1.Bubble sort

Given an array, **arr[]**. Sort the array using bubble 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, 2, 3, 4, 5]

**Output**: [1, 2, 3, 4, 5]  
**Explanation**: An array that is already sorted should remain unchanged after applying bubble sort.

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

Program:

//{ Driver Code Starts

#include <bits/stdc++.h>

#include <stdio.h>

using namespace std;

// } Driver Code Ends

// User function Template for C++

class Solution {

public:

// Function to sort the array using bubble sort algorithm.

void bubbleSort(vector<int>& arr) {

int n=arr.size();

for(int i=0;i<n-1;i++)

{

for(int j=0;j<n-i-1;j++)

{

if(arr[j]>arr[j+1])

{

swap(arr[j],arr[j+1]);

}

}

}

}

};

//{ Driver Code Starts.

// Driver program to test above functions

int main() {

int t;

cin >> t;

cin.ignore();

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.bubbleSort(arr);

for (int e : arr) {

cout << e << " ";

}

cout << endl;

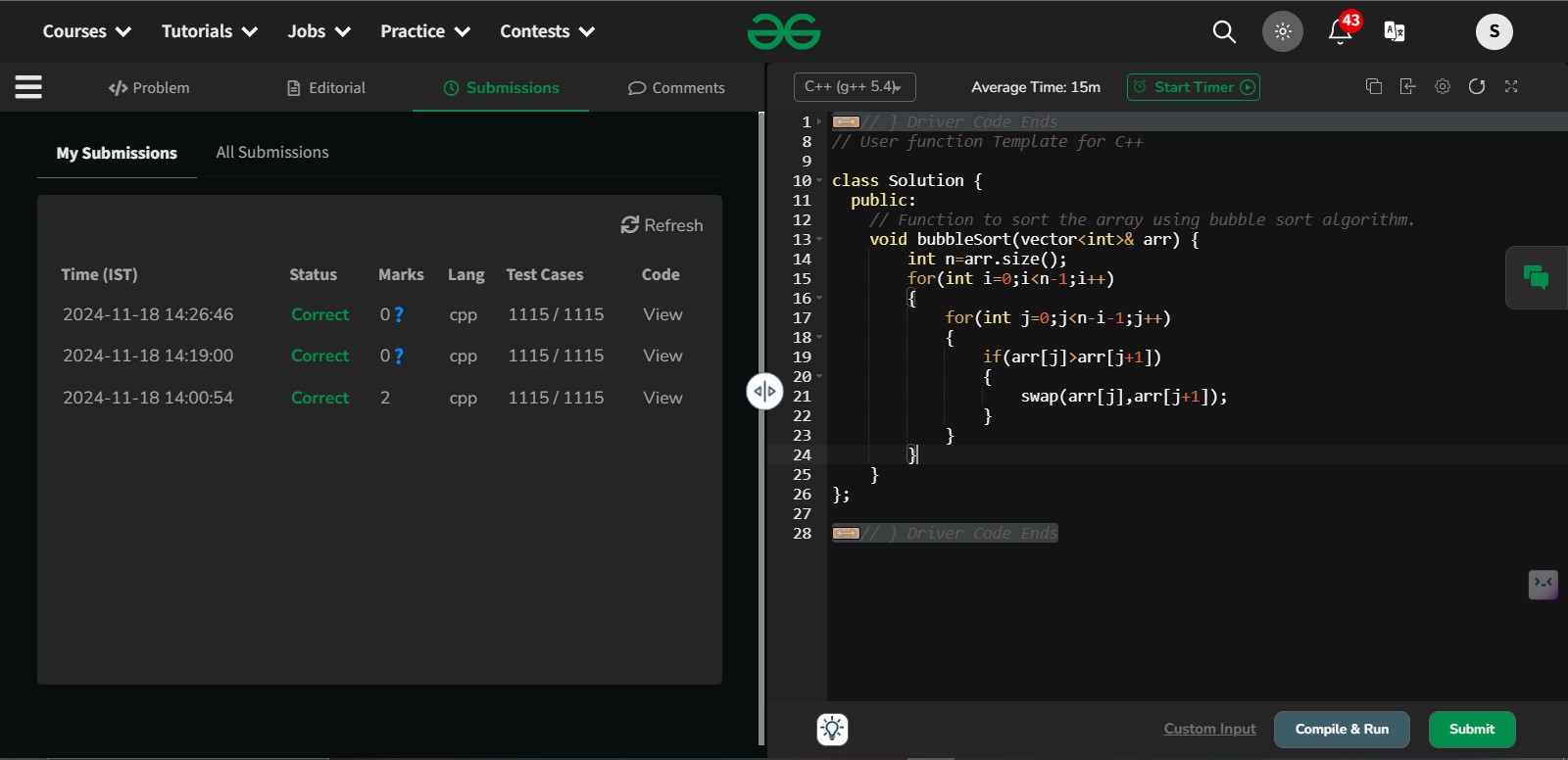
}

return 0;

}

// } Driver Code Ends

Output:



Time Complexity: O(N2)

2.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

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:

// Function to sort an array using quick sort algorithm.

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:

// Function that takes last element as pivot, places the pivot element at

// its correct position in sorted array, and places all smaller elements

// to left of pivot and all greater elements to right of pivot.

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);

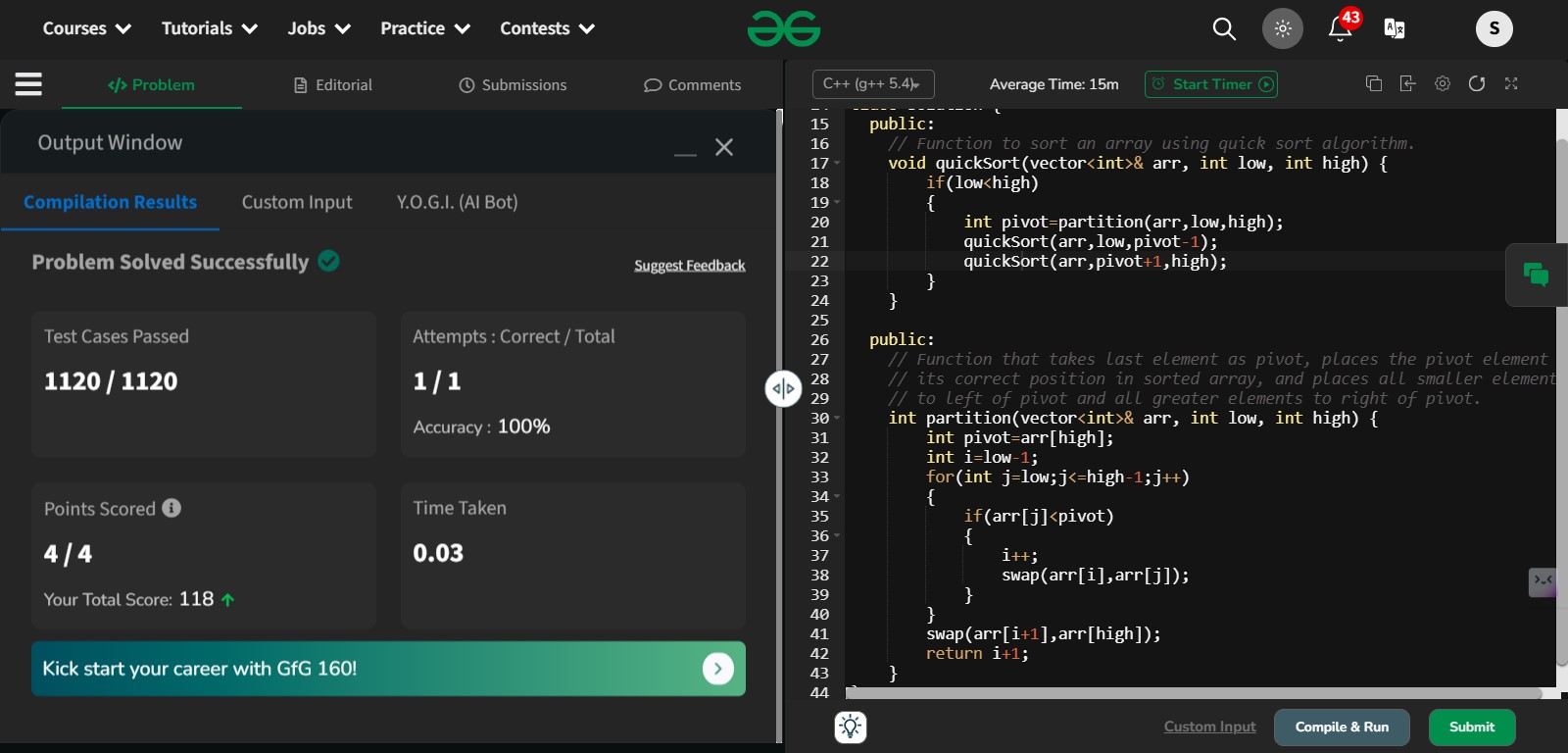
printArray(arr);

