1.3sum closest

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

**Example 1:**

**Input:** nums = [-1,2,1,-4], target = 1

**Output:** 2

**Explanation:** The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

**Example 2:**

**Input:** nums = [0,0,0], target = 1

**Output:** 0

**Explanation:** The sum that is closest to the target is 0. (0 + 0 + 0 = 0).

**Constraints:**

* 3 <= nums.length <= 500
* -1000 <= nums[i] <= 1000
* -104 <= target <= 104

Program:

class Solution {

public:

    int threeSumClosest(vector<int>& nums, int target) {

        int a=nums.size();

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

        int ans=nums[0]+nums[1]+nums[2];

        for(int i=0;i<a-2;i++)

        {

            int left=i+1;

            int right=a-1;

            while(left<right)

            {

                int sum=nums[i]+nums[left]+nums[right];

                if (abs(target - sum) < abs(target - ans)) {

                    ans = sum;

                }

                //ans=min(ans,abs(target-sum));

                if(target>sum)

                left++;

                else

                right--;

            }

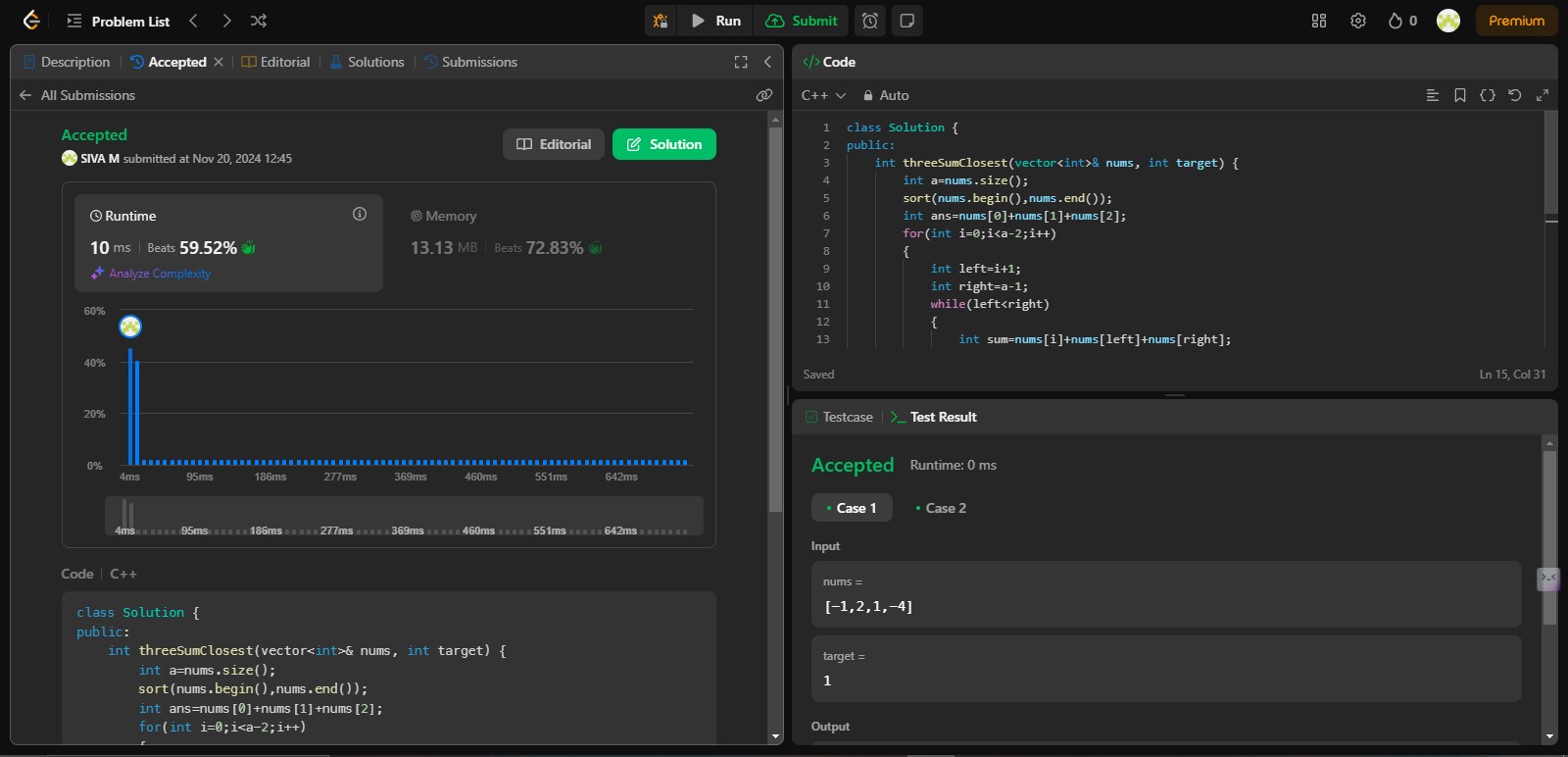
        }

        return ans;

    }

};

Output:



Time Complexity: O(N2)

2.Jump Game II

You are given a **0-indexed** array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

* 0 <= j <= nums[i] and
* i + j < n

Return *the minimum number of jumps to reach*nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

**Example 1:**

**Input:** nums = [2,3,1,1,4]

**Output:** 2

**Explanation:** The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

**Example 2:**

**Input:** nums = [2,3,0,1,4]

**Output:** 2

**Constraints:**

* 1 <= nums.length <= 104
* 0 <= nums[i] <= 1000
* It's guaranteed that you can reach nums[n - 1].

