**Stock buy and sell**

**Difficulty: MediumAccuracy: 29.18%Submissions: 277K+Points: 4**

**The cost of stock on each day is given in an array A[] of size N. Find all the segments of days on which you buy and sell the stock such that the sum of difference between sell and buy prices is maximized. Each segment consists of indexes of two elements, first is index of day on which you buy stock and second is index of day on which you sell stock.  
Note: Since there can be multiple solutions, the driver code will print 1 if your answer is correct, otherwise, it will return 0. In case there's no profit the driver code will print the string "No Profit" for a correct solution.  
  
Example 1:**

**Input:**

**N = 7**

**A[] = {100,180,260,310,40,535,695}**

**Output:**

**1**

**Explanation:**

**One possible solution is (0 3) (4 6)**

**We can buy stock on day 0,**

**and sell it on 3rd day, which will**

**give us maximum profit. Now, we buy**

**stock on day 4 and sell it on day 6.**

class Solution{

ArrayList<ArrayList<Integer> > stockBuySell(int arr[], int n) {

int profit=0,buy=arr[0];

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

if(buy>arr[i]) buy=arr[i];

else if(arr[i]-buy>profit) profit=arr[i]-buy;

}

return profit;

}

}

**Wave Array**

**Difficulty: EasyAccuracy: 63.69%Submissions: 257K+Points: 2**

**Given a sorted array arr[] of distinct integers. Sort the array into a wave-like array(In Place). In other words, arrange the elements into a sequence such that arr[1] >= arr[2] <= arr[3] >= arr[4] <= arr[5].....  
If there are multiple solutions, find the lexicographically smallest one.**

**Note: The given array is sorted in ascending order, and you don't need to return anything to change the original array.**

**Examples:**

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

**Output: [2, 1, 4, 3, 5]**

**Explanation: Array elements after sorting it in the waveform are 2, 1, 4, 3, 5.**

class Solution {

public static void convertToWave(int[] arr) {

int t1=arr[0];int a=arr[arr.length-1];

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

if(i%2==0 && i+1<arr.length){

t1=arr[i];

arr[i]=arr[i+1];

//arr[i]=t1;

}else{

arr[i]=t1;

