }
};
```

Q343. Assign Cookies

EASY

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input : Student = [1, 2, 3] , Cookie = [1, 1]
Output :1
Explanation : Only the first cookie (1) satisfies the first student (1), therefore only 1 student is content.

Input : Student = [1, 2] , Cookie = [1, 2, 3]
Output : 2
Explanation : Cookie 1 satisfies student 1 and cookie 2 satisfies student 2. Therefore, 2 students are content.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n*logn + m*logm), Both the arrays are sorted in increasing order.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the maximum number of content students
    int findContentChildren(vector<int>& student, vector<int>& cookie) {
        // Sort both arrays to apply the greedy strategy
        sort(student.begin(), student.end());
        sort(cookie.begin(), cookie.end());

        int studentIndex = 0;
        int cookieIndex = 0;

        // Try to assign cookies until any one list is fully processed
        while (studentIndex < student.size() && cookieIndex < cookie.size()) {
            // If the cookie satisfies the student's greed
            if (cookie[cookieIndex] >= student[studentIndex]) {
                studentIndex++;
            }
            // Move to next cookie in both cases
            cookieIndex++;
        }

        // Number of students satisfied is equal to studentIndex
        return studentIndex;
    }
};
```

Q344. Minimum Coins (DP - 20)

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input: coins = [1, 2, 5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1. We need 3 coins to make up the amount 11.

Input : coins = [2, 5], amount = 3
Output: -1
Explanation : It's not possible to make amount 3 with coins 2 and 5. Since we can't combine t... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*T), there are total of N*T states.*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Solution class to implement tabulation
  approach
class Solution {
public:
    // Function to find minimum coins
    int coinChange(vector<int>& coins, int
      amount) {
        // Creating dp array of size amount+1
        vector<int> dp(amount + 1, INT_MAX);

        // Base case: dp[0] = 0
        dp[0] = 0;

        // Loop through all amounts from 1 to
          amount
        for (int i = 1; i <= amount; i++) {
            // Try each coin
            for (int coin : coins) {
                // If coin can be used
                if (i - coin >= 0 && dp[i -
                  coin] != INT_MAX) {
                    // Update dp[i] with
                      minimum coins
                    dp[i] = min(dp[i], 1 +
                      dp[i - coin]);
                }
            }
        }

        // If dp[amount] is still infinity,
          return -1
        return dp[amount] == INT_MAX ? -1 :
          dp[amount];
    }
};

// Driver function
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*T), there are total of N*T states. S*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Solution class to implement memoization
  approach
class Solution {
public:
    // Function to find minimum coins
    int coinChange(vector<int>& coins, int
      amount) {
        // Creating dp array initialized with
          -2
        // (-2 means not calculated yet)
        vector<int> dp(amount + 1, -2);

        // Calling helper function
        return helper(coins, amount, dp);
    }

private:
    // Helper recursive function
    int helper(vector<int>& coins, int rem,
      vector<int>& dp) {
        // If remaining amount is zero
        if (rem == 0) return 0;

        // If remaining amount is negative
        if (rem < 0) return -1;

        // If already computed
        if (dp[rem] != -2) return dp[rem];

        // Initialize minimum with large value
        int mini = INT_MAX;

        // Try every coin
        for (int coin : coins) {
            // Recursive call
            int res = helper(coins, rem -
              coin, dp);

            // If result is valid
            if (res >= 0 && res < mini)
                mini = 1 + res;
        }

        // Store result in dp
        dp[rem] = (mini == INT_MAX) ? -1 :
          mini;
        return dp[rem];
    }
};

// Driver function
```

Q345. Target sum

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input : nums = [1,1,1,1,1], target = 3
Output : 5
Explanation : There are 5 ways to assign symbols to make the sum of nums be target 3.
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3
+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3
Input : n... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*K),There are two nested loops Space Complexity: O(N*K),We are us*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find number of ways to
      assign + or - to reach the target
    int findTargetSumWays(vector<int>& nums,
      int target) {
        int n = nums.size();

        // First calculate the total sum of
          all numbers
        int totalSum =
          accumulate(nums.begin(), nums.end(),
          0);

        // If (target + totalSum) is odd or
          target > totalSum, no valid
          partition exists
        if ((totalSum + target) % 2 != 0 ||
          abs(target) > totalSum) return 0;

        // We now need to count subsets with
          sum = (target + totalSum) / 2
        int newTarget = (totalSum + target) /
          2;

        // Create DP table: dp[i][j] = number
          of ways to make sum j using first i
          numbers
        vector<vector<int>> dp(n + 1,
          vector<int>(newTarget + 1, 0));

        // Base case: One way to form sum 0
          (by taking no elements)
        for (int i = 0; i <= n; i++) dp[i][0]
          = 1;

        // Fill DP table iteratively
        for (int i = 1; i <= n; i++) {
            for (int j = 0; j <= newTarget;
              j++) {
                // Exclude current element
                dp[i][j] = dp[i - 1][j];

                // Include current element if
                  it does not exceed current
                  target j
                if (nums[i - 1] <= j) {
                    dp[i][j] += dp[i - 1][j -
                      nums[i - 1]];
                }
            }
        }

        return dp[n][newTarget];
    }
};

// Driver code
```

[OPTIMAL] Memorization Approach

Complexity: *Time Complexity: O(N*K),There are N*K states therefore at max 'N*K' new problems will*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to count number of ways to
      assign '+' or '-' to reach target
    int findTargetSumWays(vector<int>& nums,
      int target) {
        // Calculate the total sum of the
          array
        int totalSum =
          accumulate(nums.begin(), nums.end(),
          0);
        // If target is not achievable
          (because of sum or parity), return 0
        if ((totalSum - target) < 0 ||
          (totalSum - target) % 2 != 0)
            return 0;
        // Our problem reduces to subset sum
          with sum = (totalSum - target) / 2
        int subsetSum = (totalSum - target) /
          2;
        // Initialize memo table with -1
        vector<vector<int>> dp(nums.size(),
          vector<int>(subsetSum + 1, -1));
        // Call recursive function with
          memoization
        return countSubsets(nums, nums.size()
          - 1, subsetSum, dp);
    }
private:
    // Recursive function with memoization
    int countSubsets(vector<int>& nums, int
      ind, int target, vector<vector<int>>&
      dp) {
        // Base case: when we are at index 0
        if (ind == 0) {
            // Two scenarios:
            if (target == 0 && nums[0] == 0)
              return 2; // {pick or not pick
              0}
            if (target == 0 || target ==
              nums[0]) return 1; // Either
              pick or skip
            return 0; // Otherwise no valid
              way
        }
        // Return if already computed
        if (dp[ind][target] != -1) return
          dp[ind][target];
        // Exclude current element
        int notPick = countSubsets(nums, ind -
          1, target, dp);
        // Include current element if it is <=
          target
        int pick = 0;
        if (nums[ind] <= target)
            pick = countSubsets(nums, ind - 1,
              target - nums[ind], dp);
        // Store and return result
        return dp[ind][target] = pick +
          notPick;
    }
};
// Driver code
```

Q346. Coin Change 2 (DP - 22)

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Example 1:
Input: coins = [2, 4,10], amount = 10
Output: 4
Explanation: The four combinations are:
10 = 10
10 = 4 + 4 + 2
10 = 4 + 2 + 2 + 2
10 = 2 + 2 + 2 + 2 + 2

Example 2:
Input: coins = [5], amount = 5
Output: 1
Explanation: There is one combination: 5 =... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*T), as There are two nested loops. Space Com*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to count the number of ways to
  make change for a given target sum
long countWaysToMakeChange(vector<int>& arr,
  int n, int T) {
    vector<vector<long>> dp(n, vector<long>(T
      + 1, 0)); // Create a DP table

    // Initializing base condition
    for (int i = 0; i <= T; i++) {
        if (i % arr[0] == 0)
            dp[0][i] = 1;
        // Else condition is automatically
          fulfilled,
        // as dp array is initialized to zero
    }

    for (int ind = 1; ind < n; ind++) {
        for (int target = 0; target <= T;
          target++) {
            long notTaken = dp[ind -
              1][target];

            long taken = 0;
            if (arr[ind] <= target)
                taken = dp[ind][target -
                  arr[ind]];

            dp[ind][target] = notTaken +
              taken;
        }
    }

    return dp[n - 1][T];
}
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N × Target), since each state (ind, T) is*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Recursive function with memoization to
  count ways to make change
long countWaysToMakeChangeUtil(vector<int>&
  arr, int ind, int T, vector<vector<long>>&
  dp) {
    // Base case: if we are at the first
      element
    if (ind == 0) {
        return (T % arr[0] == 0); // Only one
          way if divisible
    }

    // If already calculated, return stored
      value
    if (dp[ind][T] != -1)
        return dp[ind][T];

    // Do not take the current coin
    long notTaken =
      countWaysToMakeChangeUtil(arr, ind - 1,
      T, dp);

    // Take the current coin (if feasible)
    long taken = 0;
    if (arr[ind] <= T)
        taken = countWaysToMakeChangeUtil(arr,
          ind, T - arr[ind], dp);

    // Store the result in DP table
    return dp[ind][T] = notTaken + taken;
}

// Function to initialize DP table and start
  recursion
long countWaysToMakeChange(vector<int>& arr,
  int n, int T) {
    vector<vector<long>> dp(n, vector<long>(T
      + 1, -1));
    return countWaysToMakeChangeUtil(arr, n -
      1, T, dp);
}
```

Q347. Unbounded knapsack

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input: n = 3, W = 8, wt = [2, 4, 6], val = [5, 11, 13]
Output: 22
Explanation: We can take item with weight 2 (value 5) four times to fill capacity 8,total value = 5 × 4 = 20.
But a better choice: take item with weight 2 (value 5) twice and item with weight 4 (... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*W),There are two nested loops Space Complexity: O(N*W),We are us*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to solve the unbounded
      knapsack problem using tabulation
    int unboundedKnapsack(int n, int W,
      vector<int>& val, vector<int>& wt) {
        // Create a DP table where dp[i][j]
          represents max value using first i
          items and capacity j
        vector<vector<int>> dp(n,
          vector<int>(W + 1, 0));

        // Base condition: fill first row
          considering infinite supply of first
          item
        for (int i = wt[0]; i <= W; i++) {
            dp[0][i] = (i / wt[0]) * val[0];
        }

        // Loop through all items starting
          from second
        for (int ind = 1; ind < n; ind++) {
            // Loop through all capacities
              from 0 to W
            for (int cap = 0; cap <= W; cap++)
              {
                // Case 1: Not taking the
                  current item
                int notTaken = dp[ind -
                  1][cap];

                // Case 2: Taking the current
                  item (if it fits)
                int taken = INT_MIN;
                if (wt[ind] <= cap)
                    taken = val[ind] +
                      dp[ind][cap - wt[ind]];

                // Store the best of both
                  choices
                dp[ind][cap] = max(notTaken,
                  taken);
            }
        }

        // Return the maximum value possible
          for n items and capacity W
        return dp[n - 1][W];
    }
};

// Driver code
```

[OPTIMAL] Memorization Approach

Complexity: *Time Complexity: O(N*W),There are N*W states therefore at max 'N*W' new problems will*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Utility function for solving the
      unbounded knapsack problem using
      recursion + memoization
    int knapsackUtil(vector<int>& wt,
      vector<int>& val, int ind, int W,
      vector<vector<int>>& dp) {
        // Base case: if we're at the first
          item
        if (ind == 0) {
            // Calculate and return the
              maximum value for the given
              weight limit
            return (W / wt[0]) * val[0];
        }

        // If the result for this index and
          weight limit is already calculated,
          return it
        if (dp[ind][W] != -1)
            return dp[ind][W];

        // Calculate the maximum value without
          taking the current item
        int notTaken = knapsackUtil(wt, val,
          ind - 1, W, dp);

        // Initialize 'taken' with a very
          small value
        int taken = INT_MIN;

        // If current item can fit in the
          remaining capacity
        if (wt[ind] <= W)
            taken = val[ind] +
              knapsackUtil(wt, val, ind, W -
              wt[ind], dp);

        // Store and return the maximum of
          taking or not taking the current
          item
        return dp[ind][W] = max(notTaken,
          taken);
    }

    // Main function to call the utility
      function and set up DP table
    int unboundedKnapsack(int n, int W,
      vector<int>& val, vector<int>& wt) {
        // Create a DP table initialized with
          -1
        vector<vector<int>> dp(n,
          vector<int>(W + 1, -1));

        // Call the recursive function and
          return the answer
        return knapsackUtil(wt, val, n - 1, W,
          dp);
    }
};

// Driver code
```

Q348. Rod Cutting Problem | (DP - 24)

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Subsequences

Practice Link

Input : price = [1, 6, 8, 9, 10, 19, 7, 20], N = 8
Output :25
Explanation :Cut the rod into lengths of 2 and 6 for a total price of 6 + 19= 25.
Input :price = [1, 5, 8, 9], N = 4
Output :10
Explanation :Cut the rod into lengths of 2 and 2 for a total price of... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N * W),Because we have a nested loop iterating through all `n` ite*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to solve the rod cutting
      problem
    int rodCutting(vector<int>& price, int n)
      {
        // Initialize DP table with dimensions
          [n][n + 1]
        vector<vector<int>> dp(n,
          vector<int>(n + 1, 0));

        for(int length = 0; length <= n;
          length++){
            dp[0][length] = price[0]*length;
        }