}

return 0;

}

// } Driver Code Ends  
  
Output:



Time Complexity : O(N)- average case

3. Non Repeating Character

Given a string **s** consisting of **lowercase**Latin Letters. Return the first non-repeating character in **s**. If there is no non-repeating character, return **'$'.**  
Note:When you return '$' driver code will output -1.

**Examples:**

**Input:** s = "geeksforgeeks"

**Output:** 'f'

**Explanation:** In the given string, 'f' is the first character in the string which does not repeat.

**Input:** s = "racecar"  
**Output:** 'e'  
**Explanation:** In the given string, 'e' is the only character in the string which does not repeat.

**Input:** s = "aabbccc"  
**Output:** '$'  
**Explanation:** All the characters in the given string are repeating.

**Constraints:**  
1 <= s.size() <= 105

Program:

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

class Solution {

public:

// Function to find the first non-repeating character in a string.

char nonRepeatingChar(string &s) {

unordered\_set<char> set1;

unordered\_set<char> set2;

for(char c : s) {

if(set1.find(c) != set1.end()) {

if(set2.find(c) == set2.end()) {

set2.insert(c);

}

set1.erase(c);

continue;

}

if(set2.find(c) == set2.end())

set1.insert(c);

}

for(char c : s)

if (set1.find(c) != set1.end()) return c;

return '$';

}

};

//{ Driver Code Starts.

int main() {

int T;

cin >> T;

while (T--) {

string S;

cin >> S;

Solution obj;

char ans = obj.nonRepeatingChar(S);

if (ans != '$')

cout << ans;

else

cout << "-1";

cout << endl;

cout << "~"

