**STOCK BUY AND SELL**

**CODE :**

class Solution{

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

ArrayList<ArrayList<Integer>> ans = new ArrayList<ArrayList<Integer>>();

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

if(A[i+1]>A[i]){

ArrayList<Integer> al = new ArrayList<>();

al.add(i);

al.add(i+1);

ans.add(al);

}

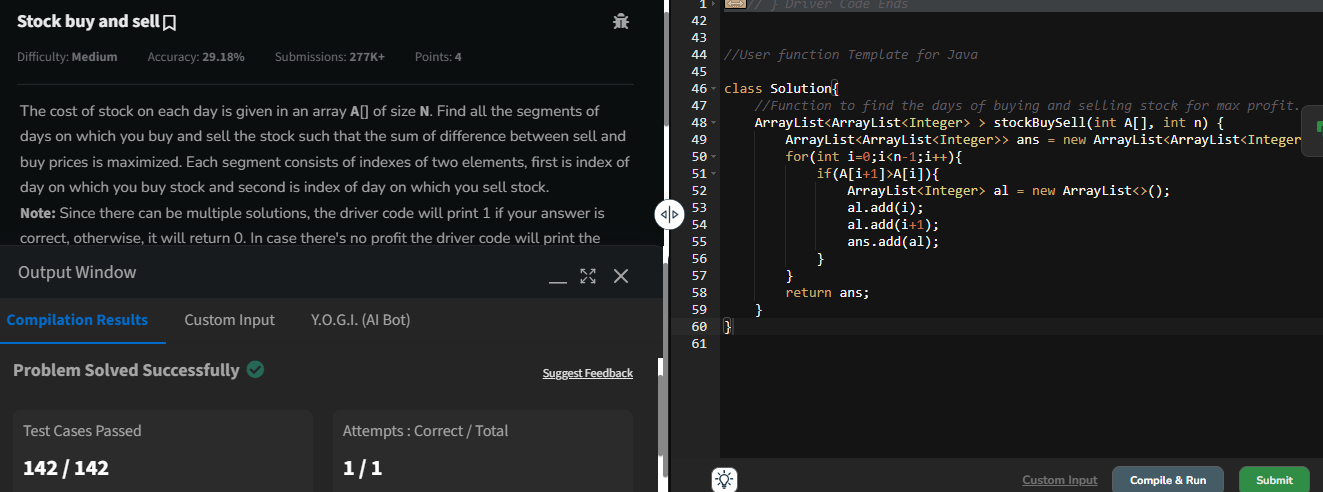
}

return ans;

}

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

**COIN CHANGE (COUNT WAYS)**

**CODE :**

class Solution {

public int count(int coins[], int sum) {

int N = coins.length;

int[][] dp = new int[N+1][sum + 1];

for(int[] t:dp) Arrays.fill(t,-1);

return find(coins,N,sum,0,dp);

}

static int find(int[] c, int n, int sum,int j,int[][] dp) {

if(sum<0||j==n) return 0;

if(sum==0) return 1;

if(dp[j][sum]!=-1) return dp[j][sum];

int s=0;

s+=find(c,n,sum-c[j],j,dp);