        // Fill the DP table
        for (int ind = 1; ind < n; ++ind) {
            for (int length = 1; length <= n;
              ++length) {

                // Case when the piece is not
                  taken
                int notTaken = 0+dp[ind -
                  1][length];

                // Case when the piece is
                  taken
                int taken = INT_MIN;

                /* Length of the rod piece
                corresponding to the current
                  index*/
                int rodLength = ind + 1;

                // Check if the piece can be
                  taken
                if (rodLength <= length) {
                    taken = price[ind] +
                      dp[ind][length -
                      rodLength];
                }

                /* Update dp[ind][length] with
                  the maximum of
                including or not including the
                  current piece*/
                dp[ind][length] =
                  max(notTaken, taken);
            }
        }

        // Return the result
        return dp[n - 1][n];
    }
};
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(n × n) Each subproblem (i, length) is computed once. Space Complex*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
  public:
    // Function to compute maximum price
      obtainable by cutting the rod
    int rodCutting(vector<int> price, int n) {

        // Initialize a DP table where
          dp[i][j] stores the max price
        // using first i rod lengths to make
          total rod length j
        vector<vector<int>> dp(n,
          vector<int>(n + 1, 0));

        // Fill the base case: only using the
          first rod length (length = 1)
        for(int length = 0; length <= n;
          length++) {
            // We can use rod of length 1
              (index 0) multiple times
            dp[0][length] = length * price[0];
        }

        // Iterate through each rod piece
          starting from index 1
        for(int i = 1; i < n; i++) {
            for(int length = 0; length <= n;
              length++) {

                // Option 1: Do not cut with
                  this rod (copy from above)
                int notTake = dp[i -
                  1][length];

                // Option 2: Cut with current
                  rod length (reuse allowed)
                int take = INT_MIN;
                int rodLength = i + 1;

                // Check if this rod fits in
                  current length
                if(rodLength <= length) {
                    take = price[i] +
                      dp[i][length -
                      rodLength];
                }

                // Store the best value in the
                  DP table
                dp[i][length] = max(take,
                  notTake);
            }
        }

        // Return the answer from the last row
          and column (full rod length)
        return dp[n - 1][n];
    }
};

// Driver code
```

Q349. Longest common subsequence

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Input: str1 = "abcd", str2="bdef"
Output: "bd"
Explanation: LCS of two strings is "bd".

Input: str1 = "apple" str2 = "waffle"
Output: "ale"
Explanation: LCS of two strings is "ale".

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N*M) + O(N+M), we first build the entire DP table and then traverse it to reconstruct the L

```cpp
#include <bits/stdc++.h>
using namespace std;
// Solution class for LCS with reconstruction
  using tabulation
class Solution {
public:
    // Function to return the LCS string of
      text1 and text2
    string longestCommonSubsequence(string
      &text1, string &text2) {
        int n = text1.size();
        int m = text2.size();
        // Create DP table to store lengths of
          LCS for all substrings
        vector<vector<int>> dp(n + 1,
          vector<int>(m + 1, 0));
        // Fill dp table bottom-up
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                if (text1[i - 1] == text2[j -
                  1]) {
                    // Characters match:
                      increase length by 1
                    dp[i][j] = 1 + dp[i - 1][j
                      - 1];
                } else {
                    // Characters don't match:
                      take max of left and top
                    dp[i][j] = max(dp[i -
                      1][j], dp[i][j - 1]);
                }
            }
        }
        // Reconstruct LCS string from dp
          table
        int i = n, j = m;
        string lcs = "";
        // Traverse dp table from bottom-right
          to top-left
        while (i > 0 && j > 0) {
            if (text1[i - 1] == text2[j - 1])
              {
                // Characters match, add to
                  result and move diagonally
                lcs += text1[i - 1];
                i--;
                j--;
            } else if (dp[i - 1][j] > dp[i][j
              - 1]) {
                // Move up if top cell has
                  greater value
                i--;
            } else {
                // Move left otherwise
                j--;
            }
        }
        // Reverse string since it was built
          backwards
        reverse(lcs.begin(), lcs.end());
        return lcs;
    }
};
```

Q350. Print Longest Common Subsequence | (DP - 26)

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Input: str1 = "abcd", str2="bdef"
Output: "bd"
Explanation: LCS of two strings is "bd".

Input: str1 = "apple" str2 = "waffle"
Output: "ale"
Explanation: LCS of two strings is "ale".

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N*M) + O(N+M), we first build the entire DP table and then traverse it to reconstruct the L

```cpp
#include <bits/stdc++.h>
using namespace std;
// Solution class for LCS with reconstruction
  using tabulation
class Solution {
public:
    // Function to return the LCS string of
      text1 and text2
    string longestCommonSubsequence(string
      &text1, string &text2) {
        int n = text1.size();
        int m = text2.size();
        // Create DP table to store lengths of
          LCS for all substrings
        vector<vector<int>> dp(n + 1,
          vector<int>(m + 1, 0));
        // Fill dp table bottom-up
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                if (text1[i - 1] == text2[j -
                  1]) {
                    // Characters match:
                      increase length by 1
                    dp[i][j] = 1 + dp[i - 1][j
                      - 1];
                } else {
                    // Characters don't match:
                      take max of left and top
                    dp[i][j] = max(dp[i -
                      1][j], dp[i][j - 1]);
                }
            }
        }
        // Reconstruct LCS string from dp
          table
        int i = n, j = m;
        string lcs = "";
        // Traverse dp table from bottom-right
          to top-left
        while (i > 0 && j > 0) {
            if (text1[i - 1] == text2[j - 1])
              {
                // Characters match, add to
                  result and move diagonally
                lcs += text1[i - 1];
                i--;
                j--;
            } else if (dp[i - 1][j] > dp[i][j
              - 1]) {
                // Move up if top cell has
                  greater value
                i--;
            } else {
                // Move left otherwise
                j--;
            }
        }
        // Reverse string since it was built
          backwards
        reverse(lcs.begin(), lcs.end());
        return lcs;
    }
};
```

Q351. Longest common substring

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Example 1:
Input: str1 = "abcde", str2 = "abfce"
Output: 2
Explanation: The longest common substring is "ab", which has a length of 2.

Example 2:
Input: str1 = "abcdxyz", str2 = "xyzabcd"
Output: 4
Explanation: The longest common substring is "abc... (cont.)

[BETTER] Space Optimized Approach

Complexity: *Time Complexity: O(n * m), where n is the length of str1 and m is the length of str2.*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    int longestCommonSubstr(string str1,
      string str2) {
        int n = str1.size();
        int m = str2.size();
        vector<int> prev(m+1, 0);
        vector<int> cur(m+1, 0);
        int ans = 0;
        // Loop through both strings
        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                // Characters match, increment
                  substring length
                if(str1[i-1] == str2[j-1]){
                    int val = 1 + prev[j-1];
                    cur[j] = val;
                    ans = max(ans, val);
                }
                else{
                    cur[j] = 0;
                }
            }
            prev = cur;
        }
        // Return the length of Longest Common
          Substring
        return ans;
    }
};
```

[OPTIMAL] Tabulation Approach

Complexity: *Time Complexity: O(n * m), where n is the length of str1 and m is the length of str2.*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    /* Function to find the length of
    the Longest Common Substring (LCS) */
    int longestCommonSubstr(string str1,
      string str2) {
        int n = str1.size();
        int m = str2.size();

        // Initialize a 2D DP table with
          dimensions
        vector<vector<int>> dp(n + 1,
          vector<int>(m + 1, 0));

        // Initialize the answer variable
        int ans = 0;

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                // Characters match, increment
                  substring length
                if (str1[i - 1] == str2[j -
                  1]) {
                    dp[i][j] = 1 + dp[i - 1][j
                      - 1];

                    /* Update the maximum
                      substring
                    length found so far */
                    ans = max(ans, dp[i][j]);
                } else {
                    /* Characters don't match,
                    substring length becomes 0
                      */
                    dp[i][j] = 0;
                }
            }
        }
        // Return the length of Longest Common
          Substring
        return ans;
    }
};
```

Q352. Longest palindromic subsequence

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Example 1:
Input: s = "eeeme"
Output: 4
Explanation: The longest palindromic subsequence is "eeee", which has a length of 4.

Example 2:
Input: s = "annb"
Output: 2
Explanation: The longest palindromic subsequence is "nn", which has a length of 2.

[BETTER] Space Optimized Approach

Complexity: *Time Complexity: O(N * M), where N is the length of the string and M is the length of*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution{
private:
    /* Function to calculate the length of
    the Longest Palindromic Subsequence*/
    int lcs(string s1, string s2) {
        int n = s1.size();
        int m = s2.size();

        /* Create two arrays to store the
        previous and current rows of DP
          values*/
        vector<int> prev(m + 1, 0), cur(m + 1,
          0);

        /* Base Case is covered as we have
        initialized the prev and cur to 0.*/

        for (int ind1 = 1; ind1 <= n; ind1++)
          {
            for (int ind2 = 1; ind2 <= m;
              ind2++) {
                if (s1[ind1 - 1] == s2[ind2 -
                  1])
                    cur[ind2] = 1 + prev[ind2
                      - 1];
                else
                    cur[ind2] =
                      max(prev[ind2], cur[ind2
                      - 1]);
            }
            // Update the prev array with
              current values
            prev = cur;
        }
        // The value at prev[m] contains
          length of LCS
        return prev[m];
    }
public:
    /* Function to calculate the length of
    the Longest Palindromic Subsequence*/
    int longestPalinSubseq(string s){
        // Store a reversed copy of the string
        string t = s;
        reverse(s.begin(), s.end());

        /* Call the LCS function to find the
        length of the Longest Common
          Subsequence*/
        return lcs(s, t);
    }
};
```

[OPTIMAL] Tabulation Approach

Complexity: *Time Complexity: O(N * M), where N is the length of the string and M is the length of*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution{
private:
    int func(string s1, string s2) {
        int n = s1.size();
        int m = s2.size();
        // Declare a 2D DP array to store
          length of the LCS
        vector<vector<int>> dp(n + 1,
          vector<int>(m + 1, -1));
        // Initialize first row and first
          column to 0
        for (int i = 0; i <= n; i++) {
            dp[i][0] = 0;
        }
        for (int i = 0; i <= m; i++) {
            dp[0][i] = 0;
        }
        // Fill in the DP array
        for (int ind1 = 1; ind1 <= n; ind1++)
          {
            for (int ind2 = 1; ind2 <= m;
              ind2++) {
                if (s1[ind1 - 1] == s2[ind2 -
                  1])
                    dp[ind1][ind2] = 1 +
                      dp[ind1 - 1][ind2 - 1];
                else
                    dp[ind1][ind2] =
                      max(dp[ind1 - 1][ind2],
                      dp[ind1][ind2 - 1]);
            }
        }
        // The value at dp[n][m] contains
          length of the LCS
        return dp[n][m];
    }
public:
    int longestPalinSubseq(string s){
        // Store a reversed copy of the string
        string t = s;
        reverse(s.begin(), s.end());
        return lcs(s, t);
    }
};
```

Q353. Minimum insertions to make string palindrome | DP-29

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Input: s = "abcaa"
Output: 2
Explanation: Insert 2 characters "c", and "b" to make "abcacba", which is a palindrome.

Input : s = "ba"
Output: 1
Explanation : Insert "a" at the beginning to make "aba", which is a palindrome.

[BETTER] Space Optimized Approach

Complexity: *Time Complexity: O(N*N), we fill the entire DP array of size N*N one by one.*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {

public:

    // Function to compute LCS using 2-row DP
    int lcs(string s1, string s2) {

        // Get sizes
        int n = s1.size();
        int m = s2.size();

        // Initialize prev and cur rows
        vector<int> prev(m + 1, 0), cur(m + 1,
          0);

        // Fill rows
        for (int ind1 = 1; ind1 <= n; ind1++)
          {
            for (int ind2 = 1; ind2 <= m;
              ind2++) {

                // If characters match
                if (s1[ind1 - 1] == s2[ind2 -
                  1])
                    cur[ind2] = 1 + prev[ind2
                      - 1];

                // Else take max from prev row
                  or left
                else
                    cur[ind2] =
                      max(prev[ind2], cur[ind2
                      - 1]);
            }

            // Move cur to prev
            prev = cur;
        }

        // Return result
        return prev[m];
    }

    // Function to compute LPS
    int longestPalindromeSubsequence(string s)
      {
        string t = s;
        reverse(t.begin(), t.end());
        return lcs(s, t);
    }

    // Function to compute min insertions
    int minInsertion(string s) {
        int n = s.size();
        int k =
          longestPalindromeSubsequence(s);
        return n - k;
    }
};
```