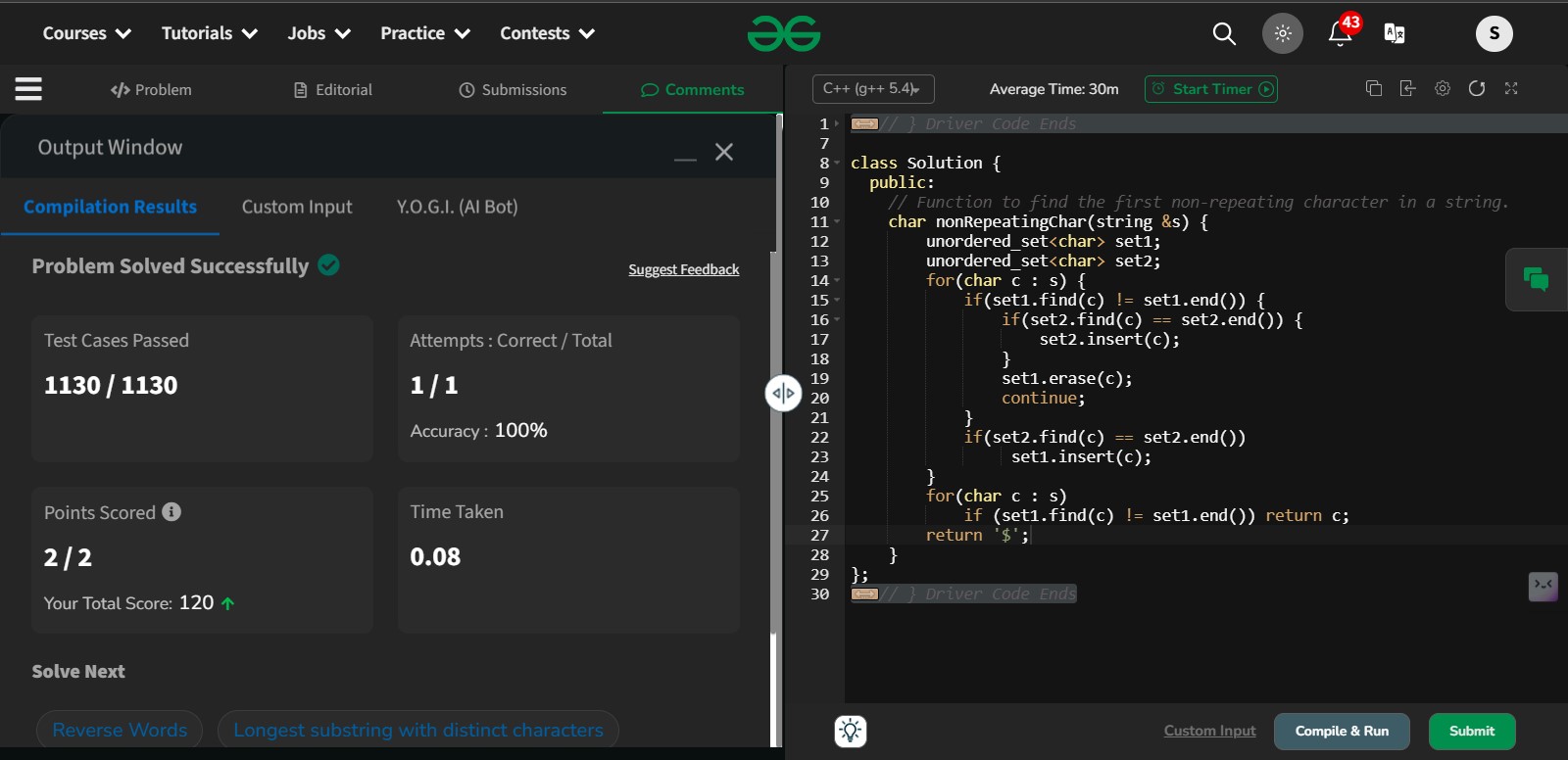
<< "\n";

}

return 0;

}

// } Driver Code Ends  
Output:



Time Complexity: O(N)

4.Edit Distance

Given two strings **s1** and **s2.**Return the minimum number of operations required to convert **s1**to **s2**.  
The possible operations are permitted:

1. Insert a character at any position of the string.
2. Remove any character from the string.
3. Replace any character from the string with any other character.

**Examples:**

**Input:** s1 = "geek", s2 = "gesek"

**Output:** 1

**Explanation:** One operation is required, inserting 's' between two 'e'.

**Input :** s1 = "gfg", s2 = "gfg"

**Output:** 0

**Explanation:** Both strings are same.

**Input :** s1 = "abc", s2 = "def"

**Output:** 3

**Explanation:** All characters need to be replaced to convert str1 to str2, requiring 3 replacement operations.

**Constraints:**  
1 ≤ s1.length(), s2.length() ≤ 500  
both the strings are in lowercase.

Program:

//{ Driver Code Starts

// Initial Template for C++

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

class Solution {

public:

// Function to compute the edit distance between two strings

int minDisRec(string& s1, string& s2, int m, int n,vector<vector<int>>& memo) {

if (m == 0) return n;

if (n == 0) return m;

if (memo[m][n] != -1) return memo[m][n];

if (s1[m - 1] == s2[n - 1]) {

memo[m][n] = minDisRec(s1, s2, m - 1, n - 1, memo);