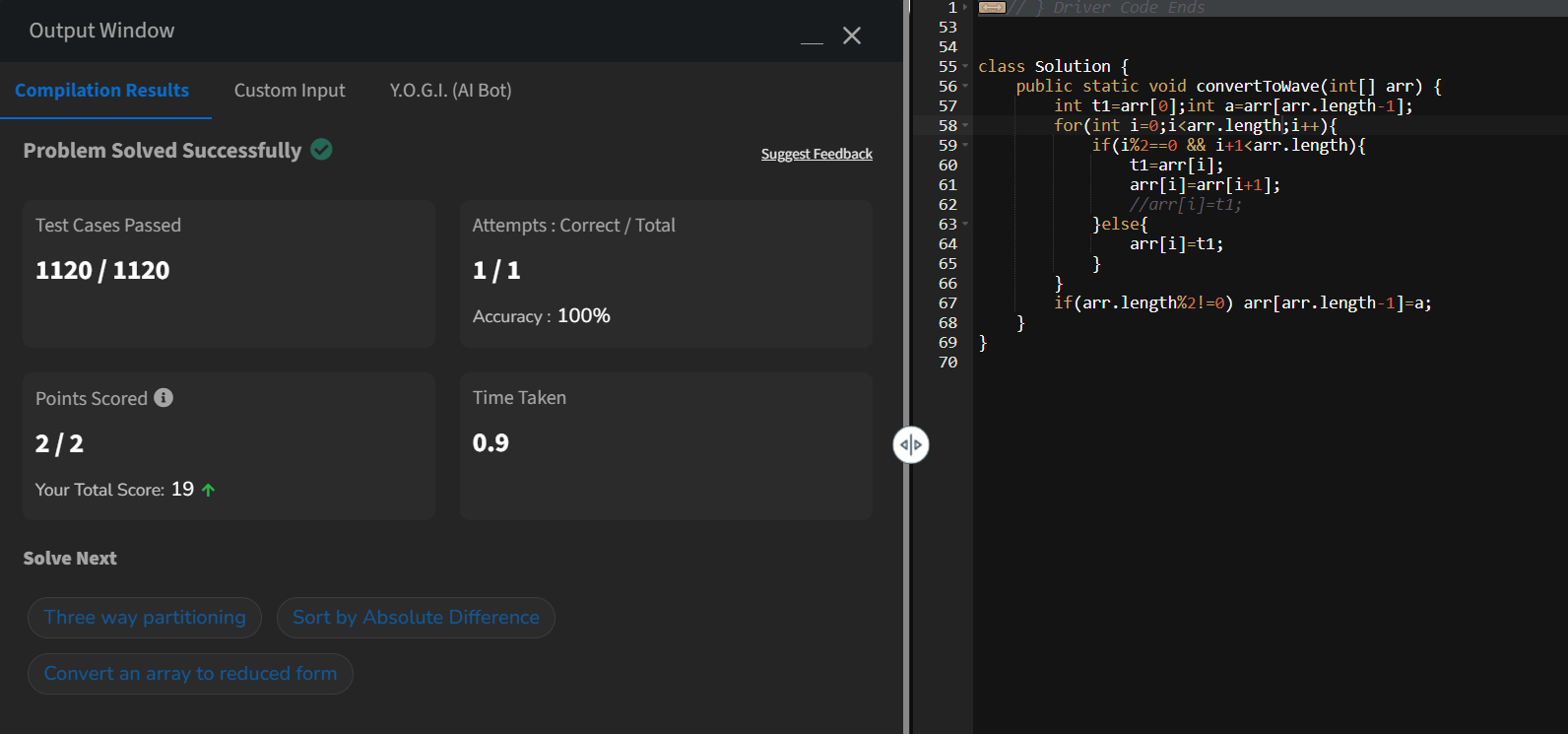
}

}

if(arr.length%2!=0) arr[arr.length-1]=a;

}

}



**Time complexity: O(N)**

**Space complexity:O(1)**

**Remove Duplicates Sorted Array**

**Difficulty: EasyAccuracy: 38.18%Submissions: 259K+Points: 2**

**Given a sorted array arr. Return the size of the modified array which contains only distinct elements.  
*Note:*  
1. Don't use set or HashMap to solve the problem.  
2. You must return the modified array size only where distinct elements are present and modify the original array such that all the distinct elements come at the beginning of the original array.**

**Examples :**

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

**Output: [2]**

**Explanation: After removing all the duplicates only one instance of 2 will remain i.e. [2] so modified array will contains 2 at first position and you should return 1 after modifying the array, the driver code will print the modified array elements.**

class Solution {

// Function to remove duplicates from the given array

public int remove\_duplicate(List<Integer> ar) {

//int op=ar[0];

List<Integer> arr=new ArrayList<>();

arr.add(ar.get(0));

int last=ar.get(0);

for(int i=1;i<ar.size();i++){

int t=ar.get(i);

if(t>last){

arr.add(t);last=t;

