1. **K-th smallest element**

Given an array arr[] and an integer k where k is smaller than the size of the array, the task is to find the kth smallest element in the given array.

Follow up: Don't solve it using the inbuilt sort function.

Examples :

Input: arr[] = [7, 10, 4, 3, 20, 15], k = 3

Output: 7

Explanation: 3rd smallest element in the given array is 7.

Input: arr[] = [2, 3, 1, 20, 15], k = 4

Output: 15

Explanation: 4th smallest element in the given array is 15.

Expected Time Complexity: O(n+(max\_element) )

Expected Auxiliary Space: O(max\_element)

Constraints:

1 <= arr.size <= 106

1<= arr[i] <= 106

1 <= k <= n  
  
**Program:**  
class Solution {

public static int kthSmallest(int[] arr, int k) {

PriorityQueue<Integer> a = new PriorityQueue<>();

for (int num : arr) {

a.add(num);

}

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

a.poll();

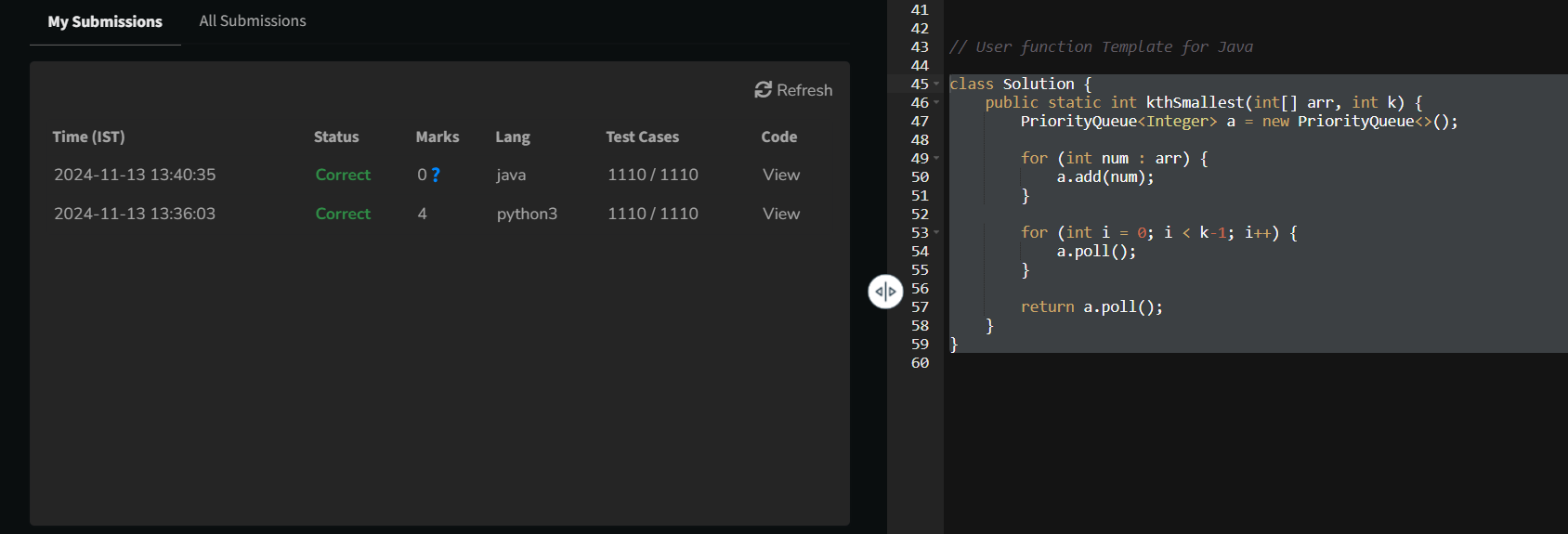
}

return a.poll();

}

}

**Output:**



Time Complexity: O(N) – average case  
  
**2.Minimize the heights-II**

Given an array arr[] denoting heights of N towers and a positive integer K.