[OPTIMAL] Tabulation Approach

Complexity: *Time Complexity: O(N*N), we fill the entire DP array of size N*N one by one.*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    int lcs(string s1, string s2) {
        int n = s1.size();
        int m = s2.size();
        vector<vector<int>> dp(n + 1,
          vector<int>(m + 1, -1));
        for (int i = 0; i <= n; i++) {
            dp[i][0] = 0;
        }
        for (int i = 0; i <= m; i++) {
            dp[0][i] = 0;
        }
        for (int ind1 = 1; ind1 <= n; ind1++)
          {
            for (int ind2 = 1; ind2 <= m;
              ind2++) {
                if (s1[ind1 - 1] == s2[ind2 -
                  1])
                    dp[ind1][ind2] =
                        1 + dp[ind1 - 1][ind2
                          - 1];
                else
                    dp[ind1][ind2] =
                        max(dp[ind1 -
                          1][ind2],
                            dp[ind1][ind2 -
                              1]);
            }
        }
        return dp[n][m];
    }
    int longestPalindromeSubsequence(string s)
      {
        string t = s;
        reverse(s.begin(), s.end());
        return lcs(s, t);
    }
    int minInsertion(string s) {
        int n = s.size();
        int k =
          longestPalindromeSubsequence(s);
        return n - k;
    }
};
```

Q354. Minimum insertions or deletions to convert string A to B

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Input: str1 = "kitten", str2 = "sitting"
Output: 5
Explanation: To transform "kitten" to "sitting", delete "k" and insert "s" to get "sitten", then delete 'e' and insert "i" to get "sittin", and insert "g" at the end to get "sitting".

Input : str1 = "flaw",... (cont.)

[BETTER] Space Optimized Approach

Complexity: *Time Complexity: O(N*M), we fill the entire DP array of size N*M one by one.*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Class containing the solution
class Solution {
public:
    // Function to calculate minimum
      operations
    int minOperations(string s1, string s2) {
        // Get the lengths of both strings
        int n = s1.size(), m = s2.size();

        // Create two 1D DP arrays
        vector<int> prev(m + 1, 0), cur(m + 1,
          0);

        // Iterate through each character of
          s1
        for (int i = 1; i <= n; i++) {

            // Iterate through each character
              of s2
            for (int j = 1; j <= m; j++) {

                // If characters match,
                  increment LCS length
                if (s1[i - 1] == s2[j - 1])
                    cur[j] = 1 + prev[j - 1];

                // Else, take max from left or
                  top
                else
                    cur[j] = max(prev[j],
                      cur[j - 1]);
            }

            // Update prev row to current
            prev = cur;
        }

        // Calculate total operations
        int lcs = prev[m];
        return (n - lcs) + (m - lcs);
    }
};

// Main function
```

[OPTIMAL] Tabulation Approach

Complexity: *Time Complexity: O(N*M), we fill the entire DP array of size N*M one by one.*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find minimum
      insertions/deletions
    int minOperations(string s1, string s2) {
        // Get lengths of both strings
        int n = s1.length();
        int m = s2.length();

        // Create DP table of size (n+1) x
          (m+1)
        vector<vector<int>> dp(n + 1,
          vector<int>(m + 1, 0));

        // Fill the table using bottom-up
          approach
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                // If characters match, extend
                  LCS
                if (s1[i - 1] == s2[j - 1])
                    dp[i][j] = 1 + dp[i - 1][j
                      - 1];
                else
                    dp[i][j] = max(dp[i -
                      1][j], dp[i][j - 1]);
            }
        }

        // Get LCS length
        int lcs = dp[n][m];

        // Return total operations = deletions
          + insertions
        return (n - lcs) + (m - lcs);
    }
};
```

Q355. Shortest common supersequence

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Input : str1 = "mno", str2 = "nop"
Output :"mnop"
Explanation : The shortest common supersequence is "mnop". It contains "mno" as the first three characters and "nop" as the last three characters, thus including both strings as subsequences.
Input :str1 = "dynamic", str2 = "program"
Output : "dynprogramic"
Explanation :The shortest common supersequence is "dynprogramic". It includes all characters from both "dynamic" and "program", with minimal overlap. For example, "dynamic" appears as "dyn...a... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n * m), Filling the DP table takes O(n * m), and backtracking adds O(n + m), which is negli

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    string shortestSupersequence(string s1,
      string s2) {
        int n = s1.size();
        int m = s2.size();
        vector<vector<int>> dp(n + 1,
          vector<int>(m + 1, 0));
        for (int i = 0; i <= n; i++) {
            dp[i][0] = 0;
        }
        for (int i = 0; i <= m; i++) {
            dp[0][i] = 0;
        }
        for (int ind1 = 1; ind1 <= n; ind1++)
          {
            for (int ind2 = 1; ind2 <= m;
              ind2++) {
                if (s1[ind1 - 1] == s2[ind2 -
                  1])
                    dp[ind1][ind2] = 1 +
                      dp[ind1 - 1][ind2 - 1];
                else
                    dp[ind1][ind2] =
                      max(dp[ind1 - 1][ind2],
                      dp[ind1][ind2 - 1]);
            }
        }
        int i = n;
        int j = m;
        string ans = "";
        while (i > 0 && j > 0) {
            if (s1[i - 1] == s2[j - 1]) {
                ans += s1[i - 1];
                i--;
                j--;
            }
            else if (dp[i - 1][j] > dp[i][j -
              1]) {
                ans += s1[i - 1];
                i--;
            } else {
                ans += s2[j - 1];
                j--;
            }
        }
        while (i > 0) {
            ans += s1[i - 1];
            i--;
        }
        while (j > 0) {
            ans += s2[j - 1];
            j--;
        }
        reverse(ans.begin(), ans.end());
        return ans;
    }
};
```

Q356. Distinct subsequences

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Input: s = "axbxax", t = "axa"
Output: 2
Explanation: In the string "axbxax", there are two distinct subsequences "axa":
(a)(x)bx(a)x
(a)xb(x)(a)x

Input: s = "babgbag", t = "bag"
Output: 5
Explanation: In the string "babgbag", there are five distinct subsequences "bag":
(ba)(b)(ga)(g)
(ba)(bg)(ag)
(bab)(ga)(g)
(bab)(g)(ag)
(babg)(a)(g)

[OPTIMAL SOLUTION] General — Time Complexity: O(N * M), each state defined by character of string s and character of string t is computed o

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to count distinct subsequences
    int numDistinct(string s, string t) {
        // Length of source string
        int n = s.size();
        // Length of target string
        int m = t.size();

        // Create dp array of size m+1,
          initialize with 0
        vector<long long> dp(m + 1, 0);

        // Base case: Empty target string can
          always be formed
        // in exactly 1 way (by deleting all
          characters of s)
        dp[m] = 1;

        // Traverse source string from end to
          start
        for (int i = n - 1; i >= 0; i--) {
            // Create a copy of dp for current
              iteration
            vector<long long> curr = dp;

            // Traverse target string from end
              to start
            for (int j = m - 1; j >= 0; j--) {
                // If characters match, we can
                  either take it or skip it
                if (s[i] == t[j]) {
                    curr[j] = dp[j + 1] +
                      dp[j];
                }
                // Else we can only skip this
                  character of s
                else {
                    curr[j] = dp[j];
                }
            }

            // Update dp for next iteration
            dp = curr;
        }

        // Answer is number of ways to form
          entire t
        return (int)dp[0];
    }
};
```

Q357. Edit distance

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Example 1:
Input: start = "planet", target = "plan"
Output: 2
Explanation:
To transform "planet" into "plan", the following operations are required:
1. Delete the character 'e': "planet" -> "plan"
2. Delete the character 't': "plan" -> "plan"
Thus, a total of... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N × M), as There are two nested loops running for N and M*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the edit distance
  between two strings
int editDistance(string& S1, string& S2) {
    int n = S1.size();
    int m = S2.size();

    // Create a DP table to store edit
      distances
    vector<vector<int>> dp(n + 1,
      vector<int>(m + 1, 0));

    // Initialize the first row and column
    for (int i = 0; i <= n; i++) {
        dp[i][0] = i;
    }
    for (int j = 0; j <= m; j++) {
        dp[0][j] = j;
    }

    // Fill in the DP table
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (S1[i - 1] == S2[j - 1]) {
                // If the characters match, no
                  additional cost
                dp[i][j] = dp[i - 1][j - 1];
            } else {
                // Minimum of three choices:
                // 1. Replace the character at
                  S1[i-1] with S2[j-1]
                // 2. Delete the character at
                  S1[i-1]
                // 3. Insert the character at
                  S2[j-1] into S1
                dp[i][j] = 1 + min(dp[i - 1][j
                  - 1], min(dp[i - 1][j],
                  dp[i][j - 1]));
            }
        }
    }

    // The value at dp[n][m] contains the edit
      distance
    return dp[n][m];
}
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N*M), as There are two nested loops. Space Com*

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to calculate the edit distance
  between two strings
int editDistance(string& S1, string& S2) {
    int n = S1.size();
    int m = S2.size();

    // Create two arrays to store previous and
      current row of edit distances
    vector<int> prev(m + 1, 0);
    vector<int> cur(m + 1, 0);

    // Initialize the first row
    for (int j = 0; j <= m; j++) {
        prev[j] = j;
    }

    // Calculate edit distances row by row
    for (int i = 1; i <= n; i++) {
        cur[0] = i; // Initialize the first
          column of the current row
        for (int j = 1; j <= m; j++) {
            if (S1[i - 1] == S2[j - 1]) {
                // If the characters match, no
                  additional cost
                cur[j] = prev[j - 1];
            } else {
                // Minimum of three choices:
                // 1. Replace the character at
                  S1[i-1] with S2[j-1]
                // 2. Delete the character at
                  S1[i-1]
                // 3. Insert the character at
                  S2[j-1] into S1
                cur[j] = 1 + min(prev[j - 1],
                  min(prev[j], cur[j - 1]));
            }
        }
        prev = cur; // Update the previous row
          with the current row
    }

    // The value at cur[m] contains the edit
      distance
    return cur[m];
}
```

Q358. Wildcard matching

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Strings

Practice Link

Input: S1 = "ab*cd", S2 = "abdefcd"
Output: true
Explanation: "ab" matches "ab", '*' matches "def", "cd" matches "cd". The strings match.
Input: S1 = "*a*b", S2 = "aaab"
Output: true
Explanation: First '*' matches "aa", 'a' matches 'a', second '*' matches empt... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*M),There are two nested loops Space Complexity: O(N*M),We are us*

```
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    bool isAllStars(string &S1, int i) {
        for (int j = 1; j <= i; j++) {
            if (S1[j - 1] != '*')
                return false;
        }
        return true;
    }
    bool wildcardMatching(string &S1, string
      &S2) {
        int n = S1.size();
        int m = S2.size();
        vector<vector<bool>> dp(n + 1,
          vector<bool>(m + 1, false));
        dp[0][0] = true;
        for (int j = 1; j <= m; j++) {
            dp[0][j] = false;
        }
        for (int i = 1; i <= n; i++) {
            dp[i][0] = isAllStars(S1, i);
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                if (S1[i - 1] == S2[j - 1] ||
                  S1[i - 1] == '?') {
                    dp[i][j] = dp[i - 1][j -
                      1];
                }
                else if (S1[i - 1] == '*') {
                    dp[i][j] = dp[i - 1][j] ||
                      dp[i][j - 1];
                }
                else {
                    dp[i][j] = false;
                }
            }
        }
        return dp[n][m];
    }
};
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*M),There are N*M states therefore at max 'N*M' new problems will*

```
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to check if the substring
      S1[0..i] contains only '*'
    bool isAllStars(string &S1, int i) {
        // Loop from 0 to i to verify all
          characters are '*'
        for (int j = 0; j <= i; j++) {
            if (S1[j] != '*')
                return false;
        }
        return true;
    }
    // Recursive function with memoization to
      check wildcard matching
    bool wildcardMatchingUtil(string &S1,
      string &S2, int i, int j,
      vector<vector<int>> &dp) {
        // Base Case 1: Both strings are
          exhausted
        if (i < 0 && j < 0)
            return true;
        // Base Case 2: Pattern exhausted but
          text remains
        if (i < 0 && j >= 0)
            return false;
        // Base Case 3: Text exhausted but
          pattern may still have '*'
        if (j < 0 && i >= 0)
            return isAllStars(S1, i);
        // If already computed, return stored
          value
        if (dp[i][j] != -1)
            return dp[i][j];
        // If characters match or pattern has
          '?'
        if (S1[i] == S2[j] || S1[i] == '?')
            return dp[i][j] =
              wildcardMatchingUtil(S1, S2, i -
              1, j - 1, dp);
        // If pattern has '*', we have two
          choices:
        // 1. Treat '*' as matching empty
          sequence -> move pattern index i-1
        // 2. Treat '*' as matching one or
          more characters -> move text index
          j-1
        if (S1[i] == '*')
            return dp[i][j] =
              wildcardMatchingUtil(S1, S2, i -
              1, j, dp) ||
wildcardMatchingUtil(S1, S2, i, j - 1, dp);
        // If no match
        return dp[i][j] = false;
    }
    // Main function to be called from driver
      code
    bool wildcardMatching(string &S1, string
      &S2) {
        int n = S1.size();
        int m = S2.size();
        // DP array initialized to -1 to
          indicate uncomputed states
        vector<vector<int>> dp(n,
          vector<int>(m, -1));
        // Call the recursive utility function
          starting from the last indices
        return wildcardMatchingUtil(S1, S2, n
          - 1, m - 1, dp);
    }
};
// Driver Code
```

Q359. Best time to buy and sell stock

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on Stocks

Practice Link

Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note: That buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n),This is because we are iterating through the array of prices exactly once. There are no

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to calculate maximum profit using single pass
    int stockbuySell(vector<int>& prices) {
        // Initialize the minimum price to a large number
        int minPrice = INT_MAX;

        // Initialize the maximum profit to 0
        int maxProfit = 0;

        // Traverse each price in the array
        for (int price : prices) {
            // If current price is less than minPrice, update minPrice
            if (price < minPrice) {
                minPrice = price;
            }
            // Else calculate profit and update maxProfit if it's greater
            else {
                maxProfit = max(maxProfit, price - minPrice);
            }
        }

        // Return the maximum profit found
        return maxProfit;
    }
};

// Driver code
```