Program:

class Solution {

 public:

  int jump(vector<int>& nums) {

    int ans=0;

    int end=0;

    int farthest=0;

    int n=nums.size();

    for(int i=0;i<n-1;i++)

    {

        farthest=max(farthest,i+nums[i]);

        if(farthest>=n-1)

        {

            ans++;

            break;

        }

        if(i==end)

        {

            ans++;

            end=farthest;

        }

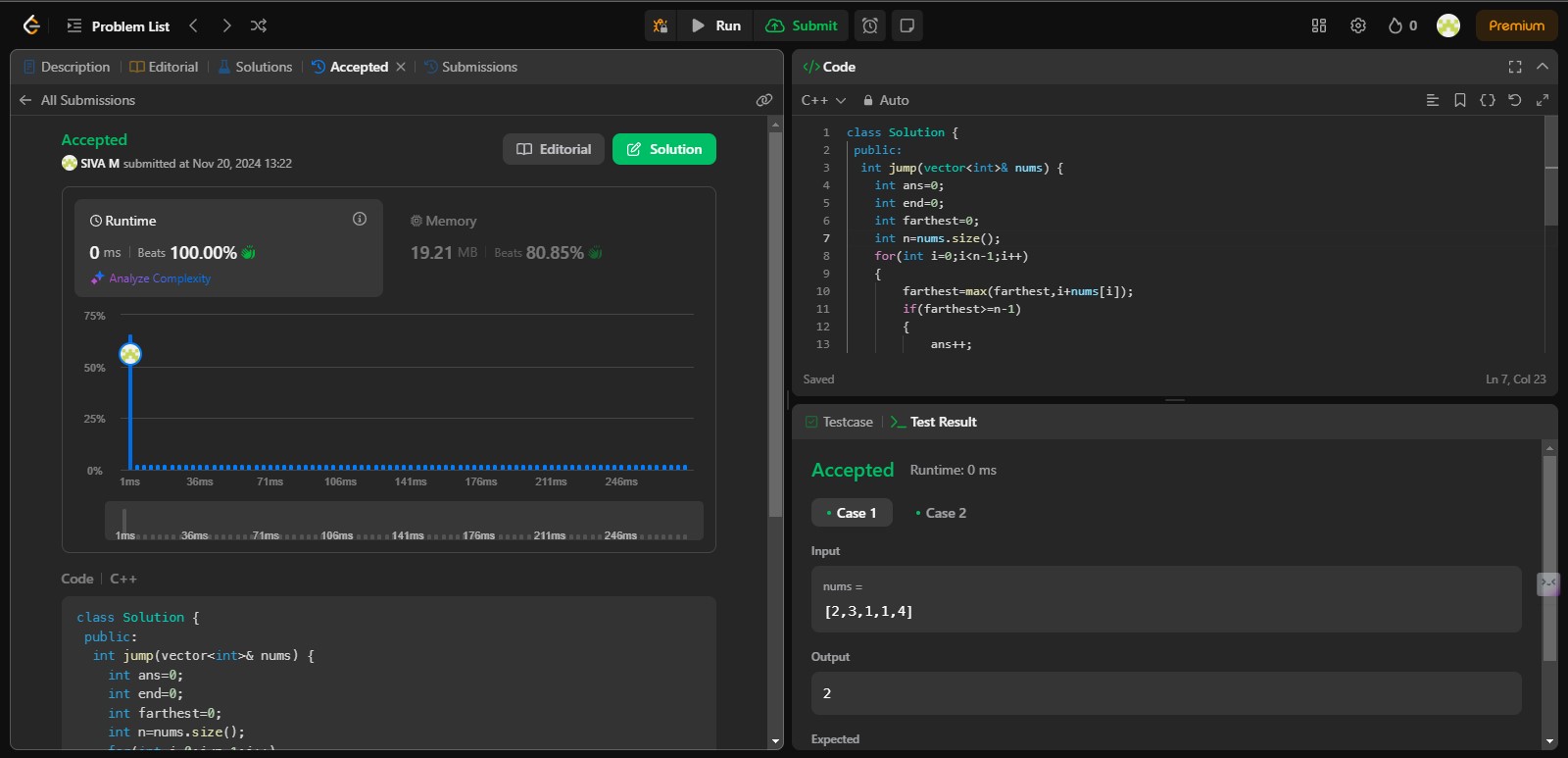
    }

    return ans;

    }

};

Output:



Time Complexity: O(N)

3.Group Anagrams

Given an array of strings strs, group the

anagrams

 together. You can return the answer in **any order**.

**Example 1:**

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

**Explanation:**

* There is no string in strs that can be rearranged to form "bat".
* The strings "nat" and "tan" are anagrams as they can be rearranged to form each other.
* The strings "ate", "eat", and "tea" are anagrams as they can be rearranged to form each other.

**Example 2:**

**Input:** strs = [""]

**Output:** [[""]]

**Example 3:**

**Input:** strs = ["a"]

**Output:** [["a"]]

**Constraints:**

* 1 <= strs.length <= 104
* 0 <= strs[i].length <= 100
* strs[i] consists of lowercase English letters.