} else {

memo[m][n] = 1 + min({

minDisRec(s1, s2, m, n - 1, memo), // Insert

minDisRec(s1, s2, m - 1, n, memo), // Remove

minDisRec(s1, s2, m - 1, n - 1, memo) // Replace

});

}

return memo[m][n];

}

int minDis(string& s1, string& s2)

{

int m = s1.length(), n = s2.length();

vector<vector<int>> memo(m + 1, vector<int>(n + 1, -1));

return minDisRec(s1, s2, m, n, memo);

}

int editDistance(string s1, string s2) {

return minDis(s1,s2);

}

};

//{ Driver Code Starts.

int main() {

int T;

cin >> T;

cin.ignore();

while (T--) {

string s1;

getline(cin, s1);

string s2;

getline(cin, s2);

Solution ob;

int ans = ob.editDistance(s1, s2);

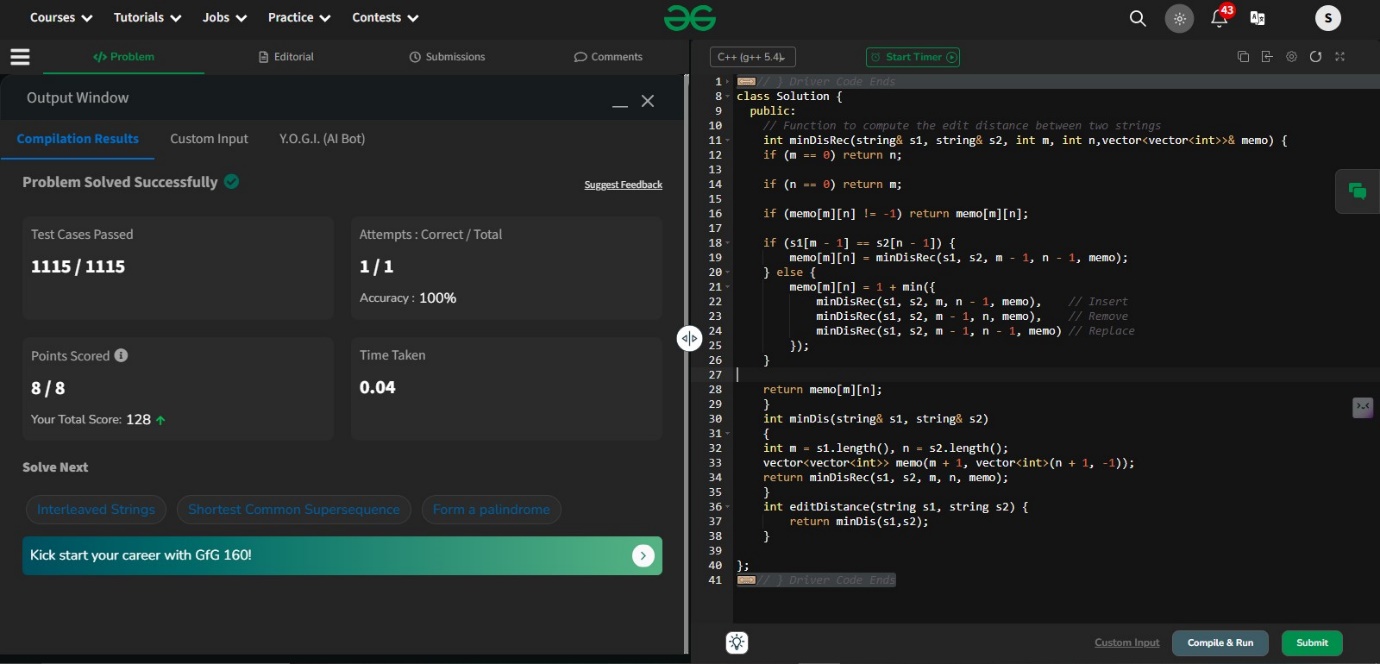
cout << ans << "\n";

}

return 0;

}

// } Driver Code Ends  
  
Output:



Time Complexity: O(M\*N)

5.k largest elements

Given an array **arr[]** of positive integers and an integer **k**, Your task is to return **k largest elements**in decreasing order.