Q360. Best time to buy and sell stock II

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on Stocks

Practice Link

Example 1:

Array={7, 1, 5, 3, 6, 4}
N=6
MaxProfit=7 as ((5-1) + (6-3))
Explanation: Buy the stock on 1st day at price 1 and sell the stock on 2nd day at the price of 5. Then again buy stock at day 3 and sell the stock at 4

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*2). Reason: There are two nested loops that account for O(N*2) c*

```
#include <bits/stdc++.h>
using namespace std;
class StockProfit {
long getMaximumProfit(long *Arr, int n) {
    // Create a DP table to memoize results
    vector<vector<long>> dp(n + 1,
      vector<long>(2, -1));

    // Base condition: When we have no stocks
      left, the profit is 0.
    dp[n][0] = dp[n][1] = 0;

    long profit;

    // Loop through the array in reverse order
    for (int ind = n - 1; ind >= 0; ind--) {
        for (int buy = 0; buy <= 1; buy++) {
            if (buy == 0) { // We can buy the
              stock
                profit = max(0 + dp[ind +
                  1][0], -Arr[ind] + dp[ind +
                  1][1]);
            }

            if (buy == 1) { // We can sell the
              stock
                profit = max(0 + dp[ind +
                  1][1], Arr[ind] + dp[ind +
                  1][0]);
            }

            dp[ind][buy] = profit;
        }
    }

    // The maximum profit is stored in
      dp[0][0] after all calculations
    return dp[0][0];
}
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*2). There are N*2 states therefore at max 'N*2' new problems wil*

```
#include <bits/stdc++.h>
using namespace std;
class StockProfit{
long getAns(long *Arr, int ind, int buy, int
  n, vector<vector<long>> &dp) {
    // Base case: When we reach the end of the
      array, there are no more decisions to
      make.
    if (ind == n) {
        return 0;
    }

    // If the result for this state has
      already been calculated, return it
    if (dp[ind][buy] != -1) {
        return dp[ind][buy];
    }

    long profit = 0;

    if (buy == 0) { // We can buy the stock
        profit = max(0 + getAns(Arr, ind + 1,
          0, n, dp), -Arr[ind] + getAns(Arr,
          ind + 1, 1, n, dp));
    }

    if (buy == 1) { // We can sell the stock
        profit = max(0 + getAns(Arr, ind + 1,
          1, n, dp), Arr[ind] + getAns(Arr,
          ind + 1, 0, n, dp));
    }

    // Store the calculated profit in the DP
      table and return it
    return dp[ind][buy] = profit;
}

long getMaximumProfit(long *Arr, int n) {
    // Create a DP table to memoize results
    vector<vector<long>> dp(n, vector<long>(2,
      -1));

    if (n == 0) {
        return 0;
    }

    long ans = getAns(Arr, 0, 0, n, dp);
    return ans;
}
```

Q361. Best time to buy and sell stock III

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on Stocks

Practice Link

Example 1:
Array={3, 3, 5, 0, 0, 1, 4}
N=8
MaxProfit = 6 as ((3-0) + (4-1))
Explaination: Buy the stock on 3rd day at price 0 and sell the stock on 5th day at the price of 3. Then again buy stock at day 6 and sell the stock at 7th day at the price of 4

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*2*3) There are three nested loops that account for O(N*2*3) comp*

```
#include <bits/stdc++.h>
using namespace std;
class StockProfit{
int maxProfit(vector<int>& Arr, int n) {
    // Creating a 3D DP array of size
      [n+1][2][3] initialized to 0
    vector<vector<vector<int>>> dp(n + 1,
      vector<vector<int>>(2, vector<int>(3,
      0)));

    // Base case: dp array is already
      initialized to 0, covering the base
      case.

    for (int ind = n - 1; ind >= 0; ind--) {
        for (int buy = 0; buy <= 1; buy++) {
            for (int cap = 1; cap <= 2; cap++)
              {
                if (buy == 0) { // We can buy
                  the stock
                    dp[ind][buy][cap] = max(0
                      + dp[ind + 1][0][cap],
-Arr[ind] + dp[ind + 1][1][cap]);
                }

                if (buy == 1) { // We can sell
                  the stock
                    dp[ind][buy][cap] = max(0
                      + dp[ind + 1][1][cap],
Arr[ind] + dp[ind + 1][0][cap - 1]);
                }
            }
        }
    }

    // The result is stored in dp[0][0][2]
      which represents maximum profit after
      the final transaction.
    return dp[0][0][2];
}
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*2*3) , There are N*2*3 states therefore at max 'N*2*3' new probl*

```
#include <bits/stdc++.h>
using namespace std;
class StockProfit{
int getAns(vector<int>& Arr, int n, int ind,
  int buy, int cap,
  vector<vector<vector<int>>>& dp) {
    // Base case: If we reach the end of the
      array or run out of allowed
      transactions, return 0.
    if (ind == n || cap == 0)
        return 0;

    // If the result is already calculated,
      return it.
    if (dp[ind][buy][cap] != -1)
        return dp[ind][buy][cap];

    int profit;

    if (buy == 0) { // We can buy the stock
        profit = max(0 + getAns(Arr, n, ind +
          1, 0, cap, dp),
                     -Arr[ind] + getAns(Arr,
                       n, ind + 1, 1, cap,
                       dp));
    }

    if (buy == 1) { // We can sell the stock
        profit = max(0 + getAns(Arr, n, ind +
          1, 1, cap, dp),
                     Arr[ind] + getAns(Arr, n,
                       ind + 1, 0, cap - 1,
                       dp));
    }

    // Store and return the calculated profit.
    return dp[ind][buy][cap] = profit;
}

int maxProfit(vector<int>& prices, int n) {
    // Creating a 3D DP array of size
      [n][2][3]
    vector<vector<vector<int>>> dp(n,
      vector<vector<int>>(2, vector<int>(3,
      -1)));

    // Call the recursive function to
      calculate maximum profit
    return getAns(prices, n, 0, 0, 2, dp);
}
```

Q362. Best time to buy and sell stock IV

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on Stocks

Practice Link

Example 1:
Array={3, 3, 5, 0, 0, 1, 4}
N=8
MaxProfit = 6 as ((3-0) + (4-1))
Explaination: Buy the stock on 3rd day at price 0 and sell the stock on 5th day at the price of 3. Then again buy stock at day 6 and sell the stock at 7th day at the price of 4

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*2*3), There are three nested loops that account for O(N*2*3) com*

```
#include <bits/stdc++.h>
using namespace std;
class StockBuySell{
int maximumProfit(vector& Arr, int n, int k) {
    // Creating a 3D DP array of size
      [n+1][2][k+1] initialized to 0
    vector>> dp(n + 1,
                                    vector>(2,
                                      vector(k
                                      + 1,
                                      0)));

    // Loop through the stock prices from the
      end to the beginning
    for (int ind = n - 1; ind >= 0; ind--) {
        for (int buy = 0; buy <= 1; buy++) {
            for (int cap = 1; cap <= k; cap++)
              {
                if (buy == 0) { // We can buy
                  the stock
                    dp[ind][buy][cap] = max(0
                      + dp[ind + 1][0][cap],
                                -Arr[ind] +
                                  dp[ind +
                                  1][1][cap]);
                }

                if (buy == 1) { // We can sell
                  the stock
                    dp[ind][buy][cap] = max(0
                      + dp[ind + 1][1][cap],
                                Arr[ind] +
                                  dp[ind +
                                  1][0][cap -
                                  1]);
                }
            }
        }
    }

    return dp[0][0][k];
}
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*2*3) ,There are N*2*3 states therefore at max ‘N*2*3’ new proble*

```
#include <bits/stdc++.h>
using namespace std;
class StockBuySell{
int getAns(vector& Arr, int n, int ind, int
  buy, int cap, vector>>& dp) {
    // Base case
    if (ind == n || cap == 0) return 0;

    // Check if the result is already computed
    if (dp[ind][buy][cap] != -1)
        return dp[ind][buy][cap];

    int profit;

    if (buy == 0) { // We can buy the stock
        profit = max(0 + getAns(Arr, n, ind +
          1, 0, cap, dp),
                     -Arr[ind] + getAns(Arr,
                       n, ind + 1, 1, cap,
                       dp));
    }

    if (buy == 1) { // We can sell the stock
        profit = max(0 + getAns(Arr, n, ind +
          1, 1, cap, dp),
                     Arr[ind] + getAns(Arr, n,
                       ind + 1, 0, cap - 1,
                       dp));
    }

    // Store the result in the DP array and
      return
    return dp[ind][buy][cap] = profit;
}

int maximumProfit(vector& prices, int n, int
  k) {
    // Creating a 3D DP array of size
      [n][2][k+1]
    vector>> dp(n,
                                    vector>(2,
                                      vector(k
                                      + 1,
                                      -1)));

    return getAns(prices, n, 0, 0, k, dp);
}
```

Q363. Best Time to Buy and Sell Stock with Cooldown

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on Stocks

Practice Link

Example 1:
Arr=[4,9, 0, 4, 10]
N=5
Max Profit=11 ((9-4)+(10-4))
Explanation: Buy the stock on 0th day at a price of 4 and sell on the 1st day with a price of 9, then again buy stock at 3rd day at a price of 4 and sell the stock at 4th day at the price of 10

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*2). There are two nested loops that account for O(N*2) complexit*

```
#include <bits/stdc++.h>
using namespace std;
class Solution{
int stockProfit(vector<int> &Arr) {
    int n = Arr.size();
    // Create a 2D DP array with dimensions
      (n+2) x 2, initialized to 0
    vector<vector<int>> dp(n + 2,
      vector<int>(2, 0));

    // Loop through the stock prices from the
      end to the beginning
    for (int ind = n - 1; ind >= 0; ind--) {
        for (int buy = 0; buy <= 1; buy++) {
            int profit;

            if (buy == 0) { // We can buy the
              stock
                profit = max(0 + dp[ind +
                  1][0], -Arr[ind] + dp[ind +
                  1][1]);
            }

            if (buy == 1) { // We can sell the
              stock
                profit = max(0 + dp[ind +
                  1][1], Arr[ind] + dp[ind +
                  2][0]);
            }

            // Store the computed profit in
              the DP array
            dp[ind][buy] = profit;
        }
    }

    return dp[0][0];
}
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*2). There are N*2 states therefore at max 'N*2' new problems wil*

```
#include <bits/stdc++.h>
using namespace std;
class Solution{
int getAns(vector<int> Arr, int ind, int buy,
  int n, vector<vector<int>> &dp) {
    // Base case: If we've reached the end of
      the array, return 0 profit.
    if (ind >= n) return 0;

    // Check if the result is already computed
    if (dp[ind][buy] != -1)
        return dp[ind][buy];

    int profit;

    if (buy == 0) { // We can buy the stock
        profit = max(0 + getAns(Arr, ind + 1,
          0, n, dp), -Arr[ind] + getAns(Arr,
          ind + 1, 1, n, dp));
    }

    if (buy == 1) { // We can sell the stock
        profit = max(0 + getAns(Arr, ind + 1,
          1, n, dp), Arr[ind] + getAns(Arr,
          ind + 2, 0, n, dp));
    }

    // Store the result in the DP array and
      return
    return dp[ind][buy] = profit;
}

int stockProfit(vector<int> &Arr) {
    int n = Arr.size();
    vector<vector<int>> dp(n, vector<int>(2,
      -1));

    int ans = getAns(Arr, 0, 0, n, dp);
    return ans;
}
```

Q364. Best time to buy and sell stock with transaction fees

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on Stocks

Practice Link

Example 1:
Array=[1, 3, 2, 8, 4, 9]
N=6
Fee=2
Max Profit=8 (5(8-1-2)+3(9-4-2))
Explanation: Buy the stock on 0th day at a price of 1 and sell on the 3rd day at a price 8 incurring a fee of 2, then again buy stock at 4th day at a price 4 and sell the stock at 5... (cont.)

