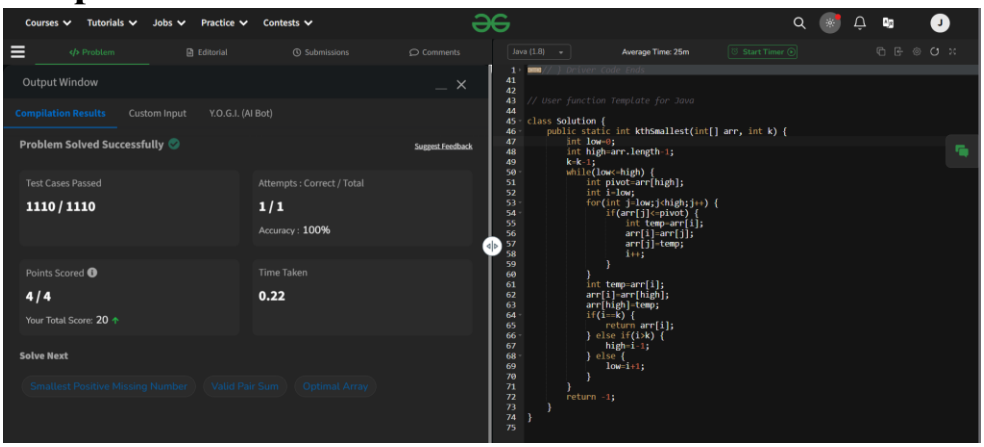
DSA PRACTICE QUESTIONS - DAY 4

Name: Jeevitha R
Reg No: 22IT040
Date: 13/11/2024

1.Kth Smallest Element

```
class Solution {
    public static int kthSmallest(int[] arr, int k) {
        int low=0;
        int high=arr.length-1;
        k=k-1;
        while(low<=high) {
            int pivot=arr[high];
            int i=low;
            for(int j=low;j<high;j++) {
                if(arr[j]<=pivot) {
                    int temp=arr[i];
                    arr[i]=arr[j];
                    arr[j]=temp;
                    i++;
                }
            }
            int temp=arr[i];
            arr[i]=arr[high];
            arr[high]=temp;
            if(i==k) {
                return arr[i];
            } else if(i>k) {
                high=i-1;
            } else {
                low=i+1;
            }
        }
        return -1;
    }
}
```

Output:

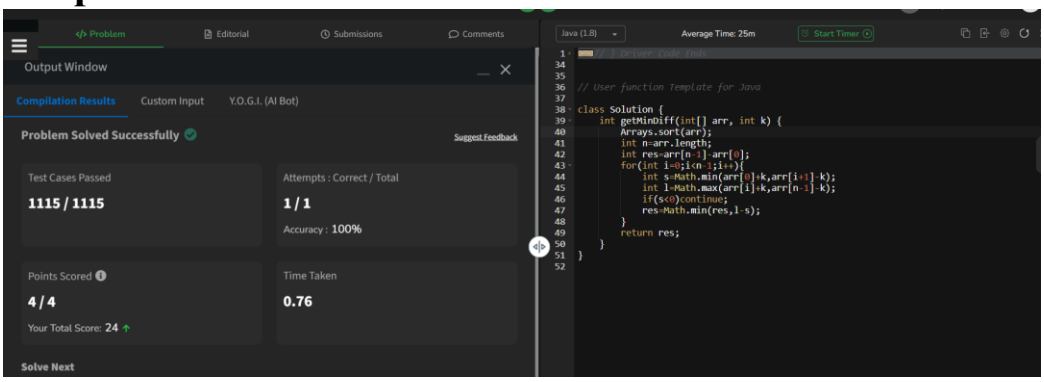


Time complexity: O(n^2)
Space complexity: O(1)