}

}

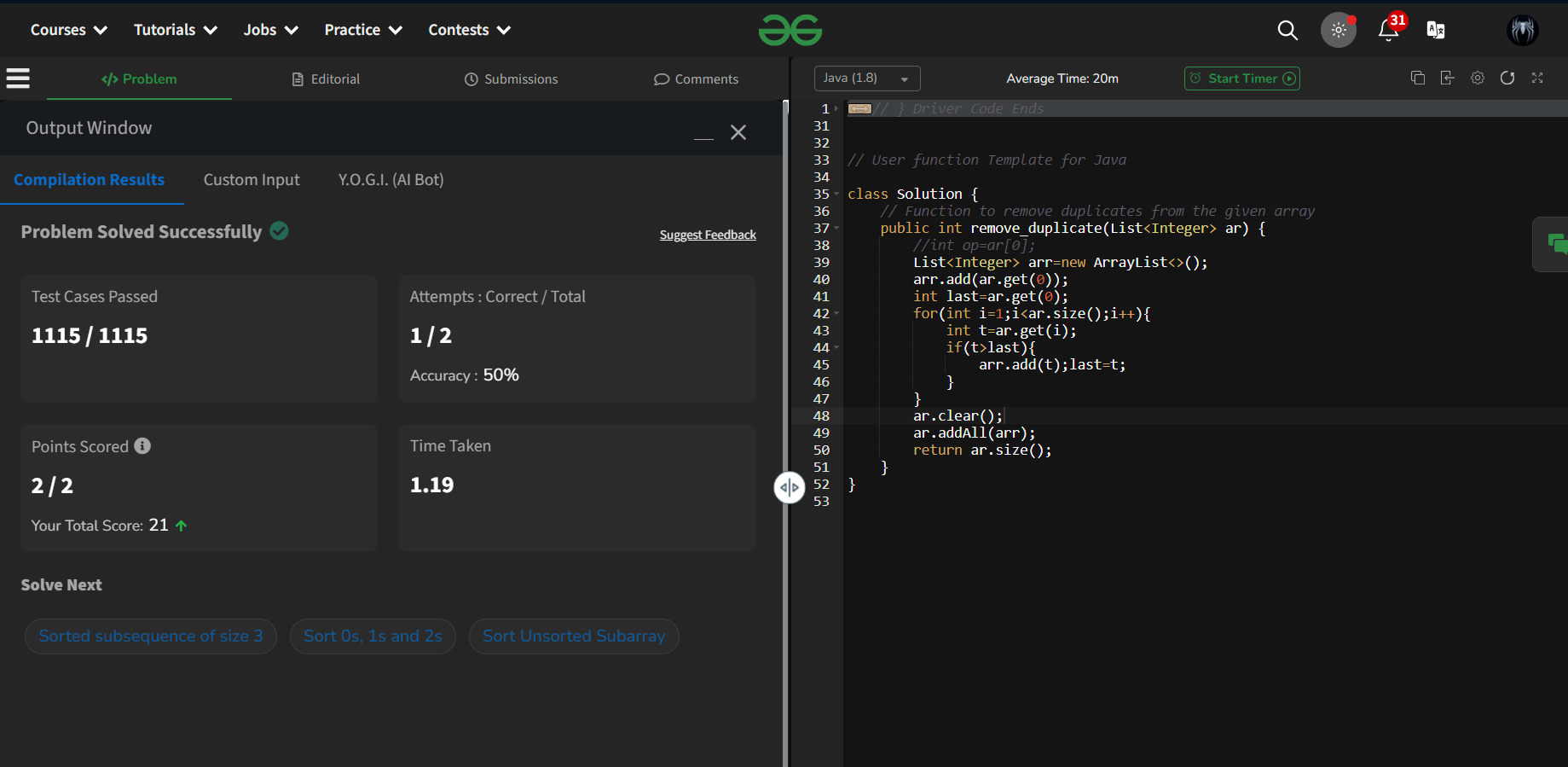
ar.clear();

ar.addAll(arr);

return ar.size();

}

}



**Time complexity: O(N)**

**Space complexity:O(1)**

**First and Last Occurrences**

Difficulty: **Medium**Accuracy: **37.36%**Submissions: **271K+**Points: **4**

Given a sorted array **arr** with possibly some duplicates, the task is to find the first and last occurrences of an element **x** in the given array.  
**Note:** If the number **x** is not found in the array then return both the indices as -1.

