**DSA PRACTICE - 4 – 13/11/24**

**1.KTH SMALLEST ELEMENT**

class Solution {

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

// Your code here

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

for(int i : arr){

pq.add(i);

}

while(k-1>0){

pq.remove();

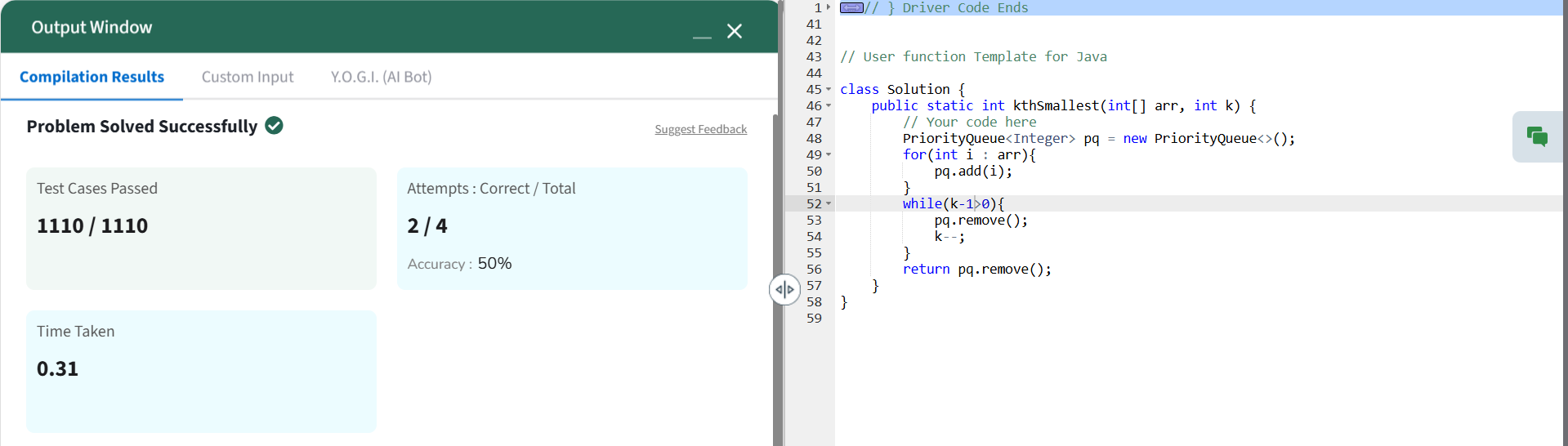
k--;

}

return pq.remove();

}

}



Time Complexity : O(k\*logn)

Space Complexity : O(n)

**2.PARENTHESIS CHECKER**

class Solution

{

//Function to check if brackets are balanced or not.

static boolean isParenthesisBalanced(String x)

{

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

for(char ch : x.toCharArray()){

if(ch == '(' || ch== '{' || ch== '['){

s.push(ch);

}

else{

if(s.isEmpty()){

return false;

}

char top = s.pop();

if((ch == ')' && top != '(') ||

(ch == '}' && top != '{') ||

(ch == ']' && top != '[')){

return false;