Program:

class Solution {

public:

    vector<vector<string>> groupAnagrams(vector<string>& strs) {

        map<string,vector<string>> m;

        for(int i=0;i<strs.size();i++)

        {

            string p=strs[i];

            sort(p.begin(),p.end());

            m[p].push\_back(strs[i]);

        }

        vector<vector<string>> ans;

        for(auto k:m)

        {

            ans.push\_back(k.second);

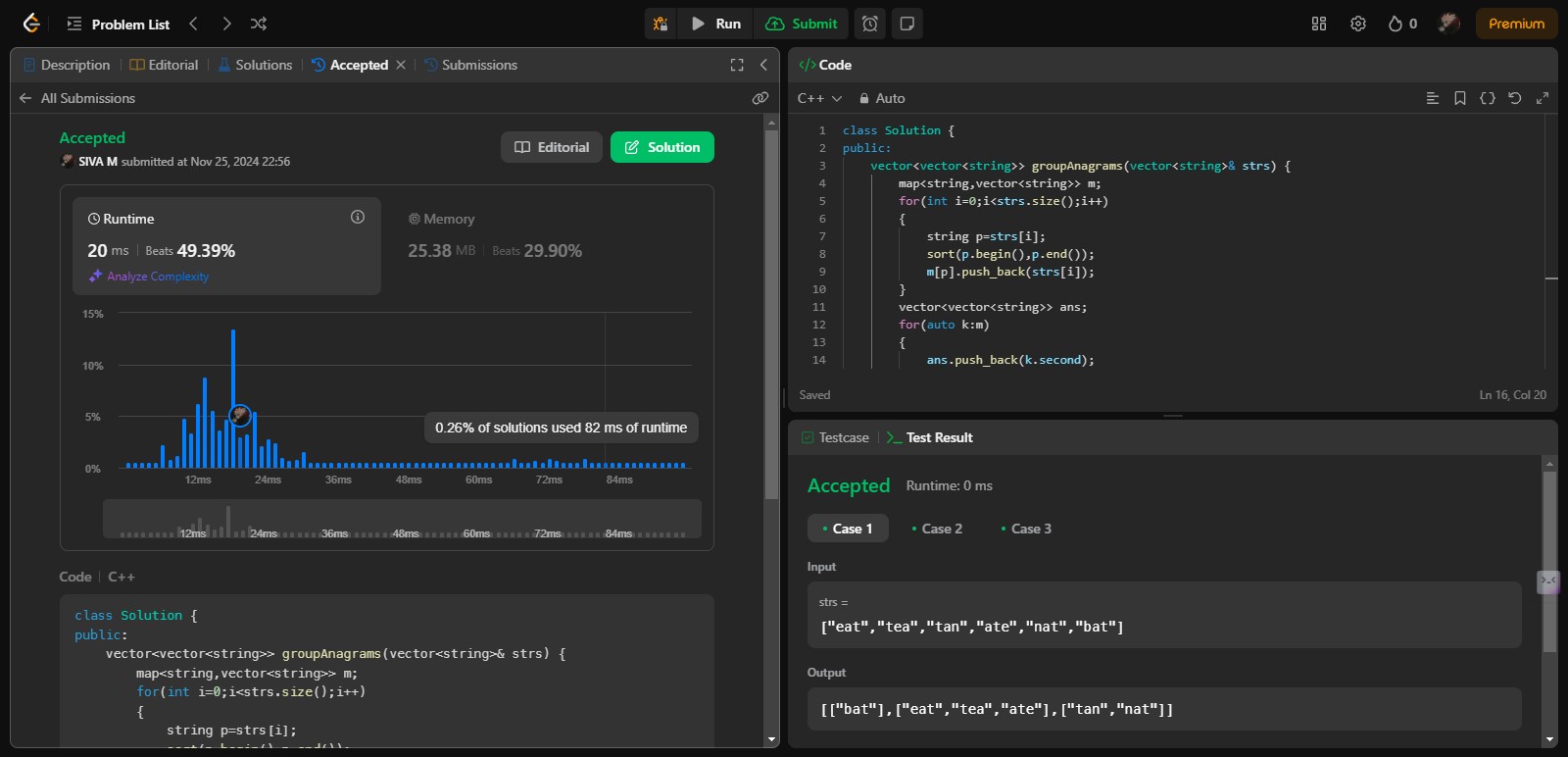
        }

        return ans;

    }

};

Output:



Time Complexity: O(n\*m\*log m)

Space complexity: O(n\*m)

4.Decode Ways

You have intercepted a secret message encoded as a string of numbers. The message is **decoded** via the following mapping:

"1" -> 'A'  
"2" -> 'B'  
...  
"25" -> 'Y'  
"26" -> 'Z'

However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes ("2" and "5" vs "25").

For example, "11106" can be decoded into:

* "AAJF" with the grouping (1, 1, 10, 6)
* "KJF" with the grouping (11, 10, 6)
* The grouping (1, 11, 06) is invalid because "06" is not a valid code (only "6" is valid).

Note: there may be strings that are impossible to decode.  
  
Given a string s containing only digits, return the **number of ways** to **decode** it. If the entire string cannot be decoded in any valid way, return 0.

The test cases are generated so that the answer fits in a **32-bit** integer.

**Example 1:**

**Input:** s = "12"

**Output:** 2

**Explanation:**

"12" could be decoded as "AB" (1 2) or "L" (12).

**Example 2:**

**Input:** s = "226"

**Output:** 3

**Explanation:**

"226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

**Example 3:**

**Input:** s = "06"

**Output:** 0

**Explanation:**

"06" cannot be mapped to "F" because of the leading zero ("6" is different from "06"). In this case, the string is not a valid encoding, so return 0.

**Constraints:**

* 1 <= s.length <= 100
* s contains only digits and may contain leading zero(s).