**Examples**

**Input:** arr[] = [12, 5, 787, 1, 23], k = 2

**Output:** [787, 23]

**Explanation:** 1st largest element in the array is 787 and second largest is 23.

**Input:** arr[] = [1, 23, 12, 9, 30, 2, 50], k = 3

**Output:** [50, 30, 23]

**Explanation:** Three Largest elements in the array are 50, 30 and 23.

**Input:** arr[] = [12, 23], k = 1

**Output:** [23]

**Explanation:** 1st Largest element in the array is 23.

**Constraints:**  
1 ≤ k ≤ arr.size() ≤ 106  
1 ≤ arr[i] ≤ 106

Program:

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// User function template for C++

class Solution {

public:

vector<int> kLargest(vector<int>& arr, int k) {

int n=arr.size();

if(k>=n)

{

return arr;

}

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

reverse(arr.begin(),arr.end());

vector<int> ans;

for(int i=0;i<k;i++)

{

ans.push\_back(arr[i]);

}

return ans;

}

};

//{ 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);

}

string ks;

getline(cin, ks);

int k = stoi(ks);

Solution ob;

vector<int> ans = ob.kLargest(arr, k);

for (auto it : ans) {

cout << it << " ";

}

cout << endl;

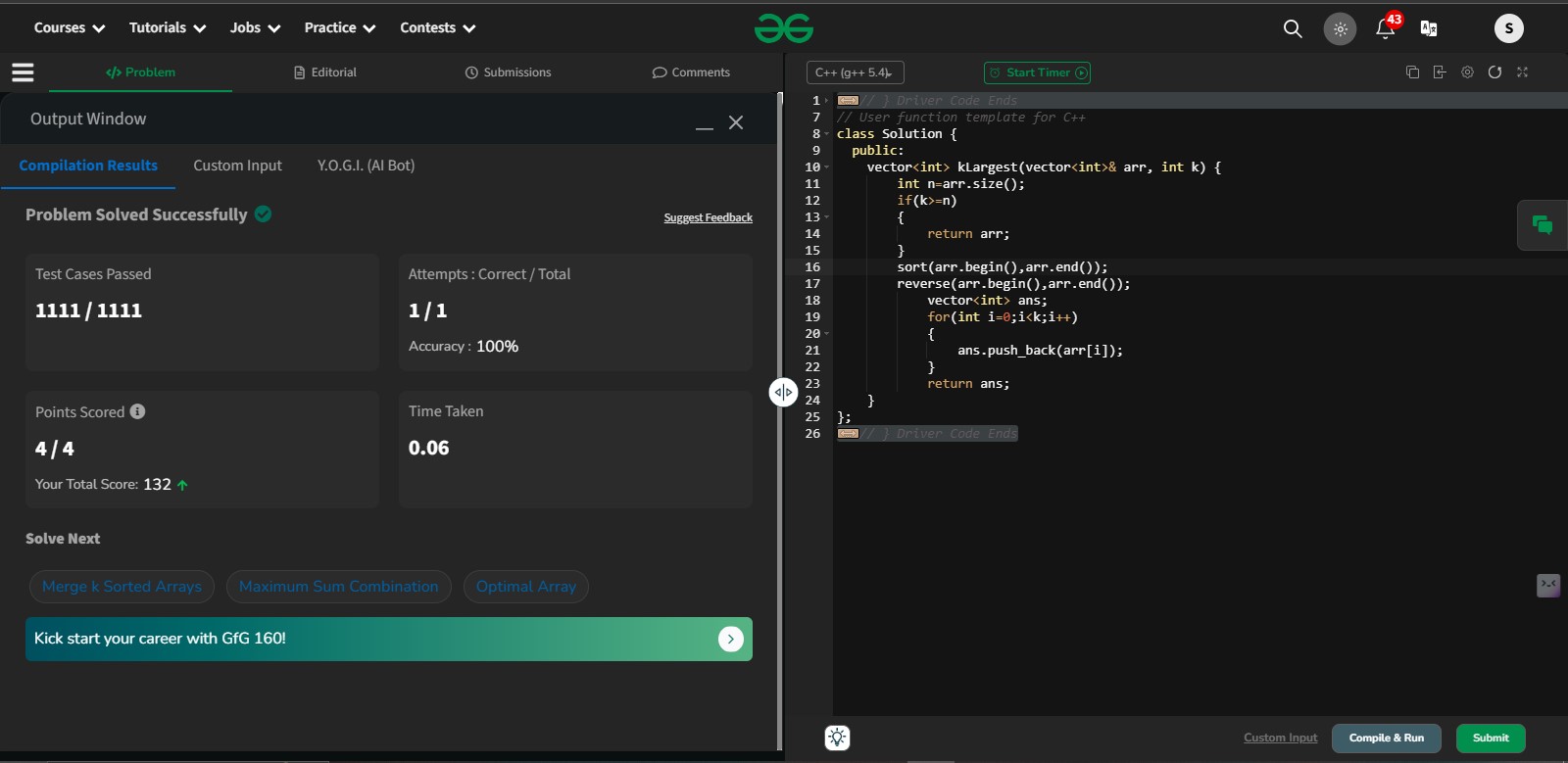
cout << "~" << endl;