For each tower, you must perform exactly one of the following operations exactly once.

* Increase the height of the tower by K
* Decrease the height of the tower by K

Find out the minimum possible difference between the height of the shortest and tallest towers after you have modified each tower.

You can find a slight modification of the problem [here](https://practice.geeksforgeeks.org/problems/minimize-the-heights-i/1/).  
Note: It is compulsory to increase or decrease the height by K for each tower. After the operation, the resultant array should not contain any negative integers.

Examples :

Input: k = 2, arr[] = {1, 5, 8, 10}

Output: 5

Explanation: The array can be modified as {1+k, 5-k, 8-k, 10-k} = {3, 3, 6, 8}.The difference between the largest and the smallest is 8-3 = 5.

Input: k = 3, arr[] = {3, 9, 12, 16, 20}

Output: 11

Explanation: The array can be modified as {3+k, 9+k, 12-k, 16-k, 20-k} -> {6, 12, 9, 13, 17}.The difference between the largest and the smallest is 17-6 = 11.

Expected Time Complexity: O(n\*logn)  
Expected Auxiliary Space: O(n)

Constraints  
1 ≤ k ≤ 107  
1 ≤ n ≤ 105  
1 ≤ arr[i] ≤ 107

**Program:**

class Solution {

int getMinDiff(int[] arr, int k) {

int n = arr.length;

Arrays.sort(arr);

int res = arr[n - 1] - arr[0];

for (int i = 1; i < arr.length; i++) {

if (arr[i] - k < 0)

continue;

int minH = Math.min(arr[0] + k, arr[i] - k);

int maxH = Math.max(arr[i - 1] + k, arr[n - 1] - k);

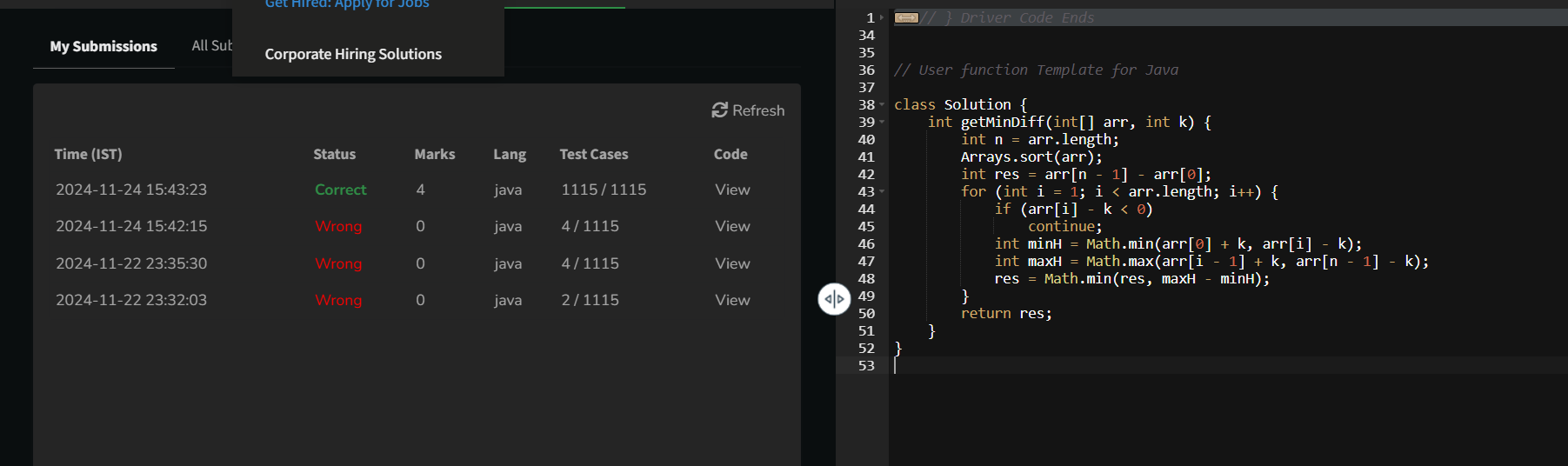
res = Math.min(res, maxH - minH);

}

return res;

}

}  
**Output:**



3.Parenthesis Checker

You are given a string **s** representing an expression containing various types of brackets: {}, (), and []. Your task is to determine whether the brackets in the expression are balanced. A balanced expression is one where every opening bracket has a corresponding closing bracket in the correct order.