}

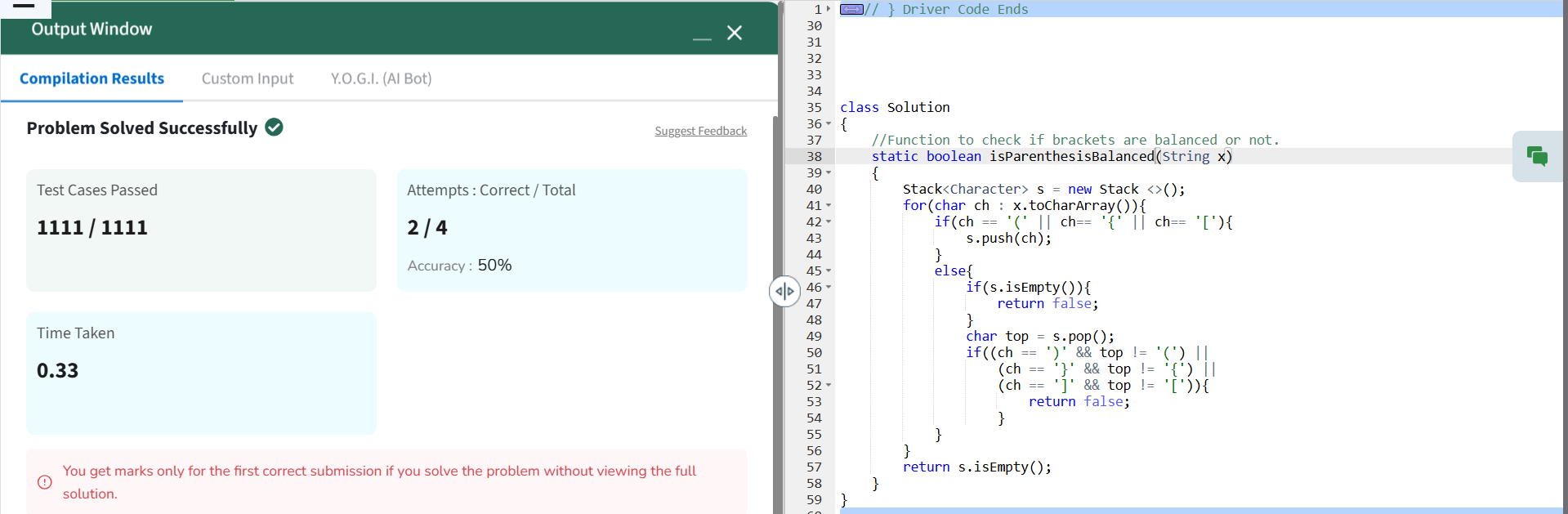
}

}

return s.isEmpty();

}

}



Time Complexity : O(n)

Space Complexity : O(n)

**3.MINIMIZE HEIGHTS II**

class Solution {

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

int n = arr.length;

if (n == 1) return 0;

Arrays.sort(arr)

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

int smallest = arr[0] + k;

int largest = arr[n - 1] - k;

int minDiff = initialDiff;

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

int minHeight = Math.min(smallest, arr[i + 1] - k);

int maxHeight = Math.max(largest, arr[i] + k);

if (minHeight < 0) continue;

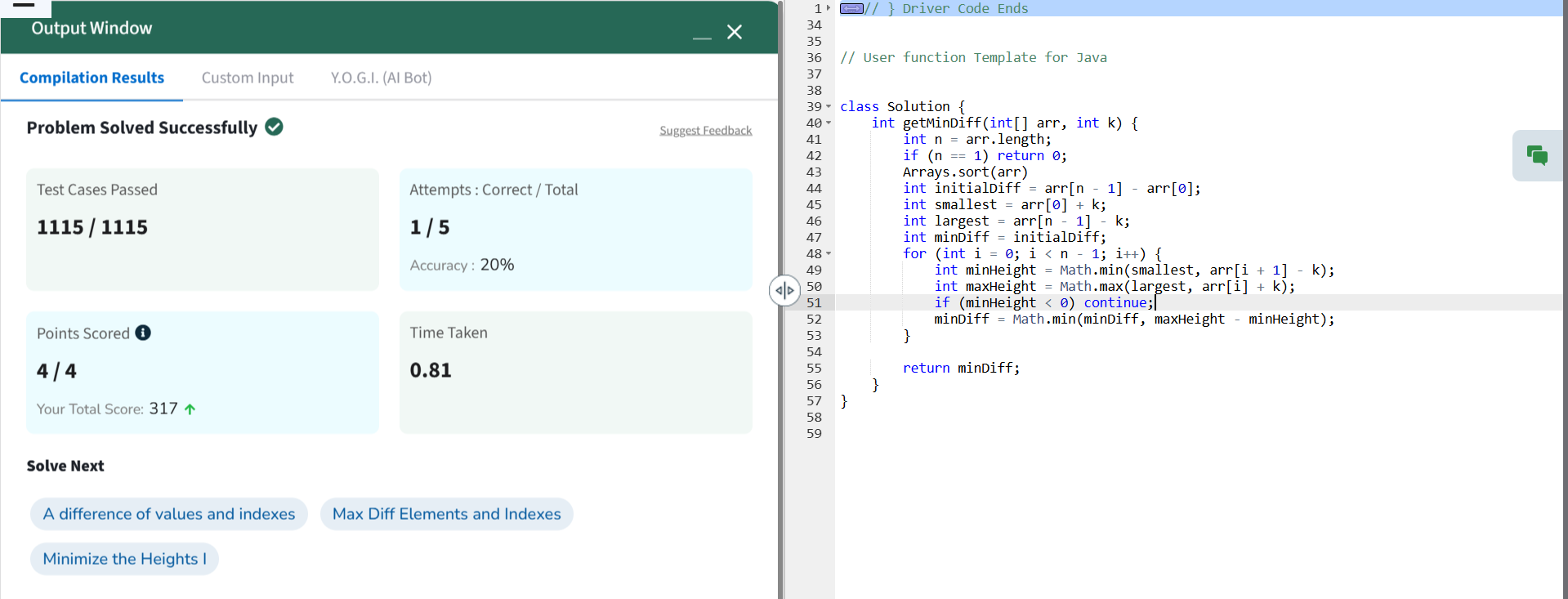
minDiff = Math.min(minDiff, maxHeight - minHeight);