**Examples:**

**Input:** arr[] = [1, 3, 5, 5, 5, 5, 67, 123, 125], x = 5

**Output:** [2, 5]

**Explanation**: First occurrence of 5 is at index 2 and last occurrence of 5 is at index 5

class GFG {

ArrayList<Integer> find(int arr[], int x) {

int[] a=new int[2];

a[0]=-1;a[1]=-1;boolean falg=true;

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

if(arr[i]==x){

if(falg){ a[0]=i;a[1]=i;falg=false;}

else a[1]=i;

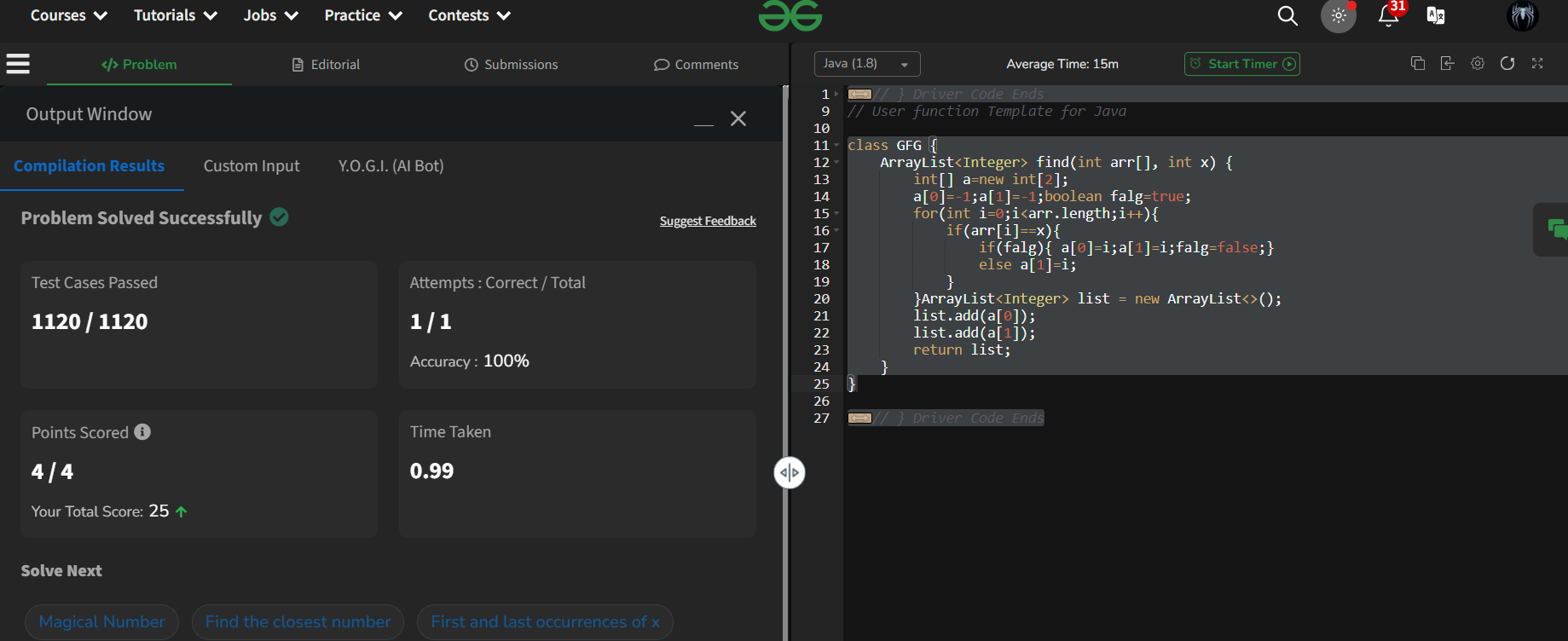
}}ArrayList<Integer> list = new ArrayList<>();

list.add(a[0]);

list.add(a[1]);

return list;

}}



**Time complexity: O(N)**

**Space complexity:O(1)**

**First Repeating Element**

Difficulty: **Easy**Accuracy: **32.57%**Submissions: **268K+**Points: **2**

Given an array **arr[],** find the first repeating element. The element should occur more than once and the index of its first occurrence should be the smallest.