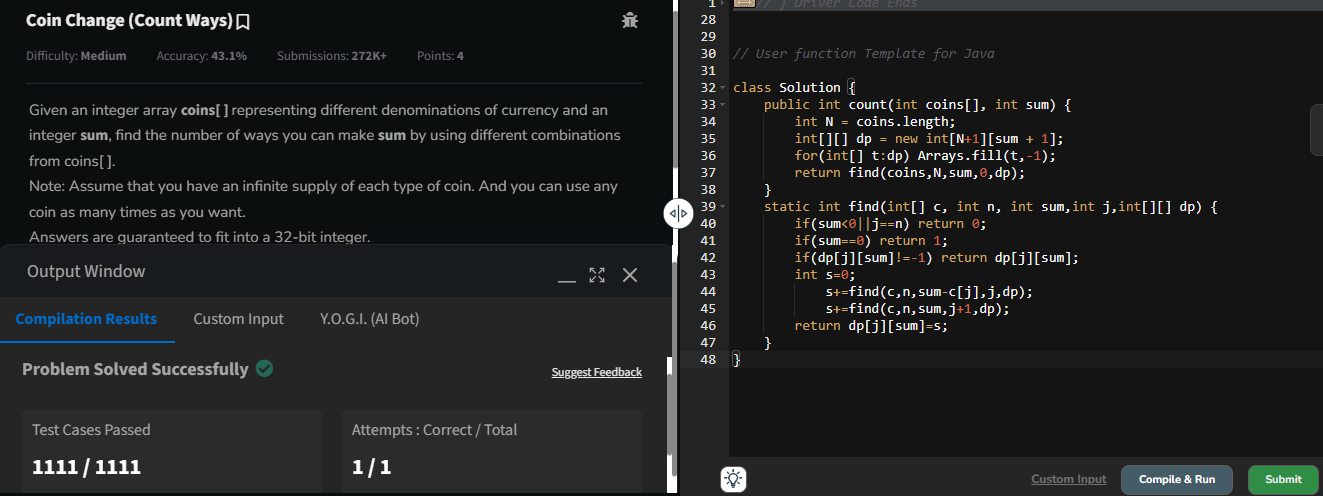
s+=find(c,n,sum,j+1,dp);

return dp[j][sum]=s;

}

}

**OUTPUT :**



TIME COMPLEXITY : O(n\*s)

SPACE COMPLEXITY : O(n\*s)

**FIRST AND LAST OCCURRENCES**

**CODE :**

class GFG {

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

ArrayList<Integer> l = new ArrayList<>();

l.add(-1);

l.add(-1);

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

if(arr[i]==x){

l.set(0,i);

while(i<arr.length && arr[i]==x){

i++;

}

l.set(1,i-1);

}

if(l.get(0)!=-1) break;

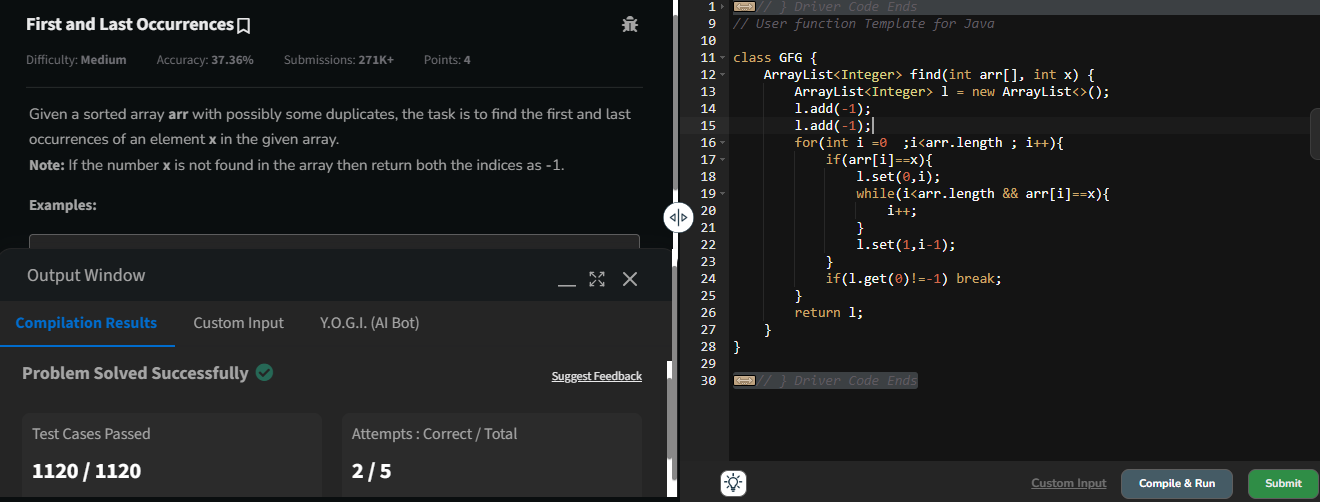
}

return l;

}

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

**FIND TRANSITION POINT**

**CODE :**

class Solution {

int transitionPoint(int arr[]) {

int z = 0;

int o = -1;

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

if(arr[i] == 0){

o = i;

}

if(arr[i]==1 && o!=-1){

return i;

}

if(arr[i]==1) z++;