}

return minDiff;

}

}



Time Complexity : O(nlogn)

Space Complexity : O(1)

**4. EQUILIBRIUM POINT**

class Solution {

// arr: input array

// Function to find equilibrium point in the array.

public static int equilibriumPoint(int arr[]) {

int n = arr.length;

int left = 0;

int right = n - 1;

long suml = 0;

long sumr = 0;

while (left < right) {

if (suml < sumr) {

suml += arr[left];

left += 1;

} else {

sumr += arr[right];

right -= 1;

}

}

if (suml == sumr) {

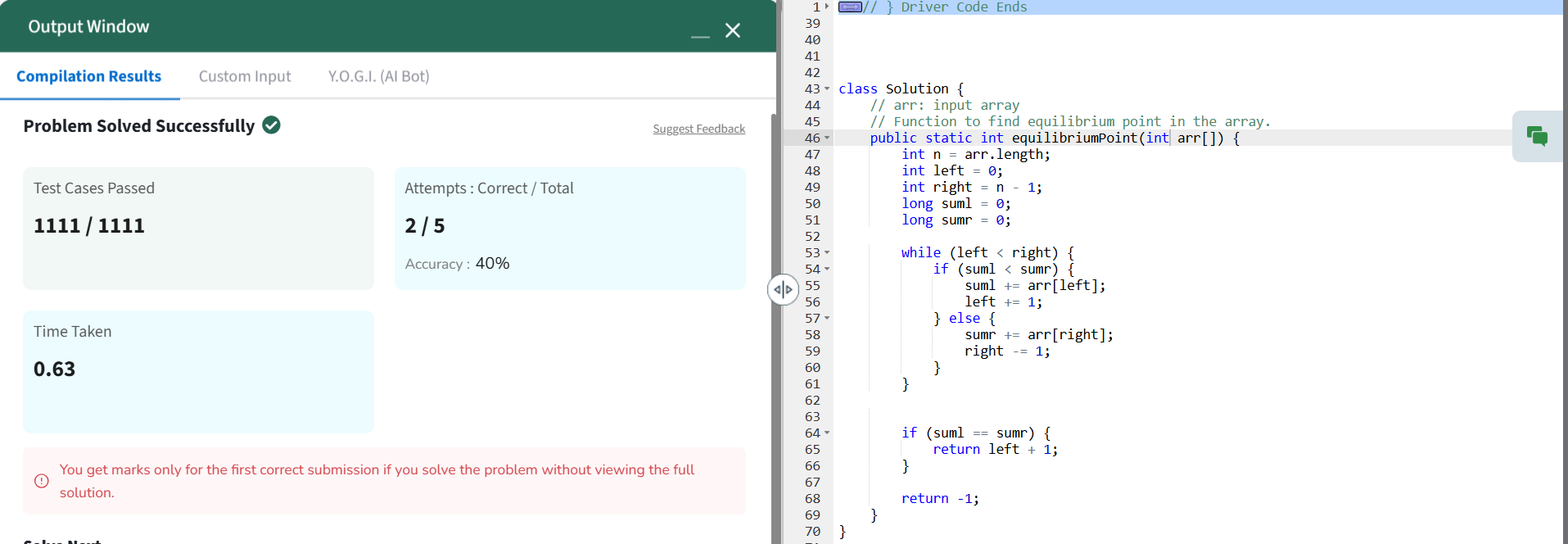
return left + 1;