[BETTER] Tabulation Approach

Complexity: *Time Complexity: O(N*2). There are two nested loops that account for O(N*2) complexit*

```
#include <bits/stdc++.h>
using namespace std;
class Solution{
int maximumProfit(int n, int fee, vector<int>&
  Arr) {
    if (n == 0) return 0; // Edge case: No
      stocks to trade.

    // Create a 2D DP array with dimensions
      (n+1) x 2, initialized to 0
    vector<vector<int>> dp(n + 1,
      vector<int>(2, 0));

    // Loop through the stock prices from the
      end to the beginning
    for (int ind = n - 1; ind >= 0; ind--) {
        for (int buy = 0; buy <= 1; buy++) {
            int profit;

            if (buy == 0) { // We can buy the
              stock
                profit = max(0 + dp[ind +
                  1][0], -Arr[ind] + dp[ind +
                  1][1]);
            }

            if (buy == 1) { // We can sell the
              stock
                profit = max(0 + dp[ind +
                  1][1], Arr[ind] - fee +
                  dp[ind + 1][0]);
            }

            dp[ind][buy] = profit;
        }
    }

    return dp[0][0]; // Return the maximum
      profit for buying.
}
```

[OPTIMAL] Memoization Approach

Complexity: *Time Complexity: O(N*2). There are N*2 states therefore at max 'N*2' new problems wil*

```
#include <bits/stdc++.h>
using namespace std;
class Solution{
int getAns(vector<int> &Arr, int ind, int buy,
  int n, int fee, vector<vector<int>> &dp) {
    // Base case: If we've reached the end of
       the array, return 0 profit.
    if (ind == n) return 0;

    // Check if the result is already computed
    if (dp[ind][buy] != -1)
        return dp[ind][buy];

    int profit;

    if (buy == 0) { // We can buy the stock
        profit = max(0 + getAns(Arr, ind + 1,
          0, n, fee, dp), -Arr[ind] +
          getAns(Arr, ind + 1, 1, n, fee,
          dp));
    }

    if (buy == 1) { // We can sell the stock
        profit = max(0 + getAns(Arr, ind + 1,
          1, n, fee, dp), Arr[ind] - fee +
          getAns(Arr, ind + 1, 0, n, fee,
          dp));
    }

    // Store the computed profit in the DP
      array
    return dp[ind][buy] = profit;
}

int maximumProfit(int n, int fee, vector<int>
  &Arr) {
    vector<vector<int>> dp(n, vector<int>(2,
      -1));

    if (n == 0) return 0; // Edge case: No
      stocks to trade.

    int ans = getAns(Arr, 0, 0, n, fee, dp);
    return ans;
}
```

Q365. Longest Increasing Subsequence

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on LIS

Practice Link

Example 1:
Input: nums = [10, 9, 2, 5, 3, 7, 101, 18]
Output: 4
Explanation: The longest increasing subsequence is [2, 3, 7, 101], and its length is 4.

Example 2:
Input: nums = [0, 1, 0, 3, 2, 3]
Output: 4
Explanation: The longest increasing subsequence is [0, 1, 2, 3], and its length is 4.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N log N), where N is the number of elements in the input array. This is because we iterate

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    /* Function to find the longest increasing
    subsequence in the given array */
    int LIS(vector<int>& nums) {
        int n = nums.size();

        vector<int> temp;
        temp.push_back(nums[0]);

        // Iterate on the elements of the array
        for(int i=1; i < n; i++) {

            // If the current element can be added to the subsequence
            if(nums[i] > temp.back())
                temp.push_back(nums[i]);

            // Otherwise
            else {
                // Index at which the current element must be placed
                int ind = lower_bound(temp.begin(), temp.end(), nums[i]) -
                          temp.begin();

                // Place the current element in the subsequence
                temp[ind] = nums[i];
            }

        }

        // Return the length of temporary subsequence created so far
        return temp.size();
    }
};
```

Q366. Print Longest Increasing Subsequence

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on LIS

Practice Link

Input: arr = [10, 22, 9, 33, 21, 50, 41, 60, 80]
Output: [10, 22, 33, 50, 60, 80]
Explanation: The LIS is [10, 22, 33, 50, 60, 80] and it is the lexicographically smallest.

Input: arr = [1, 3, 2, 4, 6, 5]
Output: [1, 3, 4, 6]
Explanation: Possible LIS sequences are [1, 3, 4, 6] and [1, 2, 4, 6]. Since [1, 3, 4, 6] is index-wise lexicographically smaller, it is the result.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N^2), nested loops are used to compute dp array and prev array.

```cpp
#include <bits/stdc++.h>
using namespace std;
// Solution class for LIS with reconstruction
  using DP
class Solution {
public:
    // Function to return the LIS as a
      sequence (vector)
    vector<int>
      longestIncreasingSubsequence(vector<int>
      &nums) {
        int n = nums.size();
        // DP array to store length of LIS
          ending at each index
        vector<int> dp(n, 1);
        // Array to store previous index of
          LIS element for reconstruction
        vector<int> prev(n, -1);
        // Compute LIS length for each index
        for (int i = 1; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (nums[j] < nums[i] && dp[j]
                  + 1 > dp[i]) {
                    // Update dp[i] and store
                      previous index
                    dp[i] = dp[j] + 1;
                    prev[i] = j;
                }
            }
        }
        // Find the index of maximum LIS
          length
        int maxLen = 0, maxIndex = 0;
        for (int i = 0; i < n; i++) {
            if (dp[i] > maxLen) {
                maxLen = dp[i];
                maxIndex = i;
            }
        }
        // Reconstruct LIS sequence
        vector<int> lisSeq;
        int curr = maxIndex;
        while (curr != -1) {
            lisSeq.push_back(nums[curr]);
            curr = prev[curr];
        }
        // Reverse sequence as it was built
          backwards
        reverse(lisSeq.begin(), lisSeq.end());
        return lisSeq;
    }
};
```

Q367. Longest Increasing Subsequence |(DP-43)

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on LIS

Practice Link

Example 1:
Input: nums = [10, 9, 2, 5, 3, 7, 101, 18]
Output: 4
Explanation: The longest increasing subsequence is [2, 3, 7, 101], and its length is 4.

Example 2:
Input: nums = [0, 1, 0, 3, 2, 3]
Output: 4
Explanation: The longest increasing subsequence is [0, 1, 2, 3], and its length is 4.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N log N), where N is the number of elements in the input array. This is because we iterate

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    /* Function to find the longest increasing
    subsequence in the given array */
    int LIS(vector<int>& nums) {
        int n = nums.size();

        vector<int> temp;
        temp.push_back(nums[0]);

        // Iterate on the elements of the array
        for(int i=1; i < n; i++) {

            // If the current element can be added to the subsequence
            if(nums[i] > temp.back())
                temp.push_back(nums[i]);

            // Otherwise
            else {
                // Index at which the current element must be placed
                int ind = lower_bound(temp.begin(), temp.end(), nums[i]) -
                          temp.begin();

                // Place the current element in the subsequence
                temp[ind] = nums[i];
            }

        }

        // Return the length of temporary subsequence created so far
        return temp.size();
    }
};
```

Q368. Largest Divisible Subset

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on LIS

Practice Link

Example 1:
Input: nums = [3, 5, 10, 20]
Output: [5, 10, 20]
Explanation: The subset [5, 10, 20] satisfies the divisibility condition:
- 10 % 5 == 0
- 20 % 10 == 0

Example 2:
Input: nums = [16, 8, 2, 4, 32]
Output: [2, 4, 8, 16, 32]
Explanation: The entire array forms a divisible subset since:
- 32 % 16 == 0
- 16 % 8 == 0
- 8 % 4 == 0
- 4 % 2 == 0

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n^2), where n is the number of elements in the input array. This is due to the nested loops

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to find the largest divisble
      subset
    vector<int>
      largestDivisibleSubset(vector<int> nums)
      {
        int n = nums.size(); // Size of the
          array
        sort(nums.begin(), nums.end());
        vector<int> ans; // To store the LDS
        vector<int> dp(n, 1); // DP array
        vector<int> parent(n); // Array to
          keep record of the parent
        // To store the index of last element
          in the LDS
        int lastIndex = 0;
        // To store the length of LDS
        int maxLen = 0;
        // Computing the DP array
        for(int i = 0; i < n; i++) {
            parent[i] = i; // Assign the
              parent to itself
            // For each previous index
            for(int prevInd = 0; prevInd < i;
              prevInd++) {
                // If the element at index i
                  can be included in the LDS
                  ending at index j
                if(nums[i] % nums[prevInd] ==
                  0 && dp[i] < dp[prevInd] +
                  1) {
                    dp[i] = dp[prevInd] + 1;
                      // Update the DP value
                    parent[i] = prevInd; //
                      Store the parent
                }
            }
            // If a longer LDS is found,
              update the values
            if(dp[i] > maxLen) {
                lastIndex = i;
                maxLen = dp[i];
            }
        }
        // Backtracking
        int i = lastIndex;
        // Until we reach an index which is
          its own parent
        while(parent[i] != i) {
            ans.push_back(nums[i]); // Add the
              element at current index
            i = parent[i];
        }
        ans.push_back(nums[i]); // Adding the
          last element
        // Return the computed result
        return ans;
    }
};
```

Q369. Longest String Chain

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on LIS

Practice Link

Example 1:
Input: words = ["a", "ab", "abc", "abcd", "abcde"]
Output: 5
Explanation:
The longest chain is ["a", "ab", "abc", "abcd", "abcde"].
Each word in the chain is formed by adding exactly one character to the previous word.

Example 2:
Input: words = ["dog", "dogs", "dots", "dot", "d", "do"]
Output: 4
Explanation:
The longest chain is ["d", "do", "dot", "dots"].
Each word is formed by inserting one character into the previous word.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n^2 * m) , where n is the number of words and m is the average length of the words. This is

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    // Function to return the length of the
      longest string chain possible
    int longestStringChain(vector<string>&
      words) {
        int n = words.size(); // Size of the
          array
        sort(words.begin(), words.end(),
          compare);
        vector<int> dp(n, 1); // DP array
        // To store the length of longest
          string chain
        int maxLen = 0;
        // Computing the DP array
        for(int i = 0; i < n; i++) {
            // For each previous index
            for(int j = 0; j < i; j++) {
                if(checkPossible(words[i],
                  words[j]) && dp[i] < dp[j] +
                  1) {
                    dp[i] = dp[j] + 1; //
                      Update the DP value
                }
            }
            // If a longer chain is found,
              update the values
            if(dp[i] > maxLen) maxLen = dp[i];
        }
        return maxLen;
    }
private:
    // Custom comparator function
    bool static compare(string &s, string &t)
      {
        return s.size() < t.size();
    }
    // Function to check if the string can be
      added to the chain
    bool checkPossible(string &s, string &t) {
        //  Base case
        if(s.size() != t.size() + 1) return
          false;
        int i = 0, j = 0; // Pointers
        // Traverse until the first string is
          exhausted
        while(i < s.size()) {
            // Move both pointers if
              characters matches
            if(j < t.size() && s[i] == t[j]) {
                i++, j++;
            }
            // Otherwise
            else {
                i++;
            }
        }
        // Return true if both the string gets
          exhausted
        if(i == s.size() && j == t.size())
          return true;
        return false; // Return false
          otherwise
    }
};
```

Q370. Longest Bitonic Subsequence

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on LIS

Practice Link

Example 1:
Input: arr = [5, 1, 4, 2, 3, 6, 8, 7]
Output: 6
Explanation:
The longest bitonic sequence is [1, 2, 3, 6, 8, 7] with a length of 6.
The sequence increases from 1 to 8 and then decreases at 7.

Example 2:
Input: arr = [10, 20, 30, 40, 50, 40, 30, 20]
Output: 8
Explanation: The entire array is bitonic, increasing up to 50 and then decreasing. The length of the bitonic sequence is 8.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n2), where n is the number of elements in the array. This is because we are using two neste

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to return the length of the
      longest bitonic sequence
    int LongestBitonicSequence(vector<int>&
      arr) {
        int n = arr.size(); // Size of the
          array

        // LIS_dp[i] stores the length of LIS
          ending at index i
        vector<int> LIS_dp(n, 1);

        // To store the length of longest
          bitonic sequence
        int maxLen = 0;

        // Computing the LIS DP array
        for(int i = 0; i < n; i++) {

            // For each previous index
            for(int prev = 0; prev < i;
              prev++) {

                /* If the element at index i
                  can be
                 included in the LIS ending at
                   prev index */
                if(arr[prev] < arr[i] &&
                  LIS_dp[i] < LIS_dp[prev] +
                  1) {
                    LIS_dp[i] = LIS_dp[prev] +
                      1; // Update the DP
                      value
                }
            }
        }

        // LDS_dp[i] stores the length of LDS
          starting from index i
        vector<int> LDS_dp(n, 1);

        // Computing the LDS DP array
        for(int i = n-1; i >= 0; i--) {

            // For each previous index
            for(int prev = n-1; prev > i;
              prev--) {

                /* If the element at index i
                  can be
                 included in the LIS ending at
                   prev index */
                if(arr[prev] < arr[i] &&
                  LDS_dp[i] < LDS_dp[prev] +
                  1) {
                    LDS_dp[i] = LDS_dp[prev] +
                      1; // Update the DP
                      value
                }
            }

            // Update the maximum possible
              length of Longest Bitonic
              Sequence
            maxLen = max(maxLen, LIS_dp[i] +
              LDS_dp[i] - 1);
        }

        return maxLen;
    }
};
```

Q371. Number of Longest Increasing Subsequences

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > DP on LIS

Practice Link

Input: nums = [1, 3, 5, 4, 7]
Output: 2
Explanation: There are two LIS of length 4: [1, 3, 4, 7] and [1, 3, 5, 7].

Input: nums = [2, 2, 2, 2, 2]
Output: 5
Explanation: All elements are the same, so every single element can form an LIS of length 1. There are 5 such subsequences.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N^2), for each element, we compare it with all previous elements to check if it can extend