}

return 0;

}

// } Driver Code Ends  
Output:



Time Complexity: O(N log N)

Program – using priority queue

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// User function template for C++

class Solution {

public:

vector<int> kLargest(vector<int>& arr, int k) {

priority\_queue<int>p;

int n=arr.size();

vector<int>ans;

for(int i=0;i<n;i++){

p.push(arr[i]);

}

while(k--){

ans.push\_back(p.top());

p.pop();

}

return ans;

}

};

//{ 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);

}

string ks;

getline(cin, ks);

int k = stoi(ks);

Solution ob;

vector<int> ans = ob.kLargest(arr, k);

for (auto it : ans) {

cout << it << " ";

}

cout << endl;

cout << "~" << endl;

}

return 0;

}

// } Driver Code Ends

Time Complexity: O(n+log k)

6.Form the largest number

Given an array of integers **arr[]**representing non-negative integers, arrange them so that after concatenating all of them in order, it results in the **largest**possible**number**. Since the result may be very large, return it as a string.

**Examples:**

**Input:** arr[] = [3, 30, 34, 5, 9]

**Output:** "9534330"

**Explanation:** Given numbers are {3, 30, 34, 5, 9}, the arrangement "9534330" gives the largest value.

**Input:** arr[] = [54, 546, 548, 60]

**Output:** "6054854654"

**Explanation:** Given numbers are {54, 546, 548, 60}, the arrangement "6054854654" gives the largest value.

**Input:** arr[] = [3, 4, 6, 5, 9]

**Output:** "96543"

**Explanation:** Given numbers are {3, 4, 6, 5, 9}, the arrangement "96543" gives the largest value.

**Constraints:**  
1 ≤ arr.size() ≤ 105  
0 ≤ arr[i] ≤ 105  
The sum of all the elements of the array is greater than 0.

Try more examples

Program:

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// User function template for C++

class Solution {

public:

static bool compare(string &a, string &b){

return a + b > b + a;

}

string printlargest(int n, vector<string> &arr) {

sort(arr.begin(), arr.end(), compare);

string ans;

for (string &num : arr) {

ans += num;

}

return ans;

}

string printLargest(vector<int> &arr) {

int n=arr.size();

vector<string> ans;

for(int i=0;i<n;i++)

{

ans.push\_back(to\_string(arr[i]));

}

return printlargest(n,ans);

}

};

//{ Driver Code Starts.

int main() {

int t;

cin >> t;

cin.ignore();

while (t--) {

vector<int> arr;

string input;

getline(cin, input);

stringstream ss(input);

int number;

while (ss >> number) {

arr.push\_back(number);