2. Minimize the height II

```
class Solution{
    public int getMinDiff(int[] arr,int k){
        Arrays.sort(arr);
        int n=arr.length;
        int res=arr[n-1]-arr[0];
        for(int i=0;i<n-1;i++){
            int s=Math.min(arr[0]+k,arr[i+1]-k);
            int l=Math.max(arr[i]+k,arr[n-1]-k);
            if(s<0)continue;
            res=Math.min(res,l-s);
        }
        return res;
    }
}
```

Output:



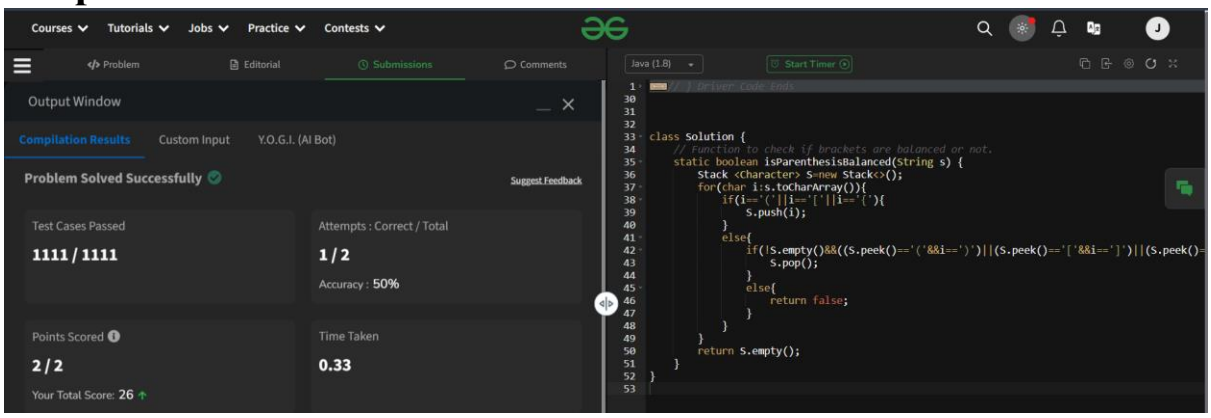
Time complexity: O(n logn)
Space complexity: O(n)
3.Valid Parantheses

```
class Solution{
    boolean valid(String s){
```

```
        Stack <Character> S=new Stack<>();
        for(char i:s.toCharArray()){
            if(i=='('||i=='['||i=='{'){
                S.push(i);
            }
            else{

if(!S.empty()&&((S.peek()=='('&&i==')')||(S.peek()=='['&&i==']')||(S.peek()=='{'&&i=='}')))
{
                S.pop();
            }
            else{
                return false;
            }
        }
    }
    return S.empty();
  }
}
```

Output



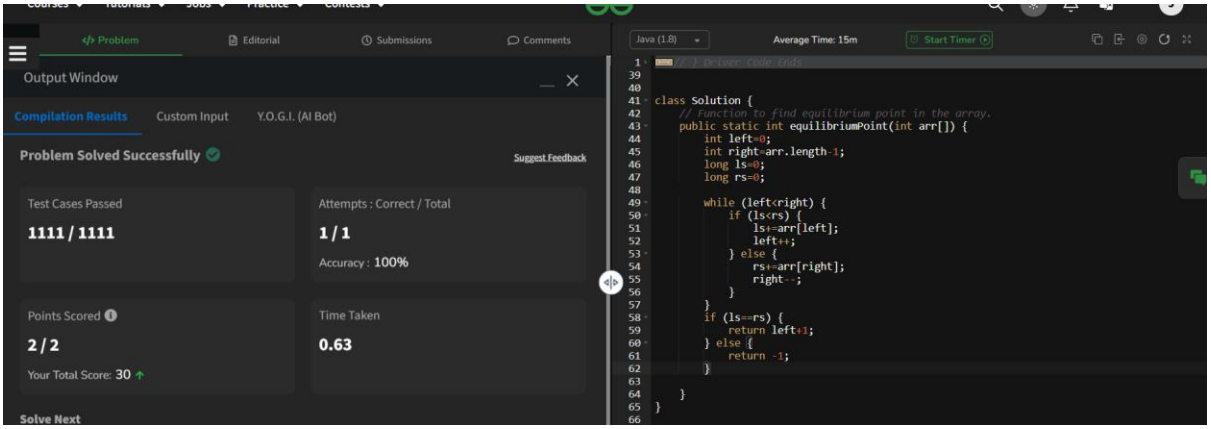
Time complexity: O(n)
Space complexity: O(n)