**Examples :**

**Input**: s = "{([])}"

**Output**: true

**Explanation**:   
- In this expression, every opening bracket has a corresponding closing bracket.  
- The first bracket { is closed by }, the second opening bracket ( is closed by ), and the third opening bracket [ is closed by ].  
- As all brackets are properly paired and closed in the correct order, the expression is considered balanced.

**Input**: s = "()"

**Output**: true

**Explanation**:   
- This expression contains only one type of bracket, the parentheses ( and ).  
- The opening bracket ( is matched with its corresponding closing bracket ).  
- Since they form a complete pair, the expression is balanced.

**Input**: s = "([]"

**Output**: false

**Explanation**:   
- This expression contains only one type of bracket, the parentheses ( and ).  
- The opening bracket ( is matched with its corresponding closing bracket ).  
- Since they form a complete pair, the expression is balanced.

**Constraints:**  
1 ≤ s.size() ≤ 106s[i] ∈ {'{', '}', '(', ')', '[', ']'}

Program:

class Solution {

static boolean isParenthesisBalanced (String s) {

Stack<Character> stack = new Stack<>();

Map<Character, Character> map = new HashMap<>();

map.put(')', '(');

map.put(']', '[');

map.put('}', '{');

for (char c : s.toCharArray()) {

if (map.containsValue(c)) {

stack.push(c);

} else if (map.containsKey(c)) {

if (stack.isEmpty() || stack.pop() != map.get(c)) {

return false;

}

}

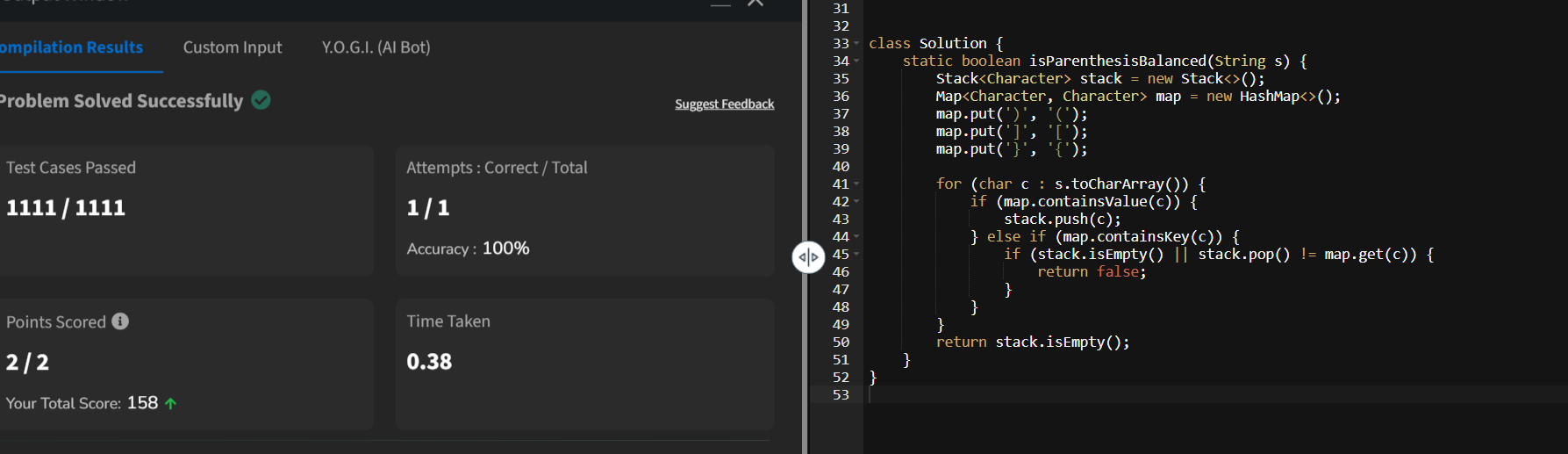
}

return stack.isEmpty();