```cpp
#include <bits/stdc++.h>
using namespace std;
// Class containing the method to find the
  count of Longest Increasing Subsequences
class Solution {
public:
    // Method to calculate the count of LIS
    int findNumberOfLIS(vector<int>& arr) {
        // Store the size of the input
          sequence
        int n = arr.size();
        // Store the length of the increasing
          subsequence ending at each position
        vector<int> dp(n, 1);
        // Store the count of increasing
          subsequences ending at each position
        vector<int> ct(n, 1);
        // Track the maximum subsequence
          length found so far
        int maxi = 1;
        // Loop through each position in the
          sequence
        for (int i = 0; i < n; i++) {
            // Compare the current position
              with all previous positions
            for (int j = 0; j < i; j++) {
                // If an increasing sequence
                  is found that is longer than
                  the current best for this
                  position
                if (arr[j] < arr[i] && dp[j] +
                  1 > dp[i]) {
                    // Update the best
                      subsequence length for
                      this position
                    dp[i] = dp[j] + 1;
                    // Reset the count to
                      match the source
                      subsequence count
                    ct[i] = ct[j];
                }
                // If an increasing sequence
                  of the same best length is
                  found
                else if (arr[j] < arr[i] &&
                  dp[j] + 1 == dp[i]) {
                    // Add the count of
                      subsequences from the
                      previous position
                    ct[i] = ct[i] + ct[j];
                }
            }
            // Update the global maximum
              subsequence length if needed
            maxi = max(maxi, dp[i]);
        }
        // Store the total number of
          subsequences of maximum length
        int countLIS = 0;
        // Sum the counts of all positions
          where the length equals the maximum
          length
        for (int i = 0; i < n; i++) {
            if (dp[i] == maxi) {
                countLIS += ct[i];
            }
        }
        // Return the total number of longest
          increasing subsequences
        return countLIS;
    }
};
```

Q372. Matrix chain multiplication

HARD

Topic: Dynamic Programming [Patterns and Problems] > MCM DP | Partition DP

Practice Link

Input : [40, 20, 30, 10, 30]
Output : 26000
Explanation : Best parenthesization is ((A1 x (A2 x A3)) x A4).
Input : [10, 20, 30, 40, 30]
Output : 30000
Explanation : Optimal parenthesization minimizes cost.

[BETTER] Memoization Approach

Complexity: *Time Complexity: O(n^3). There are O(n^2) subproblems (for each i, j), and for each w*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Recursive function with memoization
    int solve(vector<int>& arr, int i, int j,
      vector<vector<int>>& dp) {
        // Base case: only one matrix, no
          multiplication
        if (i == j) return 0;

        // If already computed, return from dp
        if (dp[i][j] != -1) return dp[i][j];

        // Initialize min cost as very large
        int minCost = INT_MAX;

        // Try every partition
        for (int k = i; k < j; k++) {
            int cost1 = solve(arr, i, k, dp);
            int cost2 = solve(arr, k + 1, j,
              dp);
            int costMultiply = arr[i - 1] *
              arr[k] * arr[j];

            int total = cost1 + cost2 +
              costMultiply;

            minCost = min(minCost, total);
        }

        // Store and return result
        return dp[i][j] = minCost;
    }

    // Function to start the memoization
      approach
    int matrixChainOrder(vector<int>& arr) {
        int n = arr.size();
        vector<vector<int>> dp(n,
          vector<int>(n, -1));
        return solve(arr, 1, n - 1, dp);
    }
};

// Driver code
```

[OPTIMAL] Recursive Approach

Complexity: *Time Complexity: O(2^n). This is because we try every possible parenthesization, whic*

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find the minimum
      multiplication cost
    int matrixChainOrder(vector<int>& arr, int
      i, int j) {
        // Base case: if only one matrix is
          present, no multiplication needed
        if (i == j) return 0;

        // Initialize minimum cost as very
          large
        int minCost = INT_MAX;

        // Try all possible partitions
        for (int k = i; k < j; k++) {
            // Cost of multiplying matrices
              from i to k
            int cost1 = matrixChainOrder(arr,
              i, k);

            // Cost of multiplying matrices
              from k+1 to j
            int cost2 = matrixChainOrder(arr,
              k + 1, j);

            // Cost of multiplying the two
              resulting matrices
            int costMultiply = arr[i - 1] *
              arr[k] * arr[j];

            // Total cost for this partition
            int total = cost1 + cost2 +
              costMultiply;

            // Update minimum
            minCost = min(minCost, total);
        }

        return minCost;
    }
};

// Driver code
```

Q373. Matrix Chain Multiplication | Bottom-Up|(DP-49)

HARD

Topic: Dynamic Programming [Patterns and Problems] > MCM DP | Partition DP

Practice Link

Example 1:
Input: nums = [10, 15, 20, 25]
Output: 8000
Explanation: There are two ways to multiply the chain:
- A1 * (A2 * A3) → requires 11250 multiplications.
- (A1 * A2) * A3 → requires 8000 multiplications.
Thus, the minimum number of multiplications required is 8000.

Example 2:
Input: nums = [4, 2, 3]
Output: 24
Explanation: There is only one way to multiply the chain: A1 * A2, which requires 24 multiplications.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N^3), where N is the number of matrices. We have three nested loops: one for the length of

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int matrixMultiplication(vector<int>& nums) {
        int n = nums.size();
        vector<vector<int>> dp(n, vector<int>(n, INT_MAX));

        // A single matrix needs no multiplication cost
        for (int i = 1; i < n; ++i) {
            dp[i][i] = 0;
        }

        // Filling the dp array
        for (int length = 2; length < n; ++length) { // length of the chain
            for (int i = 1; i <= n - length; ++i) {
                int j = i + length - 1;
                for (int k = i; k < j; ++k) {
                    int cost = dp[i][k] + dp[k + 1][j] + nums[i - 1] * nums[k]
                      * nums[j];
                    if (cost < dp[i][j]) {
                        dp[i][j] = cost;
                    }
                }
            }
        }

        // The result is in dp[1][n-1]
        return dp[1][n - 1];
    }
};
```

Q374. Minimum cost to cut the stick

HARD

Topic: Dynamic Programming [Patterns and Problems] > MCM DP | Partition DP

Practice Link

Input: n = 7, cuts = [1, 3, 4, 5]
Output: 16
Explanation: Rearranging the cuts to be [3, 5, 1, 4] for example will lead to a scenario with total cost = 16

Input: n = 7, cuts = [1, 3, 6]
Output: 14
Explanation: The optimal order for cutting the stick is [3, 1, 6].
The cost will be => 7 + 3 + 4 => 14.

[OPTIMAL SOLUTION] General — Time Complexity: O(C^3), for every state in our dp table i.e. c^2 states, we try all possible cuts between i a

```cpp
#include <bits/stdc++.h>
using namespace std;

// Solution class to compute minimum cost to
  cut a stick using tabulation (bottom-up DP)
class Solution {
public:
    // Function to calculate minimum total
      cost to cut the stick using tabulation
    int minimumCost(int n, int c, vector<int>
      &cuts) {
        // Add the two ends of the stick to
          the cuts array
        cuts.push_back(n);
        cuts.insert(cuts.begin(), 0);

        // Sort the cuts array
        sort(cuts.begin(), cuts.end());

        // Create a DP table initialized with
          0
        vector<vector<int>> dp(c + 2,
          vector<int>(c + 2, 0));

        // Fill the DP table for all possible
          segments
        // i traverses backward from c to 1
        for (int i = c; i >= 1; i--) {
            // j traverses forward from i to c
            for (int j = i; j <= c; j++) {
                int mini = INT_MAX;

                // Try making a cut at every
                  position between i and j
                for (int ind = i; ind <= j;
                  ind++) {
                    // Cost of making the
                      current cut plus costs
                      of left and right
                      subproblems
                    int ans = cuts[j + 1] -
                      cuts[i - 1] + dp[i][ind
                      - 1] + dp[ind + 1][j];

                    // Update minimum cost
                    mini = min(mini, ans);
                }

                // Store the minimum cost in
                  DP table
                dp[i][j] = mini;
            }
        }

        // The answer is the minimum cost to
          cut between indices 1 and c
        return dp[1][c];
    }
};
```

Q375. Burst balloons

HARD

Topic: Dynamic Programming [Patterns and Problems] > MCM DP | Partition DP

Practice Link

Example 1:
Input: N = 4, array[] = {3, 1, 5, 8}
Output: 167
Explanation: First, we will burst the second balloon with the value 1. Coins = 3*1*5 = 15. Second, we will burst the balloon with the value 5. Coins = 3*5*8 = 120. Third, we will burst the balloon wit... (cont.)

[BETTER] Memoization Approach

Complexity: *Time Complexity: O(N3), There are total N2 no. of states. And for each state, we are*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution{
// Recursive function to calculate the maximum
  coins obtained
int maxCoinsHelper(int i, int j, vector<int>
  &nums, vector<vector<int>> &dp) {
    if (i > j) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    int maxCoins = INT_MIN;

    // Iterate through each possible balloon
      to burst last
    for (int k = i; k <= j; k++) {
        // Calculate the coins obtained by
          bursting the k-th balloon last
        int coins = nums[i - 1] * nums[k] *
          nums[j + 1];

        // Recursively calculate the maximum
          coins for the remaining balloons
        int remainingCoins = maxCoinsHelper(i,
          k - 1, nums, dp) + maxCoinsHelper(k
          + 1, j, nums, dp);

        // Update the maximum coins
        maxCoins = max(maxCoins, coins +
          remainingCoins);
    }

    return dp[i][j] = maxCoins;
}

// Function to calculate the maximum coins
  obtained
int maxCoins(vector<int> &nums) {
    int n = nums.size();

    // Add 1 to the beginning and end of the
      nums array
    nums.insert(nums.begin(), 1);
    nums.push_back(1);

    // Create a DP array for memoization
    vector<vector<int>> dp(n + 2,
      vector<int>(n + 2, -1));

    // Call the helper function to compute the
      maximum coins
    return maxCoinsHelper(1, n, nums, dp);
}
```

[OPTIMAL] Recursive Approach

Complexity: *Time Complexity: Exponential Space Complexity: O(1). No extra space used*

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution{
// Recursive function to calculate the maximum
  coins obtained
int maxCoinsHelper(int i, int j, vector<int>
  &nums) {
    if (i > j) return 0;
    int maxCoins = INT_MIN;

    // Iterate through each possible balloon
      to burst last
    for (int k = i; k <= j; k++) {
        // Calculate the coins obtained by
          bursting the k-th balloon last
        int coins = nums[i - 1] * nums[k] *
          nums[j + 1];

        // Recursively calculate the maximum
          coins for the remaining balloons
        int remainingCoins = maxCoinsHelper(i,
          k - 1, nums) + maxCoinsHelper(k + 1,
          j, nums);

        // Update the maximum coins
        maxCoins = max(maxCoins, coins +
          remainingCoins);
    }

    return maxCoins;
}

// Function to calculate the maximum coins
  obtained
int maxCoins(vector<int> &nums) {
    int n = nums.size();

    // Add 1 to the beginning and end of the
      nums array
    nums.insert(nums.begin(), 1);
    nums.push_back(1);

    // Call the helper function to compute the
      maximum coins
    return maxCoinsHelper(1, n, nums);
}
```

Q376. Different Ways to Evaluate a Boolean Expression

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > MCM DP | Partition DP

Practice Link

Example 1:
Input: expression = "T|T&F"
Output: 1
Explanation: The only way to get the result as true is:
(T) | (T&F) = T|F = T

Example 2:
Input: expression = "F|T^F"
Output: 2
Explanation: There are 2 possible ways to get the result as true:
i. (F|T) ^ F =... (cont.)