}

return -1;

}

}



Time Complexity : O(log n)

Space Complexity : O(1)

**5. BINARY SEARCH**

class Solution {

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

// Code Here

int n=arr.length;

int left=0, right=n-1;

while(left<=right){

int mid=left+(right-left)/2;

if(arr[mid]==k){

return mid;

}

else if(arr[mid]<k){

left=mid+1;

}

else{

right=mid-1;

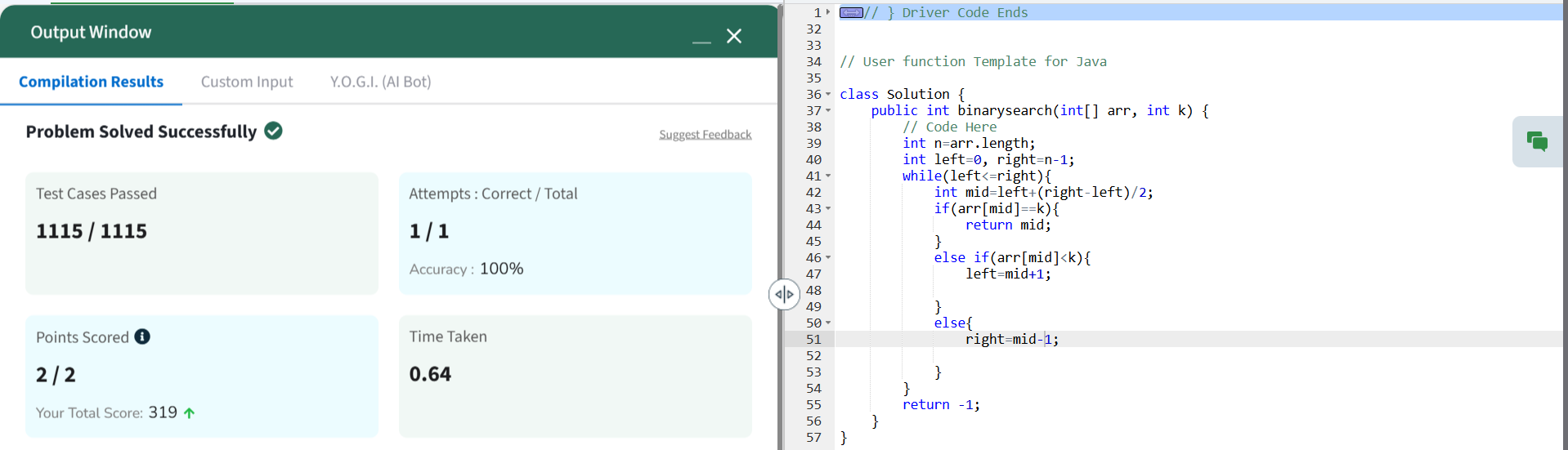
}

}

return -1;

}

}



Time Complexity : O(log n)

Space Complexity : O(1)

**6. UNION OF ARRAYS WITH DUPLICATE ELEMENTS**

class Solution {

// Function to return a list containing the union of the two arrays.

public static ArrayList<Integer> findUnion(int a[], int b[]) {

// add your code here

HashSet<Integer> uset = new HashSet<>();

for(int i : a){

uset.add(i);

