

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:

The screenshot shows a LeetCode submission for the problem "Minimize the height II". The "Output Window" on the left indicates that the problem was solved successfully, with 1110/1110 test cases passed, 1/1 attempts correct, 100% accuracy, 4/4 points scored, and a time taken of 0.22. The code on the right is a Java solution using a selection sort algorithm to find the k-th smallest element in an array, which is then used to calculate the minimum difference between the first and last elements of the array.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
```

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:

The screenshot shows a LeetCode submission for the problem "Minimize the height II". The "Output Window" on the left indicates that the problem was solved successfully, with 1115/1115 test cases passed, 1/1 attempts correct, 100% accuracy, 4/4 points scored, and a time taken of 0.76. The code on the right is a Java solution using a selection sort algorithm to find the k-th smallest element in an array, which is then used to calculate the minimum difference between the first and last elements of the array.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
```

Time complexity: $O(n \log n)$

Space complexity: $O(n)$

3. Valid Parantheses

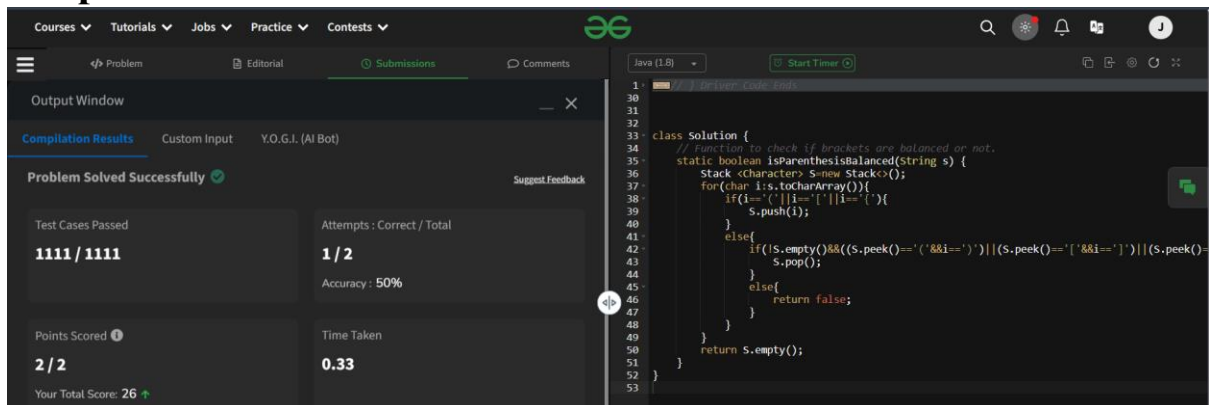
```
class Solution {
    boolean valid(String s) {
```

```

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;
    }
}

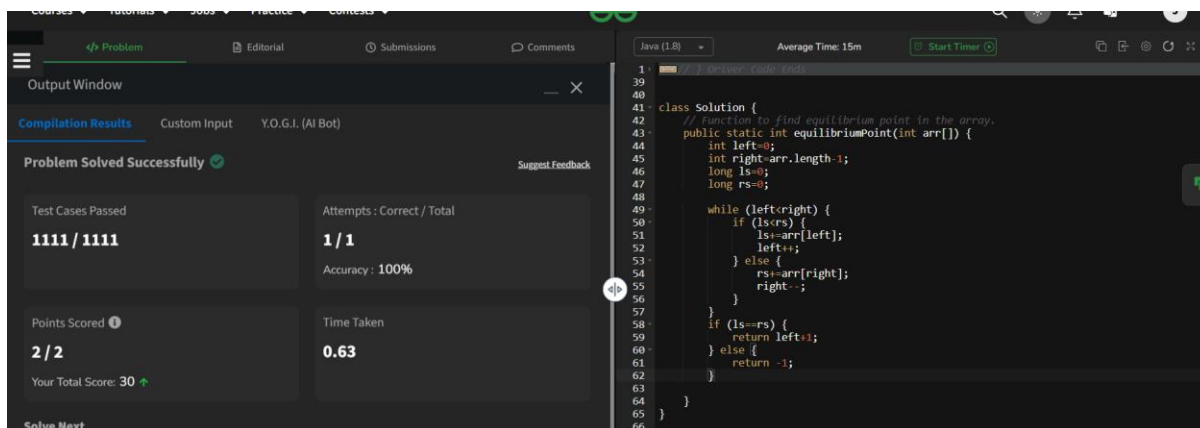
```

```

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;
            }
        }
    }
}

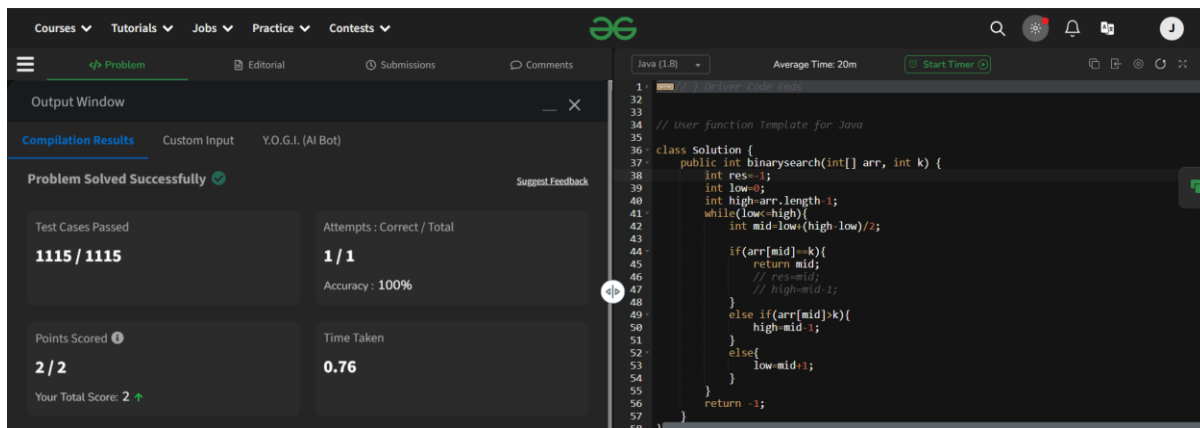
```

```

    }
    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

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1110 / 1110

Attempts : Correct / Total
1 / 1

Accuracy : 100%

Points Scored
4 / 4

Time Taken
1.65

Your Total Score: **34**