Program:

class Solution {

public:

    int numDecodings(string s) {

        if(s.length()==0 || s[0]=='0') return 0;

        if(s.length()==1) return 1;

        int cnt1=1,cnt2=1;

        for(int i=1;i<s.length();i++)

        {

            int d=s[i]-'0';

            int dd=(s[i-1]-'0')\*10+d;

            int cnt=0;

            if(d>0) cnt+=cnt2;

            if(dd>=10 && dd<=26)cnt+=cnt1;

            cnt1=cnt2;

            cnt2=cnt;

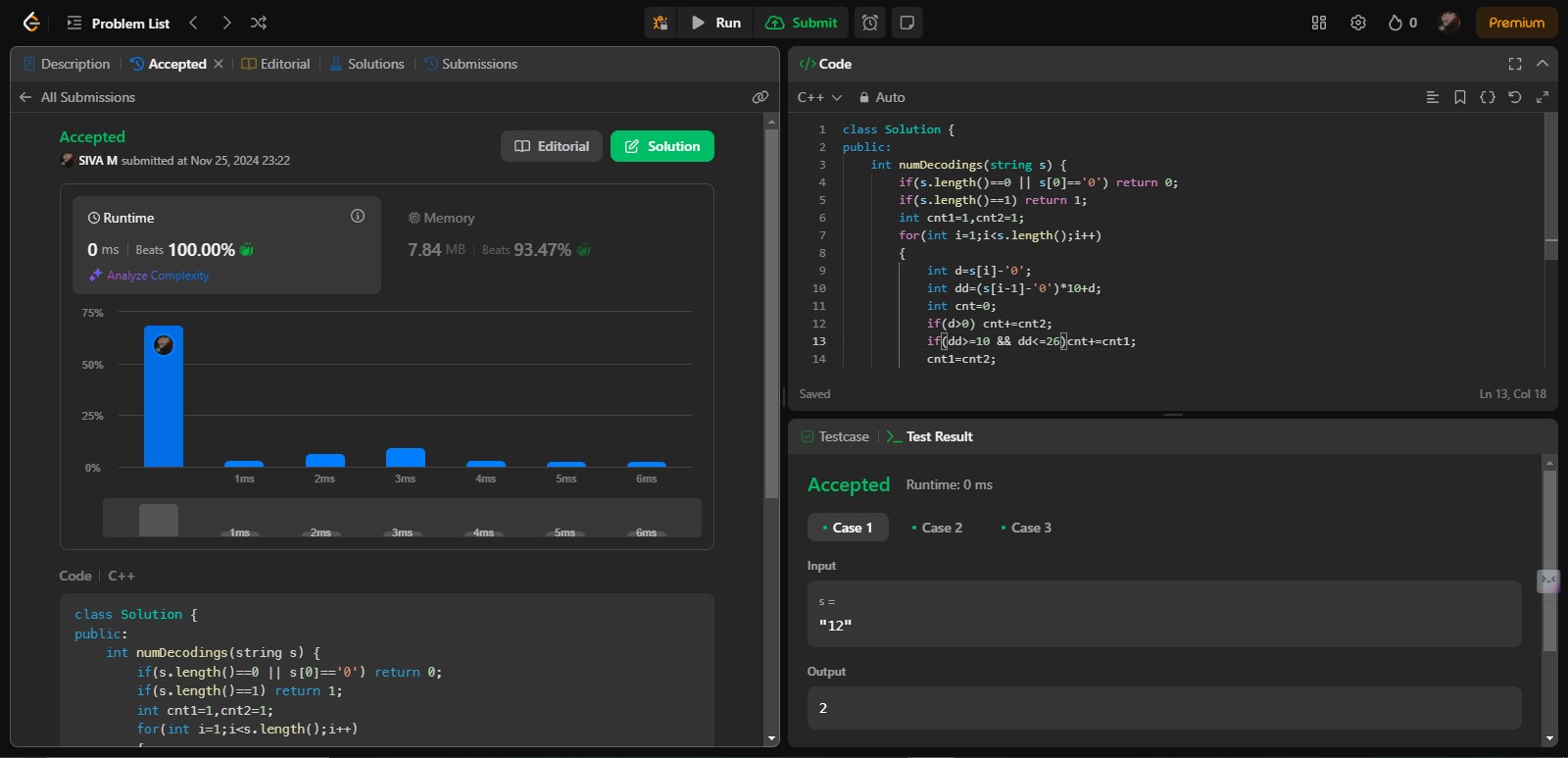
        }

        return cnt2;

    }

};

Output:



Time Complexity: O(n)

5.Best Time to Buy and Sell Stocks II

You are given an integer array prices where prices[i] is the price of a given stock on the ith day.

On each day, you may decide to buy and/or sell the stock. You can only hold at most one share of the stock at any time. However, you can buy it then immediately sell it on the same day.

Find and return the maximum profit you can achieve.

Example 1:

Input: prices = [7,1,5,3,6,4]

Output: 7

Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Total profit is 4 + 3 = 7.

Example 2:

Input: prices = [1,2,3,4,5]

Output: 4

Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.

Total profit is 4.

Example 3:

Input: prices = [7,6,4,3,1]

Output: 0

Explanation: There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

Constraints:

1 <= prices.length <= 3 \* 104

0 <= prices[i] <= 104

Program:

class Solution {

public:

    long f(int ind,int buy , vector<int>& prices,int n,vector<vector<int>> &dp)

    {

        if(ind==n) return 0;

        if(dp[ind][buy]!=-1) return dp[ind][buy];

        long profit=0;

        if (buy)

        {

            profit=max(-prices[ind]+f(ind+1,0,prices,n,dp),(0+f(ind+1,1,prices,n,dp)));

        }

        else

        {

            profit=max(prices[ind]+f(ind+1,1,prices,n,dp),(0+f(ind+1,0,prices,n,dp)));

        }

        dp[ind][buy]=profit;

        return dp[0][1];

    }

    int maxProfit(vector<int>& prices) {

        long n=prices.size();