4. Equilibrium Point

```
class Solution {
    // Function to find equilibrium point in the array.
    public static int equilibriumPoint(int arr[]) {
        // code here
        int left=0;
        int right=arr.length-1;
        long ls=0;
        long rs=0;
```

```
        while (left<right) {
            if (ls<rs) {
                ls+=arr[left];
                left++;
            } else {
                rs+=arr[right];
                right--;
            }
        }
        if (ls==rs) {
            return left+1;
        } else {
            return -1;
        }
    }
}
```

Output



Time complexity: O(n)
Space complexity: O(1)

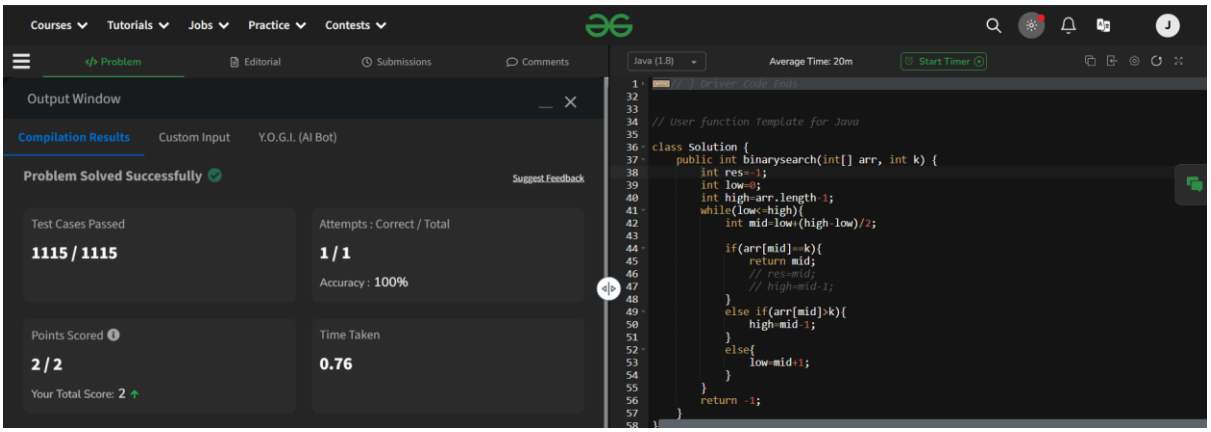
5.Binary Search

```
class Solution {
    public int binarysearch(int[] arr, int k) {
        int res=-1;
        int low=0;
        int high=arr.length-1;
        while(low<=high){
            int mid=low+(high-low)/2;

            if(arr[mid]==k){
                return mid;
                // res=mid;
                // high=mid-1;
            }
            else if(arr[mid]>k){
                high=mid-1;
```

```
        }
        else{
            low=mid+1;
        }
    }
    return -1;
  }
}
```