```
1 // User Code Starts
36
37
38 class Solution {
39     // Function to find the next greater element for each element of the array.
40     public ArrayList<Integer> nextLargerElement(int[] arr) {
41         int n=arr.length;
42         ArrayList<Integer> res=new ArrayList<>(n);
43         for(int i=0;i<n;i++){
44             res.add(-1);
45         }
46         Stack<Integer> s=new Stack<>();
47         for(int i=n-1;i>=0;i--){
48             while(!s.isEmpty()&&s.peek()<=arr[i]){
49                 s.pop();
50             }
51             if(s.isEmpty()){
52                 res.set(i,s.peek());
53             }
54             s.push(arr[i]);
55         }
56         return res;
57     }
58 }
```

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

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed
1111 / 1111

Attempts : Correct / Total
1 / 1

Accuracy : 100%

Points Scored
2 / 2

Time Taken
0.82

Your Total Score: **36**

```
1 // User Code Starts
42
43
44 // User function Template for Java
45
46 class Solution {
47     public static int findUnion(int a[], int b[]) {
48         HashSet<Integer> set=new HashSet<>();
49         for(int i:a){
50             set.add(i);
51         }
52         for(int i:b){
53             set.add(i);
54         }
55         return set.size();
56     }
57 }
```

Time complexity: $O(n+m)$

Space complexity: $O(n+m)$