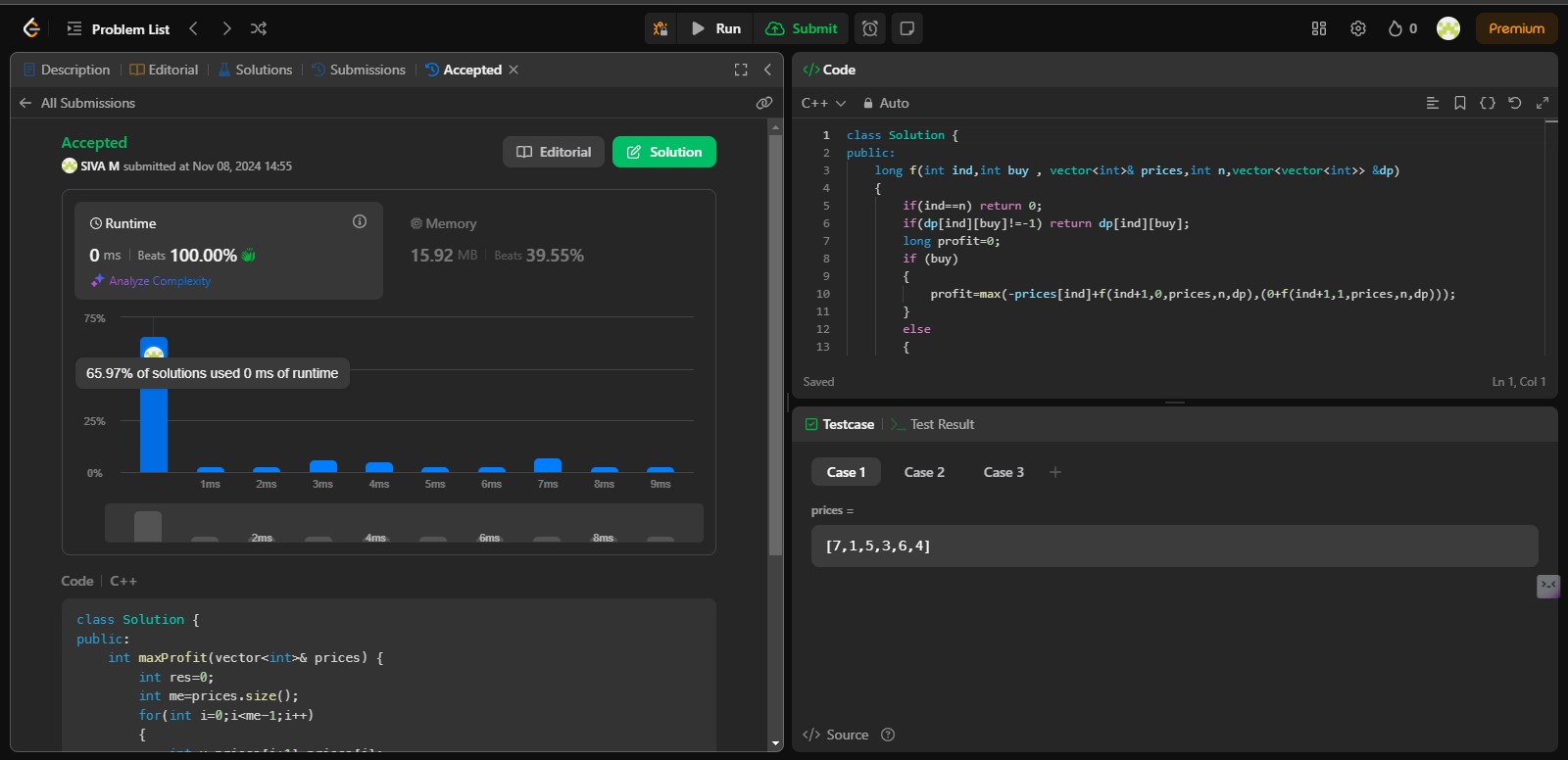
        vector<vector<long>> dp(n,vector<long>(2,-1));

        return f(0,1,prices,n,vector<vector<int>> \*dp) ;

    }

};

Output:



Time Complexity: O(N)

6.Number of islands

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example 1:**

**Input:** grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

**Output:** 1

**Example 2:**

**Input:** grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

["0","0","0","1","1"]

]

**Output:** 3

**Constraints:**

* m == grid.length
* n == grid[i].length
* 1 <= m, n <= 300
* grid[i][j] is '0' or '1'.

Program:

class Solution {

public:

    void dfs(vector<vector<char>>&grid,int i,int j)

    {

        int m=grid.size(),n=grid[0].size();

        if(i<0 || j<0 || i>=m || j>=n || grid[i][j]=='0')return;

        grid[i][j]='0';

        dfs(grid,i+1,j);

        dfs(grid,i-1,j);

        dfs(grid,i,j+1);

        dfs(grid,i,j-1);

    }

    int numIslands(vector<vector<char>>& grid) {

        int m=grid.size(),n=grid[0].size(),cnt=0;

        for(int i=0;i<m;i++)

        {

            for(int j=0;j<n;j++)

            {

                if(grid[i][j]=='1')

                {

                    ++cnt;

                    dfs(grid,i,j);