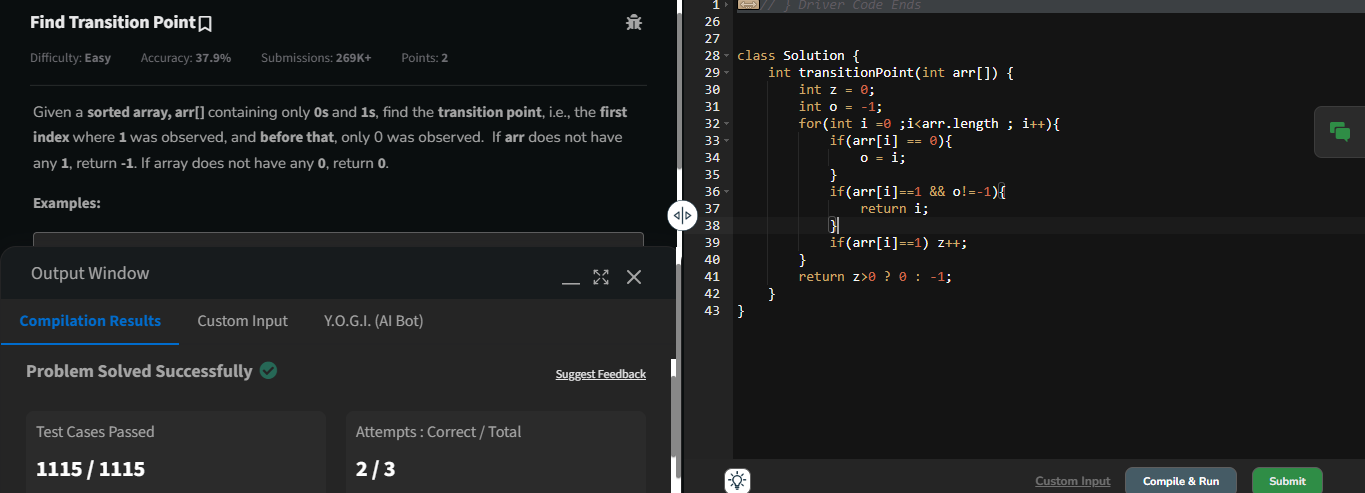
}

return z>0 ? 0 : -1;

}

}

**OUTPUT:**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

**FIRST REPEATING ELEMENT**

**CODE :**

class Solution {

public static int firstRepeated(int[] arr) {

Map<Integer,Integer> hm = new HashMap<>();

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

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

hm.put(arr[i],hm.getOrDefault(arr[i],0)+1);

}else{

hm.put(arr[i],1);

}

}

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

if(hm.get(arr[i]) > 1){

return i+1;