}

}

Output:



4.Equilibrium point

Given an array**arr**of non-negative numbers. The task is to find the first **equilibrium point** in an array. The equilibrium point in an array is an index (or position) such that the sum of all elements beforethat index is the same as the sumof elements afterit.

**Note:** Return equilibrium point in 1-based indexing. Return -1 if no such point exists.

**Examples:**

**Input:** arr[] = [1, 3, 5, 2, 2]  
**Output:** 3

**Explanation:** The equilibrium point is at position 3 as the sum of elements before it (1+3) = sum of elements after it (2+2).

**Input:** arr[] = [1]  
**Output:** 1

**Explanation:** Since there's only one element hence it's only the equilibrium point.

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

**Explanation:** There is no equilibrium point in the given array.

**Expected Time Complexity:**O(n)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 <= arr.size <= 106  
0 <= arr[i] <= 109

Program:

class Solution {

// Function to find equilibrium point in the array.

public static int equilibriumPoint(int arr[]) {

int n = arr.length;

if (n == 1) return 1;

int totalSum = 0;

for (int num : arr) {

totalSum += num;

}

int leftSum = 0;

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

totalSum -= arr[i];

if (leftSum == totalSum) {

return i + 1;

}

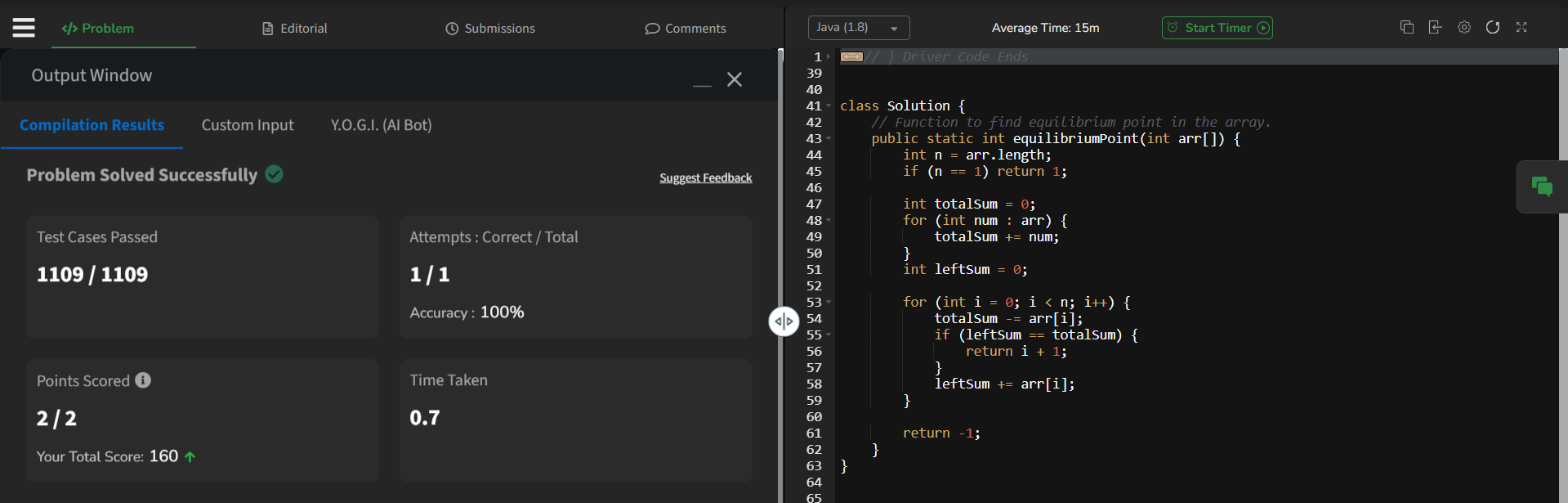
leftSum += arr[i];

}

return -1;

}

}  
Output:



Time Complexity: O(N)

5.Binary Search  
Given a sorted array **arr** and an integer **k**, find the position(0-based indexing) at which k is present in the array using binary search.