                }

            }

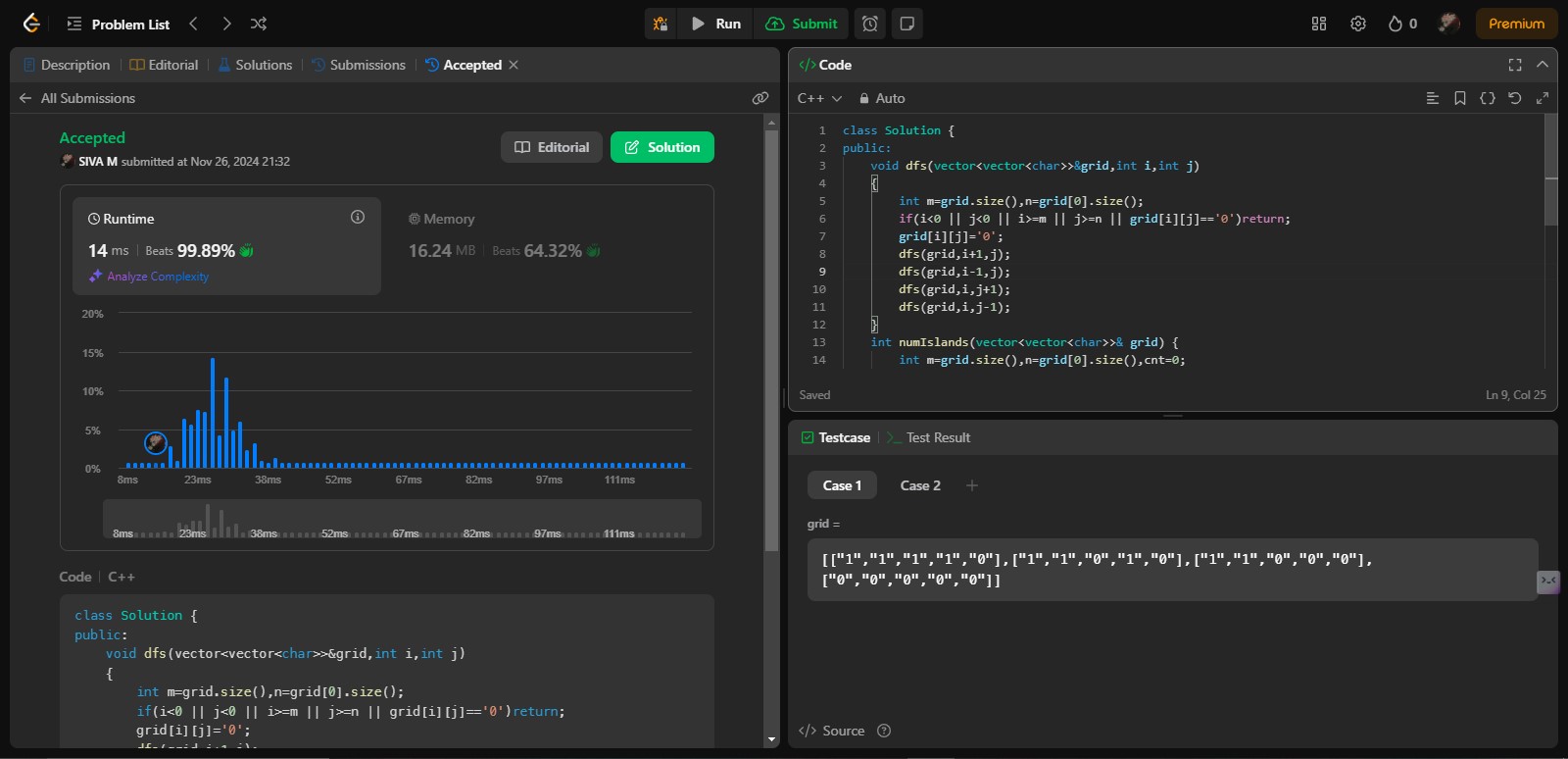
        }

        return cnt;

    }

};

Output:



Time Complexity: O(m\*n)

7.Quick Sort

Implement Quick Sort, a Divide and Conquer algorithm, to sort an array, **arr**[] in ascending order. Given an array, **arr**[], with starting index **low** and ending index **high**, complete the functions **partition()** and **quickSort()**. Use the last element as the pivot so that all elements less than or equal to the pivot come before it, and elements greater than the pivot follow it.

**Note**: The **low** and **high** are inclusive.

**Examples:**

**Input:** arr[] = [4, 1, 3, 9, 7]

**Output:** [1, 3, 4, 7, 9]  
**Explanation:** After sorting, all elements are arranged in ascending order.

**Input:** arr[] = [2, 1, 6, 10, 4, 1, 3, 9, 7]

**Output: [**1, 1, 2, 3, 4, 6, 7, 9, 10]  
**Explanation:** Duplicate elements (1) are retained in sorted order.

**Input:** arr[] = [5, 5, 5, 5]

**Output:** [5, 5, 5, 5]  
**Explanation:** All elements are identical, so the array remains unchanged.

**Constraints:**  
1 <= arr.size() <= 103  
1 <= arr[i] <= 104

Try more examples

Program:

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

/\* Function to print an array \*/

void printArray(const vector<int>& arr) {

for (int num : arr)

printf("%d ", num);

printf("\n");

}

// } Driver Code Ends

class Solution {

public:

void quickSort(vector<int>& arr, int low, int high) {

if(low<high)

{

int pivot=partition(arr,low,high);

quickSort(arr,low,pivot-1);

quickSort(arr,pivot+1,high);

}

}

public:

int partition(vector<int>& arr, int low, int high) {

int pivot=arr[high];

int i=low-1;

for(int j=low;j<=high-1;j++)

{

if(arr[j]<pivot)

{

i++;

swap(arr[i],arr[j]);

}

}

swap(arr[i+1],arr[high]);

return i+1;

}

};

//{ Driver Code Starts.

int main() {

int T;

scanf("%d", &T);

getchar(); // to consume newline after T

while (T--) {

vector<int> arr;

string input;

getline(cin, input);

stringstream ss(input);

int number;

while (ss >> number) {

arr.push\_back(number);

}

Solution ob;

ob.quickSort(arr, 0, arr.size() - 1);