Output



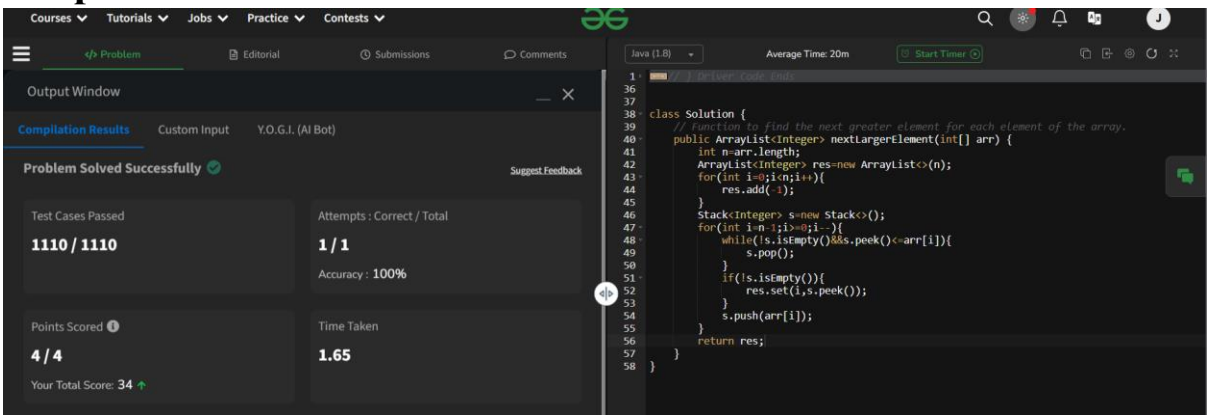
Time complexity:O(log n)
Space complexity: O(1)

6. Next Greater Element

```
class Solution{
  public ArrayList<Integer> nextLargerElement(int[] arr){
      int n=arr.length;
      ArrayList<Integer> res=new ArrayList<>(n);
      for(int i=0;i<n;i++){
          res.add(-1);
      }
      Stack<Integer> s=new Stack<>();
      for(int i=n-1;i>=0;i--){
          while(!s.isEmpty()&&s.peek()<=arr[i]){
              s.pop();
          }
          if(!s.isEmpty()){
              res.set(i,s.peek());
          }
          s.push(arr[i]);
      }
      return res;
  }
```

}

Output

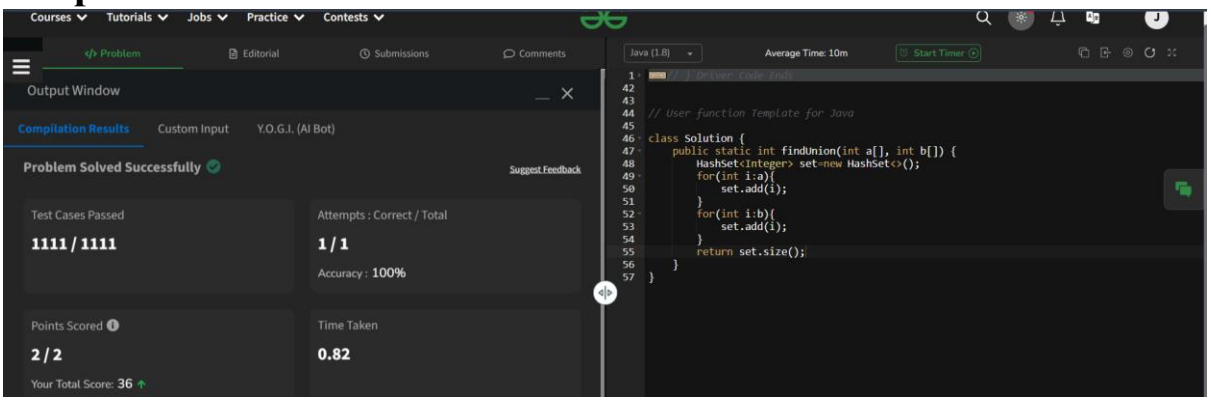


Time complexity: O(n)
Space complexity: O(n)

7. Union of two arrays with duplicate elements

```
class Solution {
    public static int findUnion(int a[], int b[]) {
        // code here
        HashSet<Integer> set=new HashSet<>();
        for(int i:a){
            set.add(i);
        }
        for(int i:b){
            set.add(i);
        }
        return set.size();
    }
}
```

Output



Time complexity: O(n+m)
Space complexity: O(n+m)