Note: If multiple occurrences are there, please return the smallest index.

**Examples:**

**Input:** arr[] = [1, 2, 3, 4, 5], k = 4

**Output:** 3

**Explanation:** 4 appears at index 3.

**Input:** arr[] = [11, 22, 33, 44, 55], k = 445

**Output:** -1

**Explanation:** 445 is not present.

*Note: Try to solve this problem in constant space i.e O(1)*

**Constraints:**1 <= arr.size() <= 1051 <= arr[i] <= 1061 <= k <= 106

Program:

class Solution {

public int binarysearch(int[] a, int k) {

int l = 0;

int r = a.length - 1;

if (k > a[r]) return -1;

while (l <= r) {

int m = (l + r) / 2;

if (a[m] == k) return m;

else if (a[m] < k) l = m + 1;

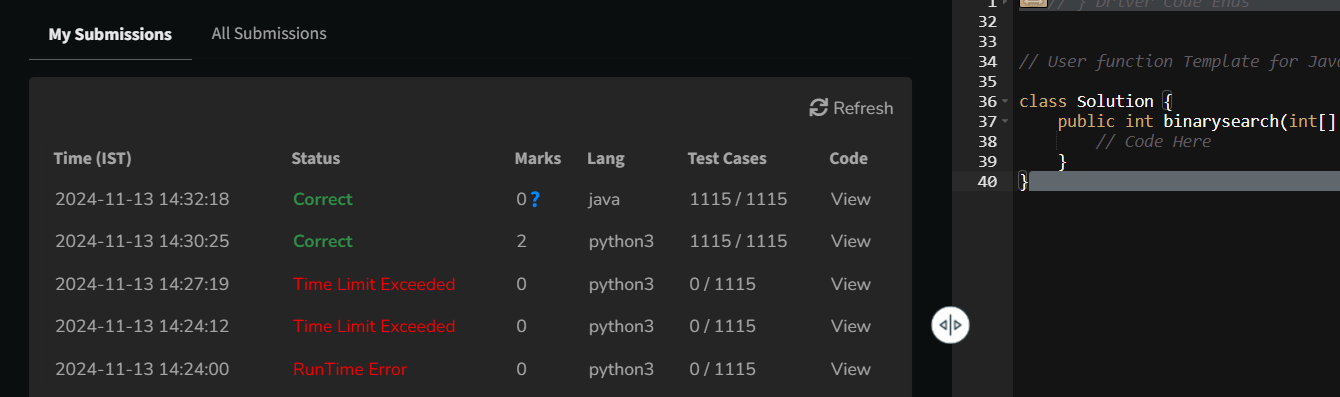
else r = m - 1;

}

return -1;

}

}  
Output:



Time Complexity: O(N)

6.Next Greater Element

Given an array **arr[ ]** of integers, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.  
If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

**Examples**

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

**Output**: [3, 4, 4, -1]

**Explanation**: The next larger element to 1 is 3, 3 is 4, 2 is 4 and for 4, since it doesn't exist, it is -1.

**Input**: arr[] = [6, 8, 0, 1, 3]

**Output**: [8, -1, 1, 3, -1]

**Explanation**: 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 right and hence -1.

**Input**: arr[] = [10, 20, 30, 50]

**Output**: [20, 30, 50, -1]

**Explanation**: For a sorted array, the next element is next greater element also exxept for the last element.

**Input**: arr[] = [50, 40, 30, 10]

**Output**: [-1, -1, -1, -1]

**Explanation**: There is no greater element for any of the elements in the array, so all are -1.

**Constraints:**  
1 ≤ arr.size() ≤ 106  
0 ≤ arr[i] ≤ 109

Program:

class Solution {

public ArrayList<Integer> nextLargerElement(int[] arr) {

Stack<Integer> s = new Stack<>();

int n = arr.length;