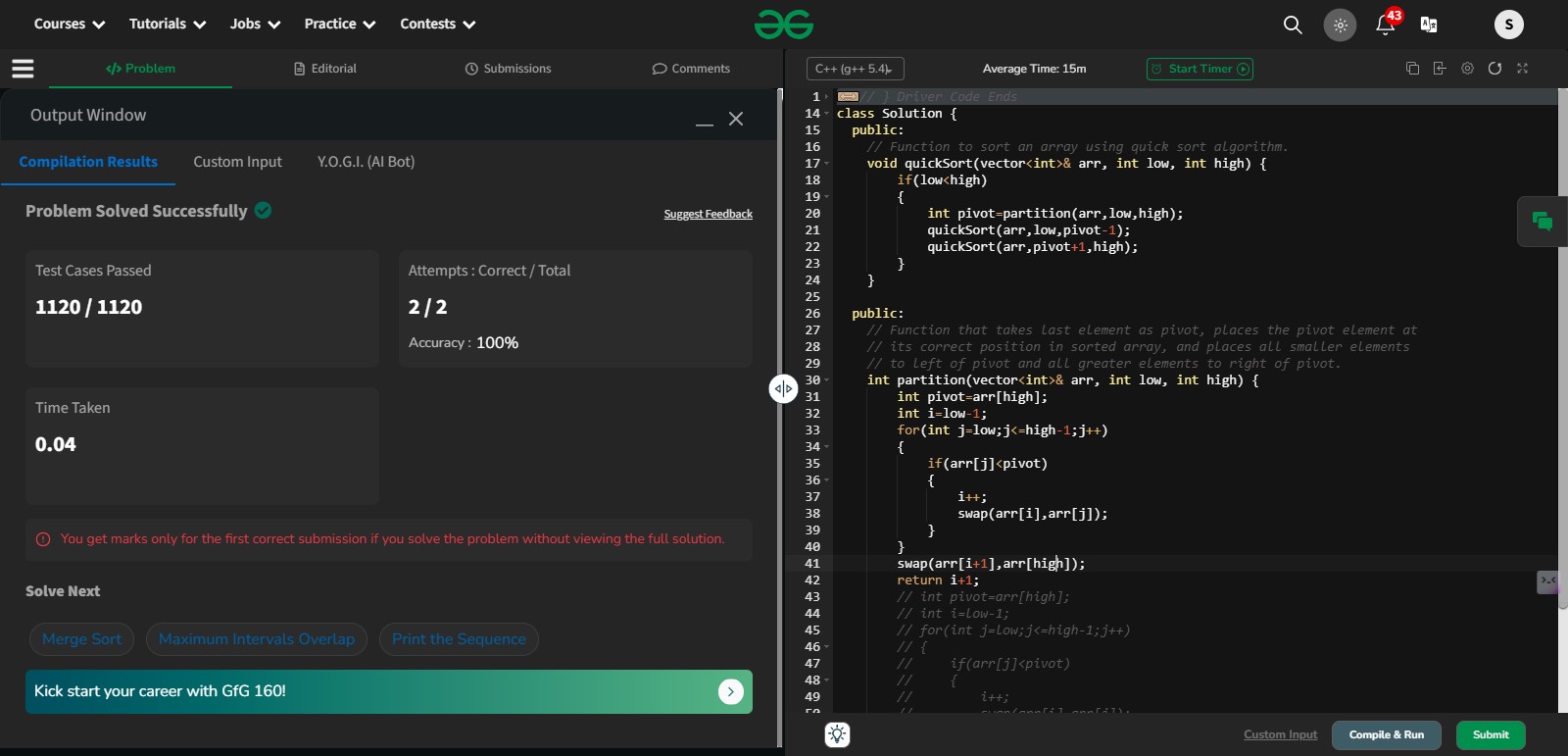
printArray(arr);

}

return 0;

}

// } Driver Code Ends  
  
Output:



Time Complexity: O(n log n)

Space Complexity: O(1)

8.Merge Sort

Given an array arr[], its starting position l and its ending position r. Sort the array using the merge sort algorithm.

**Examples:**

**Input:** arr[] = [4, 1, 3, 9, 7]

**Output:** [1, 3, 4, 7, 9]

**Input:** arr[] = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

**Output:** [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

**Input:** arr[] = [1, 3 , 2]

**Output:** [1, 2, 3]

**Constraints:**  
1 <= arr.size() <= 105  
1 <= arr[i] <= 105

Program:

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

class Solution {

public:

void ms(vector<int>&arr,int low,int high)

{

if(low==high) return;

int mid=(low+high)/2;

ms(arr,low,mid);

ms(arr,mid+1,high);

merge(arr,low,mid,high);

}

void merge(vector<int> &arr,int low,int mid,int high)

{

int left=low;

int right=mid+1;

vector<int> temp;

while(left<=mid && right<=high)

{

if(arr[left]<=arr[right])

{

temp.push\_back(arr[left]);

left++;

}

else

{

temp.push\_back(arr[right]);

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];

}

}

void mergeSort(vector<int>& arr, int l, int r) {

ms(arr,l,r);

}

};

//{ Driver Code Starts.

int main() {

string ts;

getline(cin, ts);

int t = stoi(ts);

while (t--) {

vector<int> arr;

string input;

getline(cin, input);

stringstream ss(input);

int number;

while (ss >> number) {

arr.push\_back(number);

}

Solution obj;

obj.mergeSort(arr, 0, arr.size() - 1);

for (int i = 0; i < arr.size(); i++) {

cout << arr[i] << " ";

}

cout << endl;