}

for(int i : b){

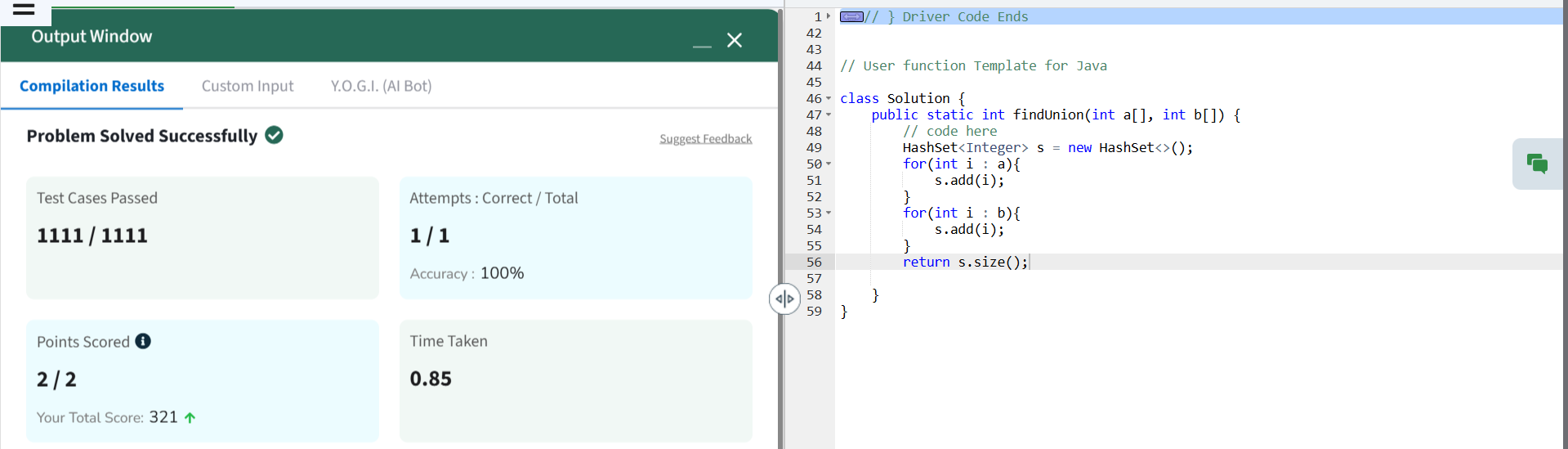
uset.add(i);

}

return uset.size();

}

}



Time Complexity : O(n)

Space Complexity : O(n)

**7. NEXT GREATER ELEMENT**

class Solution {

// Function to find the next greater element for each element of the array.

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

int n = arr.length;

ArrayList<Integer> result = new ArrayList<>(n);

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

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

result.add(-1);

}

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

while (!stack.isEmpty() && stack.peek() <= arr[i]) {

stack.pop();

}

if (!stack.isEmpty()) {

result.set(i, stack.peek());

}

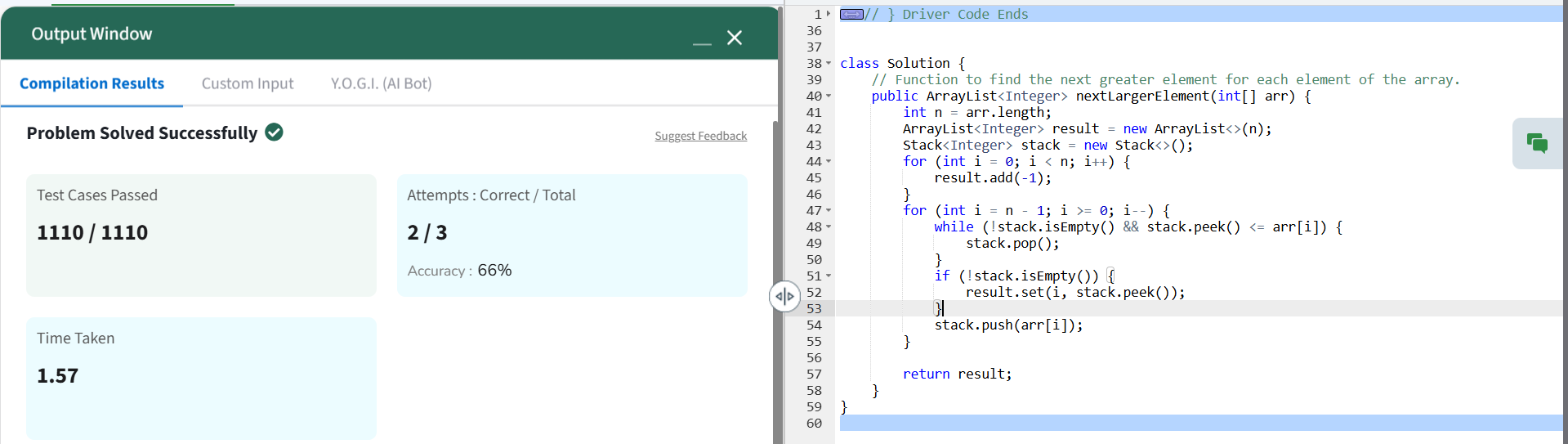
stack.push(arr[i]);

}

return result;

}

}



Time Complexity : O(n)

Space Complexity : O(n)