[BETTER] Better Approach

Complexity: *Time Complexity: O(N × N × 2 × N) = O(N3), There are a total of 2 ×*

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int mod = 1000000007;
// Function to evaluate the number of ways to
  parenthesize the expression to get True or
  False
int f(int i, int j, int isTrue, string &exp,
  vector<vector<vector<ll>>> &dp) {
    // Base case 1: If i > j, it's an invalid
      expression, return 0.
    if (i > j) return 0;
    // Base case 2: If i and j are the same,
      evaluate the single character.
    if (i == j) {
        if (isTrue == 1) return exp[i] == 'T'
          ? 1 : 0;
        else return exp[i] == 'F' ? 1 : 0;
    }
    // If the result for this subproblem has
      been computed before, return it.
    if (dp[i][j][isTrue] != -1) return
      dp[i][j][isTrue];
    ll ways = 0;
    // Iterate through the expression to
      divide it into left and right
      subexpressions
    for (int ind = i + 1; ind <= j - 1; ind +=
      2) {
        // Recursively calculate the number of
          ways to make the left and right
          subexpressions true or false
        ll lT = f(i, ind - 1, 1, exp, dp);  //
          Number of ways to make the left
          expression true.
        ll lF = f(i, ind - 1, 0, exp, dp);  //
          Number of ways to make the left
          expression false.
        ll rT = f(ind + 1, j, 1, exp, dp);  //
          Number of ways to make the right
          expression true.
        ll rF = f(ind + 1, j, 0, exp, dp);  //
          Number of ways to make the right
          expression false.
        // Check the operator at the current
          index and update ways accordingly.
        if (exp[ind] == '&') {  // AND
          operator
            if (isTrue) ways = (ways + (lT *
              rT) % mod) % mod;
            else ways = (ways + (lF * rT) %
              mod + (lT * rF) % mod + (lF *
              rF) % mod) % mod;
        }
        else if (exp[ind] == '|') {  // OR
          operator
            if (isTrue) ways = (ways + (lF *
              rT) % mod + (lT * rF) % mod +
              (lT * rT) % mod) % mod;
            else ways = (ways + (lF * rF) %
              mod) % mod;
        }
        else {  // XOR operator
            if (isTrue) ways = (ways + (lF *
              rT) % mod + (lT * rF) % mod) %
              mod;
            else ways = (ways + (lF * rF) %
              mod + (lT * rT) % mod) % mod;
        }
    }
    // Store the result in the DP table and
      return it.
    return dp[i][j][isTrue] = ways;
}
// Function to start evaluating the expression
int evaluateExp(string &exp) {
    int n = exp.size();
    vector<vector<vector<ll>>> dp(n,
      vector<vector<ll>>(n, vector<ll>(2,
      -1)));  // DP table initialization
    return f(0, n - 1, 1, exp, dp);  // Start
      evaluation with isTrue set to true.
}
```

[OPTIMAL] Optimal Approach

Complexity: *Time Complexity: O(N*N*2 * N) ~ O(N3), There are a total of 2*

```cpp
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int mod = 1000000007;
// Function to evaluate the number of ways to
  parenthesize the expression to get True or
  False
int evaluateExp(string &exp) {
    int n = exp.size();
    // DP table to store the number of ways to
      evaluate the expression from i to j with
      isTrue (0 or 1)
    vector<vector<vector<ll>>> dp(n,
      vector<vector<ll>>(n, vector<ll>(2,
      0)));
    // Iterate through the string from right
      to left for bottom-up DP
    for (int i = n - 1; i >= 0; i--) {
        for (int j = 0; j <= n - 1; j++) {
            // Base case 1: If i > j, it's an
              invalid expression, so continue.
            if (i > j) continue;
            // Iterate through the possible
              values of isTrue (either 1 or 0)
            for (int isTrue = 0; isTrue <= 1;
              isTrue++) {
                // Base case 2: If i == j,
                  evaluate the single
                  character.
                if (i == j) {
                    if (isTrue == 1)
                      dp[i][j][isTrue] =
                      exp[i] == 'T';
                    else dp[i][j][isTrue] =
                      exp[i] == 'F';
                    continue;
                }
                // Recurrence logic for other
                  subexpressions
                ll ways = 0;
                for (int ind = i + 1; ind <= j
                  - 1; ind += 2) {
                    // Number of ways to make
                      the left and right
                      subexpressions true or
                      false
                    ll lT = dp[i][ind - 1][1];
                    ll lF = dp[i][ind - 1][0];
                    ll rT = dp[ind + 1][j][1];
                    ll rF = dp[ind + 1][j][0];
                    // Perform the operation
                      based on the operator at
                      the current index
                    if (exp[ind] == '&') {  //
                      AND operator
                        if (isTrue) ways =
                          (ways + (lT * rT) %
                          mod) % mod;
                        else ways = (ways +
                          (lF * rT) % mod +
                          (lT * rF) % mod +
                          (lF * rF) % mod) %
                          mod;
                    }
                    else if (exp[ind] == '|')
                      {  // OR operator
                        if (isTrue) ways =
                          (ways + (lF * rT) %
                          mod + (lT * rF) %
                          mod + (lT * rT) %
                          mod) % mod;
                        else ways = (ways +
                          (lF * rF) % mod) %
                          mod;
                    }
                    else {  // XOR operator
                        if (isTrue) ways =
                          (ways + (lF * rT) %
                          mod + (lT * rF) %
                          mod) % mod;
                        else ways = (ways +
                          (lF * rF) % mod +
                          (lT * rT) % mod) %
                          mod;
                    }
                }
                dp[i][j][isTrue] = ways;  //
                  Store the result in the DP
                  table
            }
        }
    }
    return dp[0][n - 1][1];  // Return the
      number of ways to evaluate the entire
      expression as true
}
```

Q377. Palindrome partitioning II

HARD

Topic: Dynamic Programming [Patterns and Problems] > MCM DP | Partition DP

Practice Link

Input: s = “bababcbadcede”
Output: 4
Explanation: If we do 4 partitions in the following way, each substring of the partition will be a palindrome.
bab | abcba | d | c | ede.

Input: s = "aab"
Output: 1
Explanation: If we do 1 partition in the following way, each substring of the partition will be a palindrome.
aa | b.

[OPTIMAL SOLUTION] General — Time Complexity: O(N^2), there are total N states and for every state we check whether the substring is a pali

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
private:
    // Check if substring s[i...j] is
      palindrome
    bool isPalindrome(const string& s, int i,
      int j) {
        while (i < j) {
            if (s[i] != s[j])
                return false;
            i++;
            j--;
        }
        return true;
    }

public:
    // Tabulation approach to find minimum
      cuts for palindrome partitioning
    int minCut(string s) {
        int n = (int)s.size();
        // dp[i] = min cuts needed for
          substring s[i...end]
        vector<int> dp(n + 1, 0);
        // Base case: no cuts needed beyond
          the last index
        dp[n] = -1;

        // Fill dp from end to start
        for (int i = n - 1; i >= 0; i--) {
            int minCuts = INT_MAX;

            // Check all substrings s[i...j]
            for (int j = i; j < n; j++) {
                if (isPalindrome(s, i, j)) {
                    // 1 cut plus cuts needed
                      after j
                    minCuts = min(minCuts, 1 +
                      dp[j + 1]);
                }
            }

            dp[i] = minCuts;
        }

        // dp[0] holds the min cuts needed for
          full string
        return dp[0];
    }
};
```

Q378. Partition Array for Maximum Sum

MEDIUM

Topic: Dynamic Programming [Patterns and Problems] > MCM DP | Partition DP

Practice Link

Input: arr = [1,15,7,9,2,5,10], k = 3
Output: 84
Explanation: The partition will be the following to get the largest sum: [1, 15, 7 | 9 | 2, 5, 10].
After replacing the elements of each subarray with its maximum, the array will look like this: [15,15,15,9,10,10,10] and the sum will be 84.

Input: arr = [2,2,2,2], k = 2
Output: 8
Explanation: Partition as [2,2 | 2,2]; each sub-array's max is 2. The array stays [2,2,2,2] and the sum is 8.

[OPTIMAL SOLUTION] General — Time Complexity: O(N*K), there are total N states and for every state we check all subarrays of length K.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to find maximum sum after partitioning
    int maxSumAfterPartitioning(vector<int>& arr, int k) {
        int n = (int)arr.size();

        // DP array to store max sum from index i to end
        vector<int> dp(n + 1, 0);

        // Iterate from right to left over the array
        for (int i = n - 1; i >= 0; i--) {
            int maxElem = 0;
            int maxSum = 0;

            // Check all partitions of length 1 to k starting at i
            for (int length = 1; length <= k && i + length <= n; length++) {
                // Update maximum element for current partition
                maxElem = max(maxElem, arr[i + length - 1]);

                // Calculate sum if partition ends here
                int currentSum = maxElem * length + dp[i + length];

                // Update maxSum if currentSum is greater
                maxSum = max(maxSum, currentSum);
            }

            // Store maximum sum for subarray starting at i
            dp[i] = maxSum;
        }

        // Return max sum achievable starting at index 0
        return dp[0];
    }
};
```

Q379. Maximum Rectangle Area with all 1's|(DP-55)

HARD

Topic: Dynamic Programming [Patterns and Problems] > DP on Squares

Practice Link

Input: matrix = [[1,0,1,0,0],[1,0,1,1,1],[1,1,1,1,1],[1,0,0,1,0]]

Output: 6
Explanation: The highlighted part depicts the rectangle with the largest area i.e. 6

Input: matrix = [[1]]
Output: 1
Explanation: In this case, there is only one rectangle with area 1.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N*(M+M)), where N = total no. of rows and M = total no. of columns. Space Complexity: O

```cpp
#include <bits/stdc++.h>
using namespace std;
// Class containing the solution functions
class Solution {
public:
    // Function to calculate largest rectangle
      area in a histogram
    int largestRectangleArea(vector<int>&
      heights) {
        // Stack to store indices of histogram
          bars
        stack<int> st;
        // Variable to store the max area
        int maxArea = 0;
        // Add a sentinel zero height at the
          end
        heights.push_back(0);
        // Iterate through all bars
        for (int i = 0; i < heights.size();
          i++) {
            // While current bar is smaller
              than the bar at stack top
            while (!st.empty() && heights[i] <
              heights[st.top()]) {
                // Pop the top height
                int height =
                  heights[st.top()];
                st.pop();
                // Calculate width
                int width = st.empty() ? i : i
                  - st.top() - 1;
                // Update max area
                maxArea = max(maxArea, height
                  * width);
            }
            // Push current index
            st.push(i);
        }
        return maxArea;
    }
    // Function to find maximal rectangle in
      binary matrix
    int maximalRectangle(vector<vector<char>>&
      matrix) {
        // Return 0 if matrix is empty
        if (matrix.empty()) return 0;
        // Get number of columns
        int m = matrix[0].size();
        // Height array to build histogram
          row-wise
        vector<int> height(m, 0);
        // Variable to track max rectangle
          area
        int maxArea = 0;
        // Traverse each row of the matrix
        for (auto& row : matrix) {
            // Update histogram based on
              current row
            for (int i = 0; i < m; i++) {
                if (row[i] == '1')
                  height[i]++;
                else height[i] = 0;
            }
            // Compute area for this histogram
            maxArea = max(maxArea,
              largestRectangleArea(height));
        }
        return maxArea;
    }
};
// Main driver function
```

Q380. Count Square Submatrices with All Ones|(DP-56)

EASY

Topic: Dynamic Programming [Patterns and Problems] > DP on Squares

Practice Link

Example 1:
Input: matrix = [[0,1,1,1],
[1,1,1,1],
[0,1,1,1]]
Output: 15
Explanation:
There are 10 squares of size 1.
There are 4 squares of size 2.
There is 1 square of size 3.
Total number of squares = 10 + 4 + 1 = 15.

Example 2:
Input: matrix = [[1,0,1],
[1,1,0],
[1,1,0]]
Output: 7
Explanation:
There are 6 squares of size 1.
There is 1 square of size 2.
Total number of squares = 6 + 1 = 7.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n × m) → each cell is processed once. Space Complexity: O(n × m) → DP t

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to count total square submatrices
  with all 1s
int countSquares(int n, int m,
  vector<vector<int>> &arr) {
    // DP table to store size of largest
      square ending at (i,j)
    vector<vector<int>> dp(n, vector<int>(m,
      0));

    // Initialize first row of DP table
    for (int j = 0; j < m; j++)
        dp[0][j] = arr[0][j];

    // Initialize first column of DP table
    for (int i = 0; i < n; i++)
        dp[i][0] = arr[i][0];

    // Fill the rest of the DP table
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < m; j++) {
            // If current cell is 0, no square
              ends here
            if (arr[i][j] == 0)
                dp[i][j] = 0;
            else {
                // Minimum of top, left,
                  top-left + 1
                dp[i][j] = 1 + min(dp[i -
                  1][j],
                                   min(dp[i -
                                     1][j -
                                     1],
                                     dp[i][j -
                                     1]));
            }
        }
    }

    // Sum up all values in DP table
    int sum = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            sum += dp[i][j];
        }
    }

    // Return total count of squares
    return sum;
}
```

Q381. Trie Implementation and Operations

HARD

Topic: Tries > Theory

Practice Link

Example 1:
Input:
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[] , ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
Output:
[null, null, true, false, true, null, true]
Explanation:
- Trie trie = new Trie()... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: Insertion: O(N), where N is the length of the word being inserted. This is because we need to

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    Node* links[26] = {nullptr};
    bool flag = false;
    bool containsKey(char ch) {
        return links[ch - 'a'] != nullptr;
    }
    void put(char ch, Node* node) {
        links[ch - 'a'] = node;
    }
    Node* get(char ch) {
        return links[ch - 'a'];
    }
    void setEnd() {
        flag = true;
    }
    bool isEnd() {
        return flag;
    }
    ~Node() {
        for (int i = 0; i < 26; i++) {
            if (links[i] != nullptr) {
                delete links[i];
                links[i] = nullptr;
            }
        }
    }
};
class Trie {
private:
    Node* root;
public:
    Trie() {
        root = new Node();
    }
    ~Trie() {
        delete root;
    }
    void insert(string word) {
        Node* node = root;
        for (char ch : word) {
            if (!node->containsKey(ch)) {
                node->put(ch, new Node());
            }
            node = node->get(ch);
        }
        node->setEnd();
    }
    bool search(string word) {
        Node* node = root;
        for (char ch : word) {
            if (!node->containsKey(ch)) {
                return false;
            }
            node = node->get(ch);
        }
        return node->isEnd();
    }
    bool startsWith(string prefix) {
        Node* node = root;
        for (char ch : prefix) {
            if (!node->containsKey(ch)) {
                return false;
            }
            node = node->get(ch);
        }
        return true;
    }
};
```

Q382. Trie Implementation and Advanced Operations

HARD

Topic: Tries > Problems

Practice Link

Example 1:
Input:
["Trie", "insert", "countWordsEqualTo", "insert", "countWordsStartingWith", "erase", "countWordsStartingWith"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["apple"], ["app"]]
Output:
[null, null, 1, null, 2, null, 1]
Explanatio... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), where N is the length of the word or prefix being processed. Each method (inserting a w