ArrayList<Integer> ans = new ArrayList<>(Collections.nCopies(n, -1));

s.push(-1);

for (int i = n - 1; i >= 0; i--) {

int curr = arr[i];

while (s.peek() != -1 && s.peek() <= curr) {

s.pop();

}

ans.set(i, s.peek());

s.push(curr);

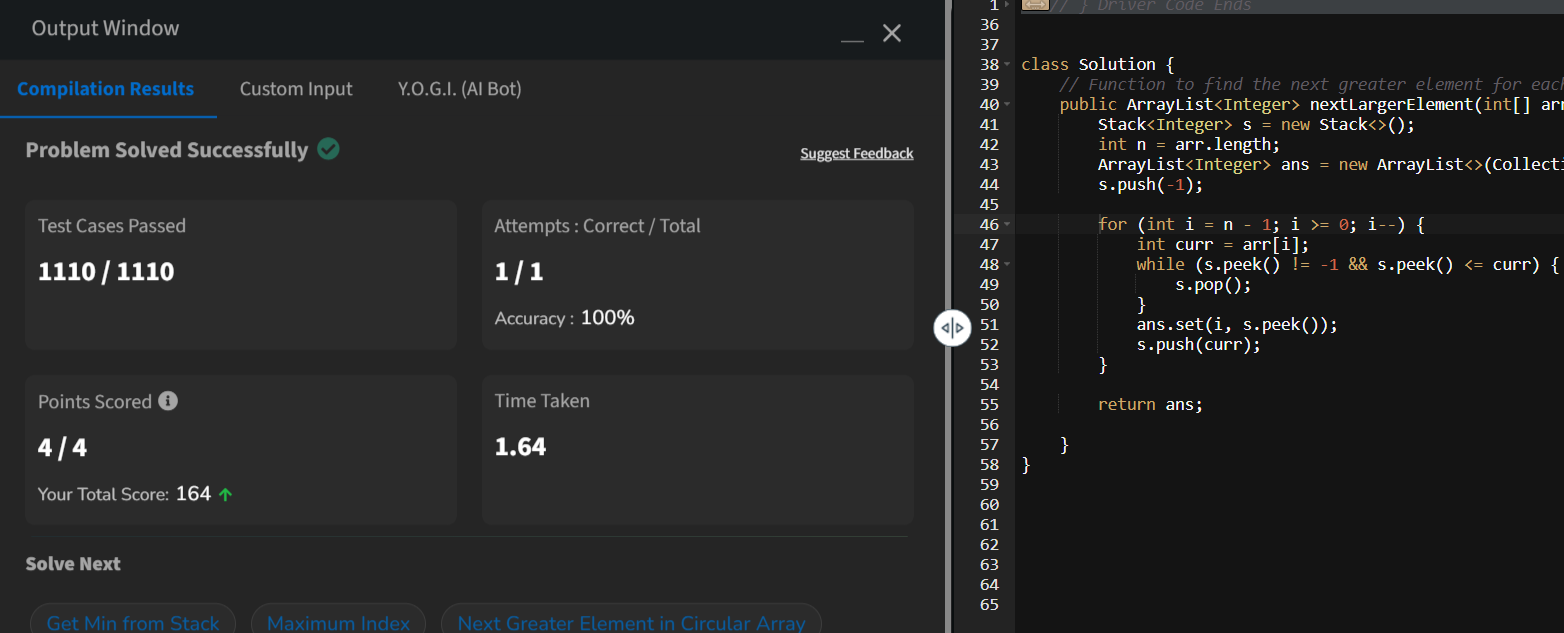
}

return ans;

}

}

Output:



7.Union of two arrays with duplicate elements

Given two arrays **a[]** and **b[]**,the task is to find the number of elements in the union between these two arrays.