}

Solution ob;

string ans = ob.printLargest(arr);

cout << ans << endl;

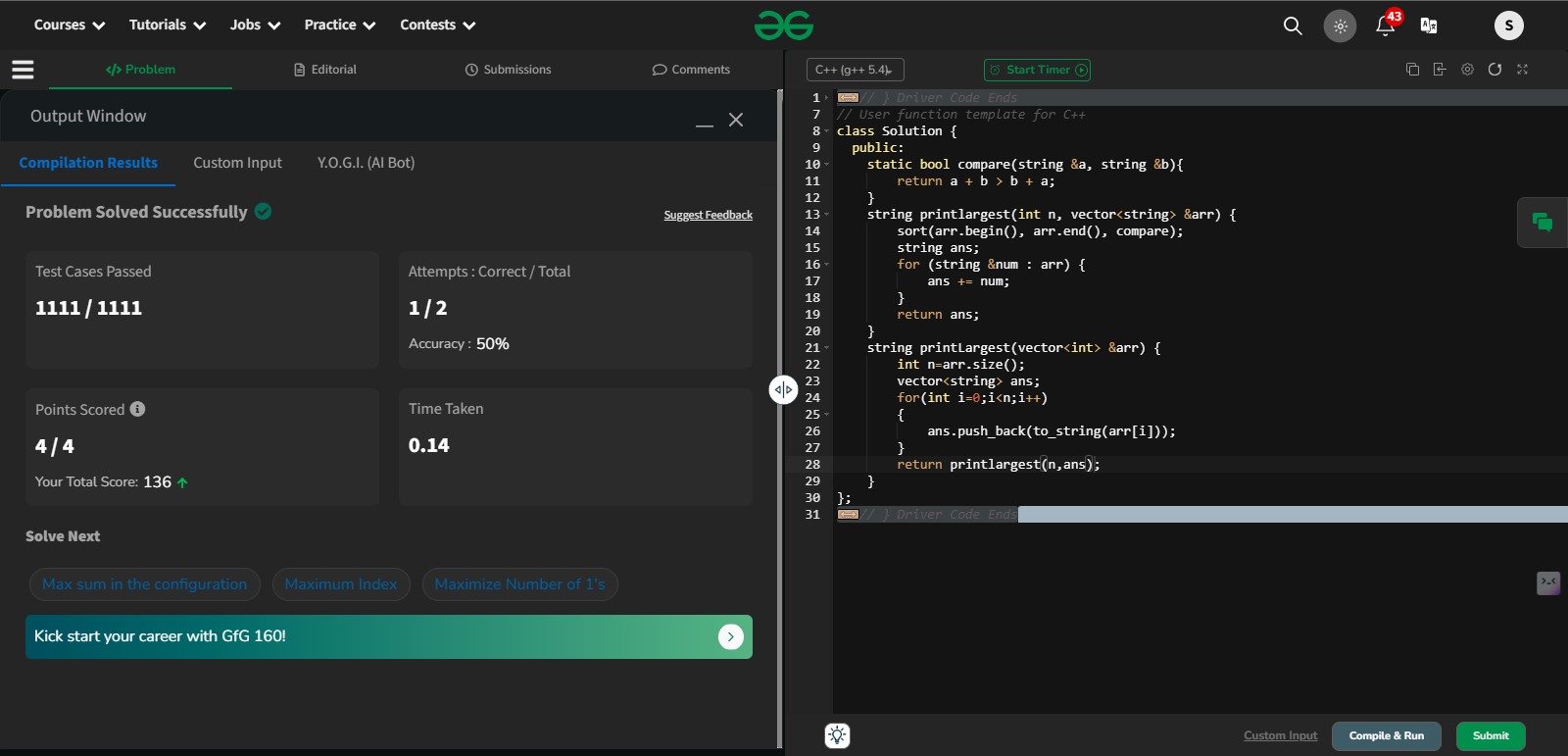
cout << "~" << endl;

}

return 0;

}

// } Driver Code Ends  
  
Output:



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