}

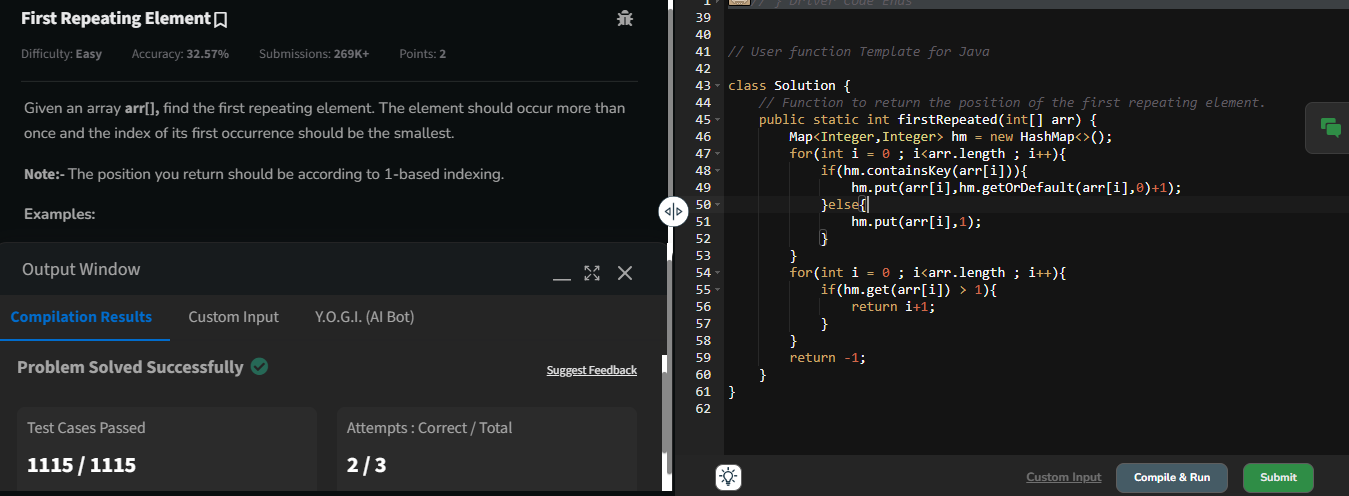
}

return -1;

}

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

**REMOVE DUPLICATES SORTED ARRAY**

**CODE :**

class Solution {

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

int n = arr.size();

int j = 0;

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

if(!arr.get(i).equals(arr.get(j))){

j++;

arr.set(j,arr.get(i));

}

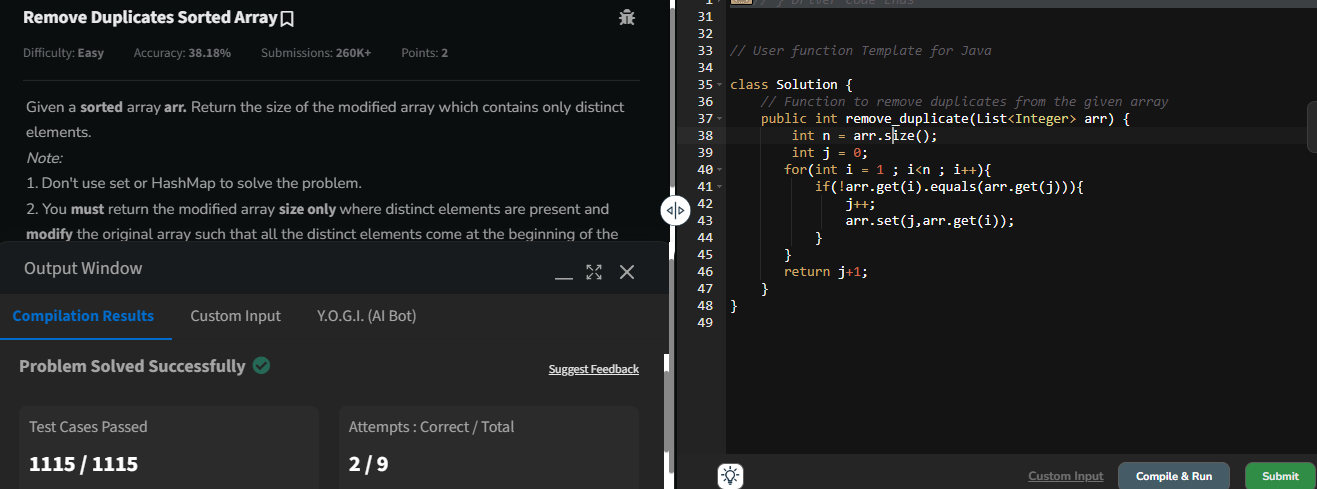
}

return j+1;

}

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)

**MAXIMUM INDEX**

**CODE :**

class Solution {

int maxIndexDiff(int[] a) {

int n = a.length ;

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

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

if(st.isEmpty() || a[st.peek()]<a[i]){

st.push(i);

}

}

int i=0;

int max=0;

while(i<n && st.size()>0){

if(a[i]>a[st.peek()]){

i++;

}else{

max= Math.max(st.peek()-i,max);

st.pop();