The Union of the two arrays can be defined as the set containing distinct elements from both arrays. If there are repetitions, then only one element occurrence should be there in the union.

*Note:*Elements are not necessarily distinct.

**Examples**

**Input:** a[] = [1, 2, 3, 4, 5], b[] = [1, 2, 3]

**Output:** 5

**Explanation:** 1, 2, 3, 4 and 5 are the elements which comes in the union setof both arrays. So count is 5.

**Input:** a[] =[85, 25, 1, 32, 54, 6], b[] = [85, 2]   
**Output:** 7

**Explanation:** 85, 25, 1, 32, 54, 6, and 2 are the elements which comes in the union set of both arrays. So count is 7.

**Input:** a[] =[1, 2, 1, 1, 2], b[] = [2, 2, 1, 2, 1]   
**Output:** 2

**Explanation:** We need to consider only distinct. So count is 2.

**Constraints:**  
1 ≤ a.size(), b.size() ≤ 1060 ≤ a[i], b[i] < 105

Program:

class Solution {

public static int findUnion(int a[], int b[]) {

Set<Integer> ans = new HashSet<>();

for (int num : a) {

ans.add(num);

}

for (int num : b) {

ans.add(num);

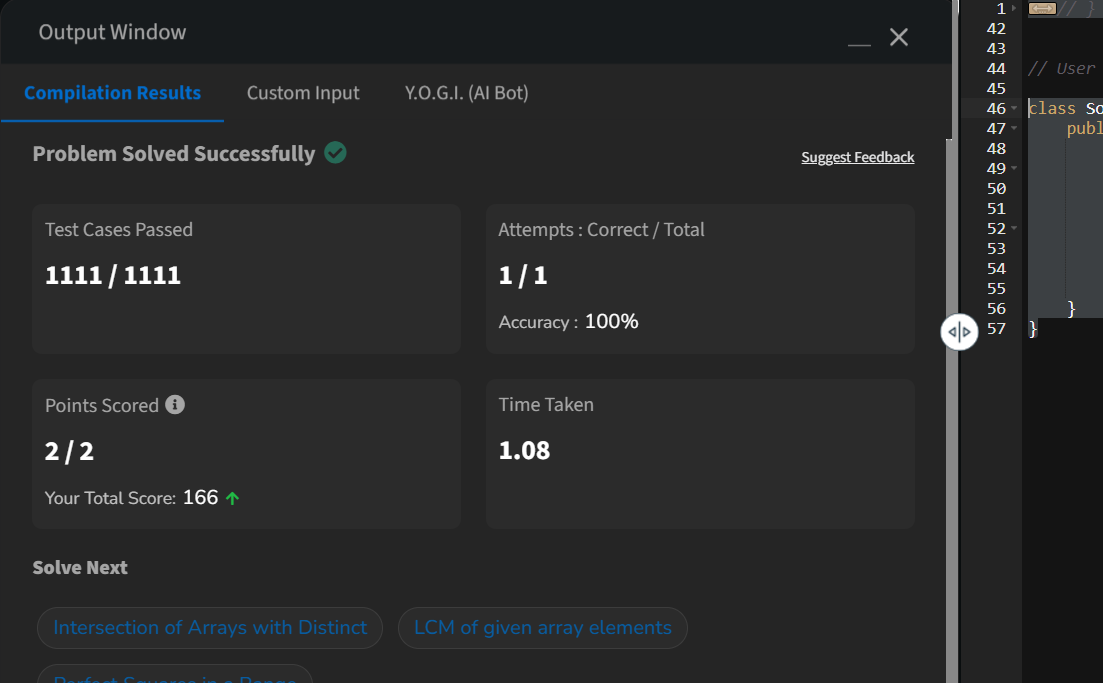
}

return ans.size();

}

}

output:



Time Complexity: O(M+N)