**Note:-**The position you return should be according to 1-based indexing.

**Examples:**

**Input:** arr[] = [1, 5, 3, 4, 3, 5, 6]

**Output:** 2

**Explanation:** 5 appears twice and its first appearance is at index 2 which is less than 3 whose first the occurring index is 3.

class Solution {

public static int firstRepeated(int[] arr) {

HashMap<Integer, Integer> map = new HashMap<>();

int index=Integer.MAX\_VALUE;

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

if (map.containsKey(arr[i])) {

index=Math.min(index,map.get(arr[i]));

}else{

map.put(arr[i], i);

} }

if(index==Integer.MAX\_VALUE){

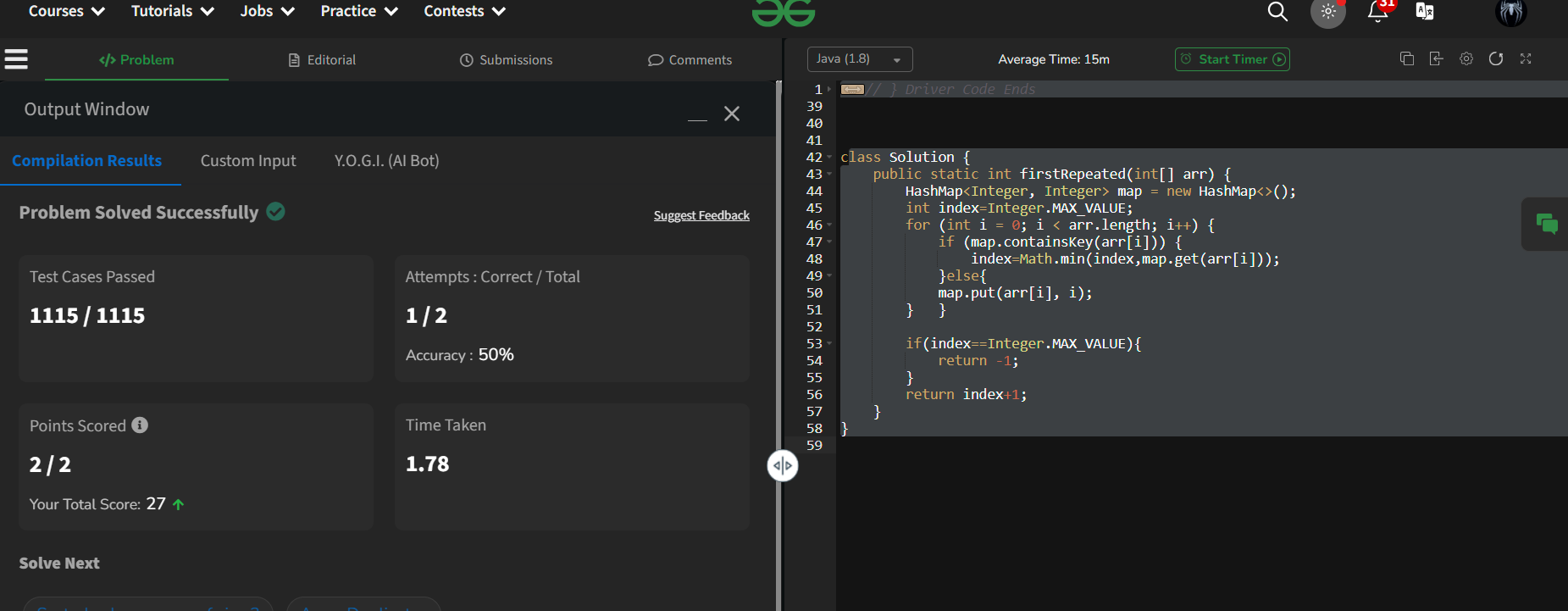
return -1;

}

return index+1;

}

}



**Time complexity: O(N)**

**Space complexity:O(1)**