}

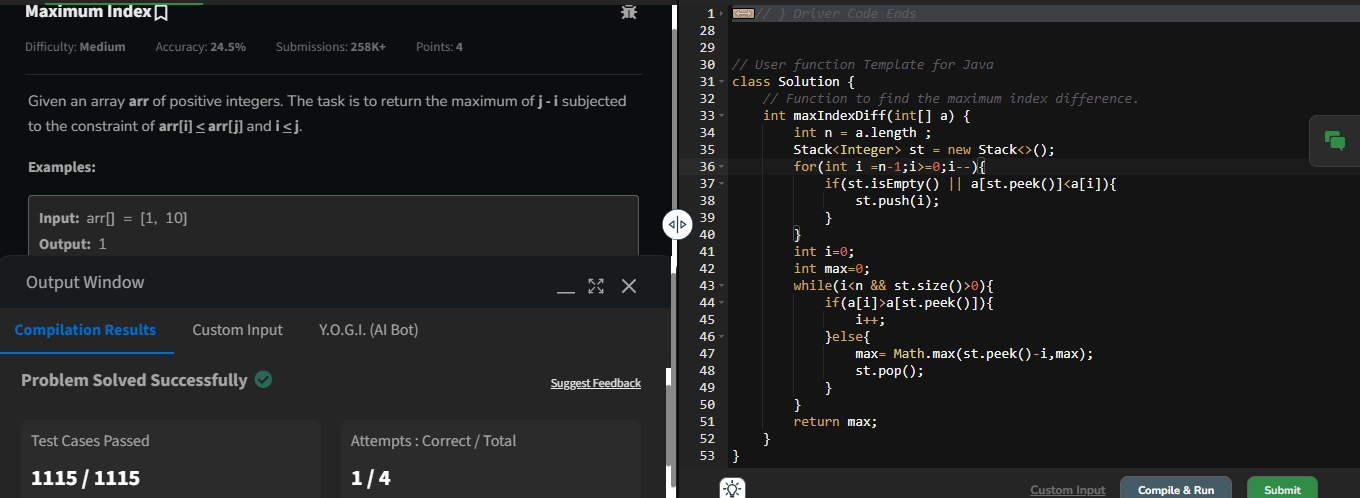
}

return max;

}

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(n)

**WAVE ARRAY**

**CODE :**

class Solution {

public static void convertToWave(int[] arr) {

if(arr.length<2){

return;

}

for(int i=0;i<arr.length-1;i+=2){

int value=arr[i];

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

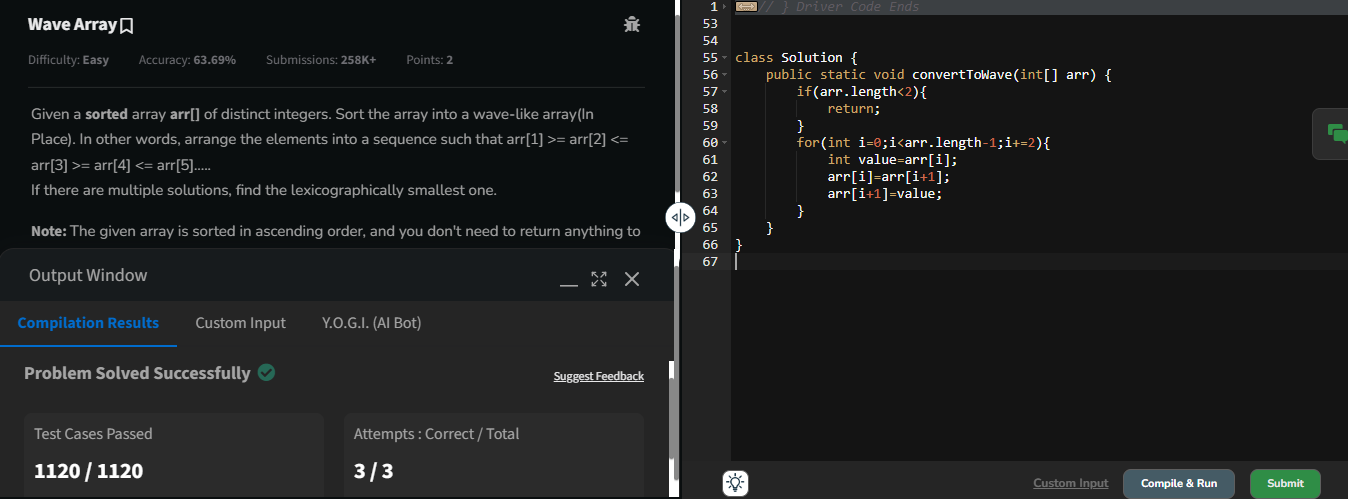
arr[i+1]=value;

}

}

}

**OUTPUT :**



TIME COMPLEXITY : O(n)

SPACE COMPLEXITY : O(1)