```
#include <bits/stdc++.h>
using namespace std;
struct Node {
    Node* links[26];
    int cntEndWith = 0;
    int cntPrefix = 0;
    bool containsKey(char ch) {
        return (links[ch - 'a'] !=
          NULL);
    }
    Node* get(char ch) {
        return links[ch - 'a'];
    }
    void put(char ch, Node* node) {
        links[ch - 'a'] = node;
    }
    void increaseEnd() {
        cntEndWith++;
    }
    void increasePrefix() {
        cntPrefix++;
    }
    void deleteEnd() {
        cntEndWith--;
    }
    void reducePrefix() {
        cntPrefix--;
    }
};
class Trie {
private:
    Node* root;
public:
    Trie() {
        root = new Node();
    }
    void insert(string word) {
        Node* node = root;
        for (int i = 0; i <
          word.size(); i++) {
            if
              (!node->containsKey(wo
              rd[i])) {
                node->put(word[i],
                  new Node());
            }
            node =
              node->get(word[i]);
            node->increasePrefix();
        }
        node->increaseEnd();
    }
    int countWordsEqualTo(string
      word) {
        Node* node = root;
        for (int i = 0; i <
          word.size(); i++) {
            if
              (node->containsKey(wor
              d[i])) {
                node =
                  node->get(word[i])
                  ;
            } else {
                return 0;
            }
        }
        return node->cntEndWith;
    }
    int
      countWordsStartingWith(string
      word) {
        Node* node = root;
        for (int i = 0; i <
          word.size(); i++) {
            if
              (node->containsKey(wor
              d[i])) {
                node =
                  node->get(word[i])
                  ;
            } else {
                return 0;
            }
        }
        return node->cntPrefix;
    }
    void erase(string word) {
        Node* node = root;
        for (int i = 0; i <
          word.size(); i++) {
            if
              (node->containsKey(wor
              d[i])) {
                node =
                  node->get(word[i])
                  ;
node->reducePrefix();
            } else {
                return;
            }
        }
        node->deleteEnd();
    }
};
```

Q383. Number of distinct substrings in a string

MEDIUM

Topic: Tries > Problems

Practice Link

Input: s = "aba"
Output: 6
Explanation: The distinct substrings are "a", "ab", "ba", "b", "aba", "".

Input: s = "abc"
Output: 7
Explanation: The distinct substrings are "a", "ab", "abc", "b", "bc", "c", "".

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(N^2), We consider all substrings of a string with length n. For each substring, inserting c

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    Node* links[26];
    bool flag = false;
    bool containsKey(char ch) {
        return (links[ch - 'a'] != NULL);
    }
    Node* get(char ch) {
        return links[ch - 'a'];
    }
    void put(char ch, Node* node) {
        links[ch - 'a'] = node;
    }
    void setEnd() {
        flag = true;
    }
    bool isEnd() {
        return flag;
    }
};
class Solution {
public:
    int countDistinctSubstrings(string &s) {
        Node* root = new Node();
        int cnt = 0;
        int n = s.size();
        for (int i = 0; i < n; i++) {
            Node* node = root;
            for (int j = i; j < n; j++) {
                if (!node->containsKey(s[j]))
                  {
                    node->put(s[j], new
                      Node());
                    cnt++;
                }
                node = node->get(s[j]);
            }
        }
        return cnt + 1;
    }
};
```

Q384. Maximum XOR of two numbers in an array

HARD

Topic: Tries > Problems

Practice Link

Input: nums = [3, 9, 10, 5, 1]
Output: 15
Explanation:
The maximum XOR value is 10 XOR 5 = 15.

Input: nums = [26, 49, 30, 15, 69]
Output: 116
Explanation:
The maximum XOR value is 69 XOR 49 = 116.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), each number is inserted and queried in the Trie in constant time (32 bits). Space C

```cpp
#include <bits/stdc++.h>
using namespace std;
class Node {
public:
    Node* links[2];
    bool containsKey(int bit) {
        return links[bit] != NULL;
    }
    Node* get(int bit) {
        return links[bit];
    }
    void put(int bit, Node* node) {
        links[bit] = node;
    }
};
class Solution {
public:
    Node* root;
    Solution() {
        root = new Node();
    }
    void insert(int num) {
        Node* node = root;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >> i) & 1;
            if (!node->containsKey(bit)) {
                node->put(bit, new Node());
            }
            node = node->get(bit);
        }
    }
    int getMaxXOR(int num) {
        Node* node = root;
        int maxXor = 0;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >> i) & 1;
            if (node->containsKey(1 - bit)) {
                maxXor |= (1 << i);
                node = node->get(1 - bit);
            } else {
                node = node->get(bit);
            }
        }
        return maxXor;
    }
    int findMaximumXOR(vector<int>& nums) {
        for (int num : nums) {
            insert(num);
        }
        int maxResult = 0;
        for (int num : nums) {
            maxResult = max(maxResult,
              getMaxXOR(num));
        }
        return maxResult;
    }
};
```

Q385. Maximum Xor with an element from an array

HARD

Topic: Tries > Problems

Practice Link

Input: Array: [3, 10, 5, 25, 2, 8], Queries: [(0, 1), (1, 2), (3, 3)]
Output: [-1, 3, 1]
Explanation: For the first query, there is no integer less than or equal to 1. For the second query, we can only choose 2. Therefore, maximum possible value is 1 XOR 2 =... (cont.)

[OPTIMAL SOLUTION] Approach — Time Complexity: O(32*N + Q(logQ) + 32*Q), for every number in array and query we traverse all its 32 bits to

```cpp
#include <bits/stdc++.h>
using namespace std;
struct Node {
    Node *links[2];
    bool containsKey(int ind) {
        return (links[ind] != NULL);
    }
    Node* get(int ind) {
        return links[ind];
    }
    void put(int ind, Node* node) {
        links[ind] = node;
    }
};
class Trie {
private:
    Node* root;
public:
    Trie() {
        root = new Node();
    }
    void insert(int num) {
        Node* node = root;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >> i) & 1;
            if (!node->containsKey(bit)) {
                node->put(bit, new Node());
            }
            node = node->get(bit);
        }
    }
    int findMax(int num) {
        Node* node = root;
        int maxNum = 0;
        for (int i = 31; i >= 0; i--) {
            int bit = (num >> i) & 1;
            if (node->containsKey(!bit)) {
                maxNum = maxNum | (1 << i);
                node = node->get(!bit);
            }
            else {
                node = node->get(bit);
            }
        }
        return maxNum;
    }
};
class Solution {
public:
    vector<int> maxXorQueries(vector<int>
      &arr,
vector<vector<int>> &queries) {
        vector<int> ans(queries.size(), 0);
        vector<pair<int, pair<int,int>>>
          offlineQueries;
        sort(arr.begin(), arr.end());
        int index = 0;
        for (auto &it: queries) {
            offlineQueries.push_back({it[1],
              {it[0], index++}});
        }
        sort(offlineQueries.begin(),
          offlineQueries.end());
        int i = 0;
        int n = arr.size();
        Trie trie;
        for (auto &it : offlineQueries) {
            while (i < n && arr[i] <=
              it.first) {
                trie.insert(arr[i]);
                i++;
            }
            if (i != 0)
                ans[it.second.second] =
                  trie.findMax(it.second.first
                  );
            else
                ans[it.second.second] = -1;
        }
        return ans;
    }
};
```

Q386. Minimum number of bracket reversals to make an expression balanced

HARD

Topic: Strings > Hard Problems

Practice Link

Input : s = ")(()((("
Output : 3
Explanation : One way to balance is: "((()))". There is no balanced sequence that can be formed in lesser reversals.

Input : s = "(()((()(()((("
Output : -1
Explanation : There's no way we can balance this sequence of braces.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(N), we traverse the string once processing each bracket in constant time.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to calculate the minimum number of reversals to balance the
      expression
    int minReversalsToBalance(string expression) {
        // Count of unmatched '(' brackets
        int openBrackets = 0;

        // Count of unmatched ')' brackets
        int closeBrackets = 0;

        // Traverse the string
        for (char ch : expression) {
            if (ch == '(') {
                // Consider '(' as unmatched for now
                openBrackets++;
            } else {
                if (openBrackets > 0) {
                    // Match this ')' with a previous '('
                    openBrackets--;
                } else {
                    // No matching '(' exists, so this ')' is unmatched
                    closeBrackets++;
                }
            }
        }

        // If total number of unmatched brackets is odd, return -1
        if ((openBrackets + closeBrackets) % 2 != 0) return -1;

        // Return minimum reversals required
        return (openBrackets + 1) / 2 + (closeBrackets + 1) / 2;
    }
};
```

Q387. Count and say

HARD

Topic: Strings > Hard Problems

Practice Link

Input: 4
Output: "1211"
Explanation:
countAndSay(1) = "1"
countAndSay(2) = RLE of "1" = "11"
countAndSay(3) = RLE of "11" = "21"
countAndSay(4) = RLE of "21" = "1211"

Input: 1
Output: "1"
Explanation: This is the base case.

[OPTIMAL SOLUTION] Approach — Time Complexity: O(n * 2^n), there can be n iterations, with string length of upto 2^n characters.

```cpp
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    // Function to generate the nth term of the count-and-say sequence
    string countAndSay(int n) {
        string result = "1";

        // Generate the sequence up to the nth term
        for (int i = 1; i < n; ++i) {
            string current = "";
            int count = 1;

            // Traverse the previous result
            for (int j = 1; j < result.size(); ++j) {
                // If current character matches previous, increment count
                if (result[j] == result[j - 1]) {
                    count++;
                } else {
                    // Append count and character to current result
                    current += to_string(count) + result[j - 1];
                    count = 1;
                }
            }

            // Append the last group
            current += to_string(count) + result.back();

            // Update result for next iteration
            result = current;
        }

        // Return the final result
        return result;
    }
};
```

Q388. KMP Algorithm or LPS array

HARD

Topic: Strings > Hard Problems

Practice Link

Input: text = "abracadabra", pattern = "abra"
Output: [0, 7]
Explanation: The pattern "abra" appears at the 0st and 7th positions in the text "abracadabra".

Input: text = "abcabcabc", pattern = "abc"
Output: [0, 3, 6]
Explanation: The pattern "abc" appears at the 0st, 3th, and 6th positions in the text "abcabcabc".

[OPTIMAL SOLUTION] General — Time Complexity: O(n + m), where n is the length of the text and m is the length of the pattern. This is becau

```cpp
#include <bits/stdc++.h>
using namespace std;
class Solution {
public:
    vector<int> computeLPS(string pattern) {
        vector<int> lps(pattern.length(), 0);
        int len = 0;
        int i = 1;
        while (i < pattern.length()) {
            if (pattern[i] == pattern[len]) {
                len++;
                lps[i] = len;
                i++;
            } else {
                if (len != 0) {
                    len = lps[len - 1];
                } else {
                    lps[i] = 0;
                    i++;
                }
            }
        }
        return lps;
    }
    vector<int> KMP(string text, string
      pattern) {
        vector<int> lps = computeLPS(pattern);
        vector<int> result;
        int i = 0;
        int j = 0;
        while (i < text.length()) {
            if (text[i] == pattern[j]) {
                i++;
                j++;
            }
            if (j == pattern.length()) {
                result.push_back(i - j);
                j = lps[j - 1];
            }
            else if (i < text.length() &&
              text[i] != pattern[j]) {
                if (j != 0) {
                    j = lps[j - 1];
                } else {
                    i++;
                }
            }
        }
        return result;
    }
};
```

Q389. Longest happy prefix

HARD

Topic: Strings > Hard Problems

Practice Link

Input: s = "abababab"
Output: "abab"
Explanation: "abab" is the longest prefix which is also suffix. They can overlap in the original string.

Input: s = "aaaa"
Output: "aaa"
Explanation: "aaa" is the longest proper prefix which is also a proper suffix in the string "aaaa".

[OPTIMAL SOLUTION] Optimal Approach — Time Complexity: O(n), we traverse the string once to compute the LPS array and use it to get the result.

```cpp
#include <bits/stdc++.h>
using namespace std;

// Solution class containing the method to find the longest happy prefix
class Solution {
public:
    // Function to find the longest happy prefix
    string longestPrefix(string s) {
        // Create a vector to store the prefix-suffix lengths
        vector<int> lps(s.length(), 0);

        // Pointer to track length of previous longest prefix suffix
        int len = 0;

        // Start from second character
        for (int i = 1; i < s.length(); i++) {
            // If characters match, increment length and store in lps
            if (s[i] == s[len]) {
                len++;
                lps[i] = len;
            }
            // If mismatch occurs and len is not zero, fallback using lps
            else if (len != 0) {
                len = lps[len - 1];
                i--; // retry with same i
            }
            // If no match and len is 0, keep lps as 0
        }

        // Return the prefix with length equal to the last lps value
        return s.substr(0, lps[s.length() - 1]);
    }
};

// Main function to test the solution
```