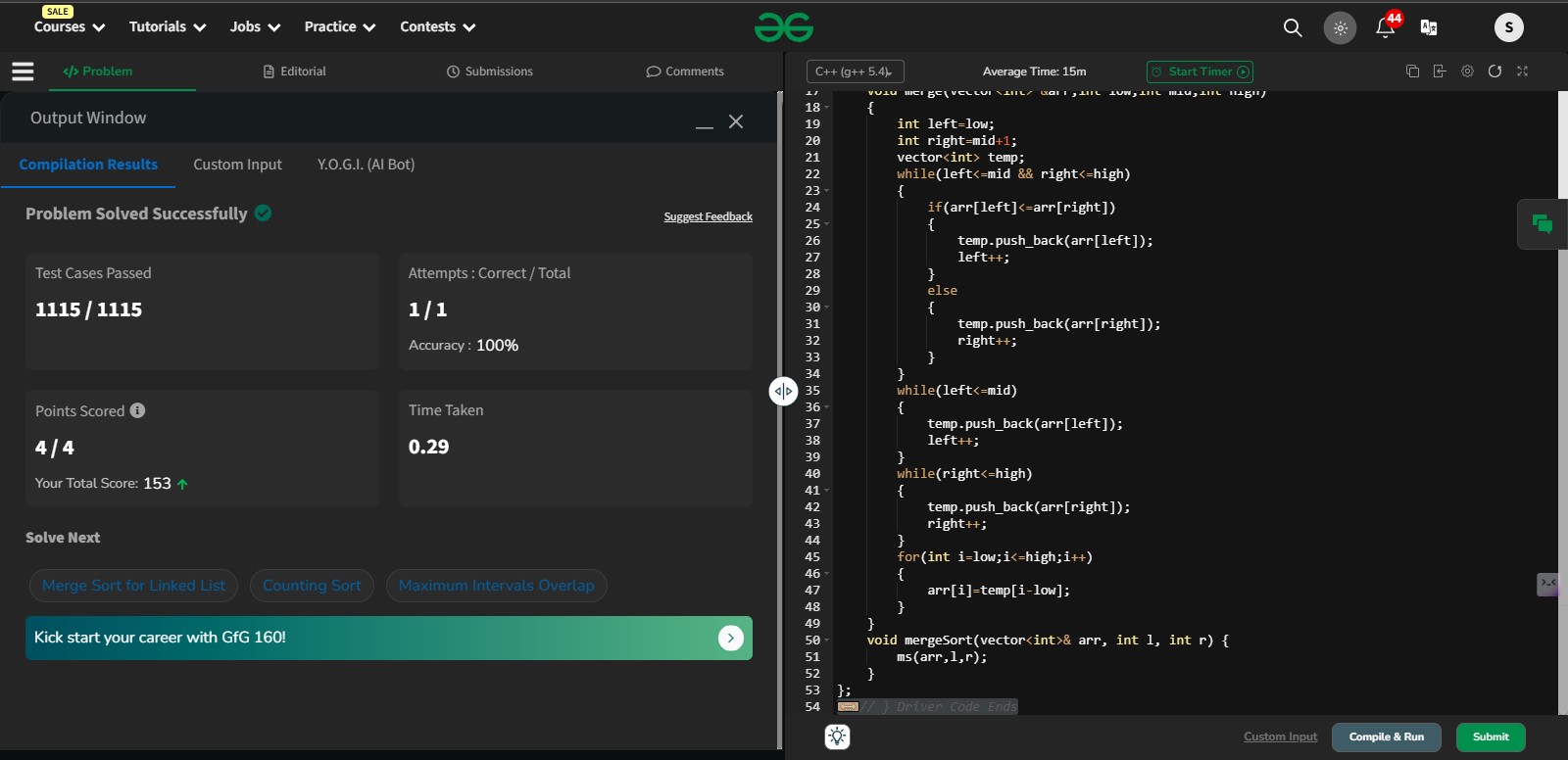
cout << "~" << endl;

}

return 0;

}

// } Driver Code Ends  
  
Output:



Time Complexity: O(N log N)

Space Complexity: O(N)

9.Ternary Search

Program:

#include <iostream>

using namespace std;

int ternarySearch(int l, int r, int key, int ar[])

{

while (r >= l) {

int mid1 = l + (r - l) / 3;

int mid2 = r - (r - l) / 3;

if (ar[mid1] == key) {

return mid1;

}

if (ar[mid2] == key) {

return mid2;

}

if (key < ar[mid1]) {

r = mid1 - 1;

}

else if (key > ar[mid2]) {

l = mid2 + 1;

}

else {

l = mid1 + 1;

r = mid2 - 1;

}

}

return -1;

}

int main()

{

int l, r, p, key;

int ar[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

l = 0;

r = 9;

key = 5;

p = ternarySearch(l, r, key, ar);

cout << "Index of "<<key<<" is " << p << endl;

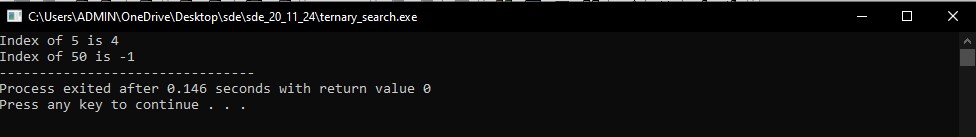
key = 50;

p = ternarySearch(l, r, key, ar);

cout << "Index of "<<key<<" is " << p;

}

Output:



Time Complexity: O(log3 n)

Space Complexity: O(1)

10.Interpolation Search

Program:

#include<bits/stdc++.h>

using namespace std;

int interpolationSearch(int arr[], int n, int x)

{

int low = 0, high = (n - 1);

while (low <= high && x >= arr[low] && x <= arr[high])

{

if (low == high)

{if (arr[low] == x) return low;

return -1;

}

int pos = low + (((double)(high - low) /

(arr[high] - arr[low])) \* (x - arr[low]));

if (arr[pos] == x)

return pos;

if (arr[pos] < x)

low = pos + 1;

else

high = pos - 1;

}

return -1;

}

int main()

{

int arr[] = {10, 12, 13, 16, 18, 19, 20, 21, 22, 23, 24, 33, 35, 42, 47};

int n = sizeof(arr)/sizeof(arr[0]);

int x = 18;

int index = interpolationSearch(arr, n, x);

if (index != -1)

cout << "Element found at index " << index;

else

cout << "Element not found.";

return 0;

}

Output:

