

DSA PRACTICE – DAY 4

Name: Deepak S

Reg No: 22IT018

1. Kth Smallest Element in an Unsorted Array

Code Solution:

```
class Solution {
    public static int kthSmallest(int[] arr, int k) {
        int maximum=arr[0];
        for(int i:arr){
            maximum=Math.max(i,maximum);
        }
        int[] count=new int[maximum+1];
        for (int i:arr){
            count[i]+=1;
        }
        int freq=0;
        for (int i=0; i<=maximum; i++){
            if (count[i]!=0){
                freq+=count[i];
                if (freq>=k){
                    return i;}
            }
        }
        return -1;
    }
}
```

Output:

The screenshot shows a coding platform interface. On the left, the 'Output Window' displays the following information:

- Compilation Results:** Custom Input, Y.O.G.I. (AI Bot)
- Problem Solved Successfully** (with a green checkmark icon)
- Test Cases Passed:** 1110 / 1110
- Attempts:** Correct / Total: 1 / 3
- Accuracy:** 33%
- Points Scored:** 4 / 4
- Time Taken:** 0.31
- Your Total Score:** 4 (with a green up arrow icon)
- Solve Next:** Smallest Positive Missing Number, Valid Pair Sum, Optimal Array

On the right, the code editor shows the Java code for the 'Kth Smallest Element in an Unsorted Array' problem. The code is as follows:

```
1 // User function Template for Java
2
3 // User function Template for Java
4
5 class Solution {
6     public static int kthSmallest(int[] arr, int k) {
7         // Your code here
8         int maximum=arr[0];
9
10        for(int i:arr){
11            maximum=Math.max(i,maximum);
12        }
13
14        int[] count=new int[maximum+1];
15
16        for (int i:arr){
17            count[i]+=1;
18        }
19
20        int freq=0;
21        for (int i=0; i<=maximum; i++){
22            if (count[i]!=0){
23                freq+=count[i];
24                if (freq>=k){
25                    return i;}
26            }
27        }
28        return -1;
29    }
30 }
```

Time complexity: $O(m+n)$

Space Complexity: $O(\text{max_element})$

2. Minimize heights II

Code Solution:

```
class Solution {
    int getMinDiff(int[] arr, int k) {
        // code here
        Arrays.sort(arr);
        int n=arr.length;
        int res=arr[n-1]-arr[0];

        for(int i=0; i<n-1; i++){
            int small=Math.min(arr[0]+k, arr[i+1]-k);
            int large=Math.max(arr[i]+k, arr[n-1]-k);
            if (small<0) continue;
            res=Math.min(res, large-small);
        }
        return res;
    }
}
```

Output:

The screenshot displays a coding platform interface. On the left, the 'Output Window' shows 'Compilation Results' with a green checkmark indicating 'Problem Solved Successfully'. It also displays 'Test Cases Passed: 1115 / 1115', 'Attempts: 1 / 2', 'Accuracy: 50%', 'Points Scored: 4 / 4', and 'Time Taken: 0.69'. On the right, the code editor shows the Java solution for the 'Minimize Heights II' problem, which matches the code provided in the 'Code Solution' section. The code is a class 'Solution' with a method 'getMinDiff' that sorts the array and iterates through it to find the minimum difference between the maximum and minimum elements after adding or subtracting 'k' from each element.

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

Space Complexity: $O(n)$

3. Valid Parentheses

Code Solution:

class Solution

```
{
    boolean valid(String s)
    {
        // code here
        Stack <Character> st=new Stack<>();

        for (char i:s.toCharArray()){
            if (i=='(' || i=='[' || i=='{'){
                st.push(i);
            }
            else{
                if(!st.empty() && ((st.peek()=='(' && i==')') || (st.peek()=='[' && i==']') || (st.peek()=='{' && i=='}'))){
                    st.pop();
                }else{
                    return false;
                }
            }
        }
        return true;
    }
}
```

Output:

The screenshot displays a code editor with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1115 / 1115', 'Attempts: 2 / 2', and 'Accuracy: 100%'. The 'Time Taken' is listed as '0.4'. A note at the bottom states: 'You get marks only for the first correct submission if you solve the problem without viewing the full solution.' On the right, the code editor shows the Java code for the 'Valid Parentheses' problem, which matches the code provided in the 'Code Solution' block. The code is numbered from 1 to 51.

Time complexity: $O(n)$

Space Complexity: $O(n)$

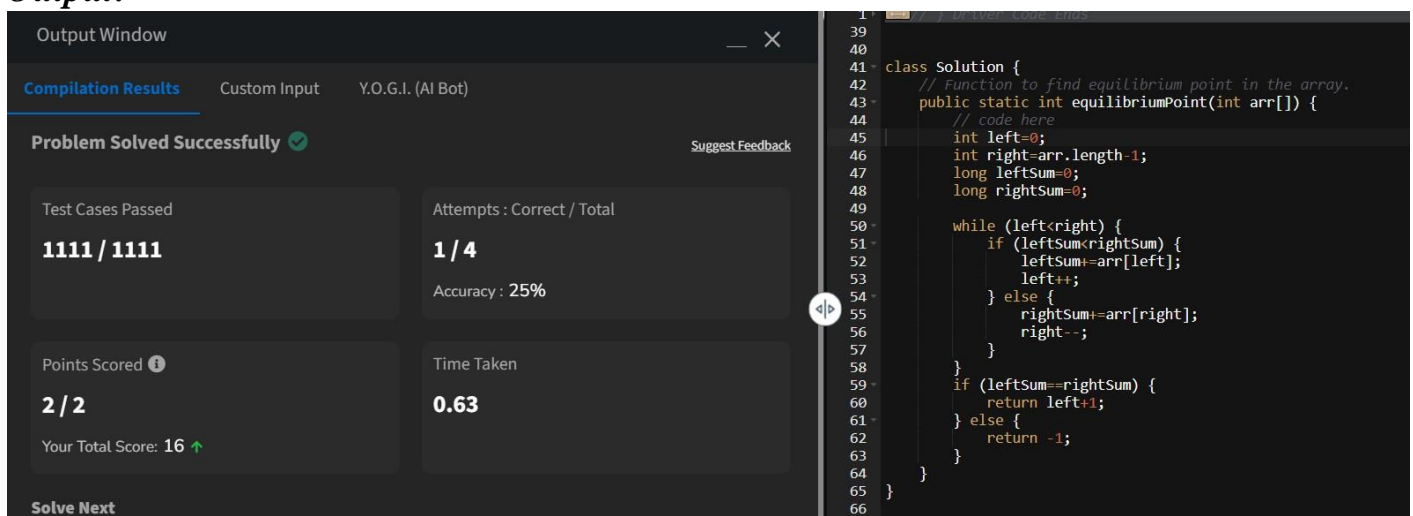
4. Equilibrium Point

Code Solution:

```
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 leftSum=0;
        long rightSum=0;

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

Output:



The screenshot displays a coding platform interface. On the left, the 'Output Window' is open, showing 'Compilation Results' for 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1111 / 1111', 'Attempts: 1 / 4', 'Accuracy: 25%', 'Points Scored: 2 / 2', and 'Your Total Score: 16'. A 'Solve Next' button is at the bottom. On the right, the code editor shows the same Java code as above, with line numbers 39 to 66. A small 'Run' button is visible between the output window and the code editor.

Time Complexity: $O(n)$

Space Complexity: $O(1)$

5. Binary Search

Code Solution:

```
class Solution {
    public int binarysearch(int[] arr, int k) {
        // Code Here
        int low=0;
        int high=arr.length-1;

        while(low<=high){
            int mid=low+(high-low)/2;
            if (arr[mid]==k){
                return mid;
            }
            else if (arr[mid]>k){
                high=mid-1;
            }
            else{
                low=mid+1;
            }
        }
        return -1;
    }
}
```

Output:

The screenshot displays a coding platform interface for the 'Binary Search' problem. The problem description states: 'Given a sorted array `arr` and an integer `k`, find the position (0-based indexing) at which `k` is present in the array using binary search. Note: If multiple occurrences are there, please return the smallest index.' An example is provided: 'Input: `arr[] = [1, 2, 3, 4, 5]`, `k = 4`' and 'Output: 3'. The 'Output Window' shows 'Problem Solved Successfully' with a green checkmark. Below this, it shows 'Test Cases Passed: 1115 / 1115' and 'Attempts: 1 / 2' with an 'Accuracy: 50%'.

On the right side, the code editor shows the Java solution for the problem, which is identical to the code provided in the 'Code Solution' section. The code is as follows:

```
32
33
34 // User function Template for Java
35
36 class Solution {
37     public int binarysearch(int[] arr, int k) {
38         // Code Here
39         int low=0;
40         int high=arr.length-1;
41
42         while(low<=high){
43             int mid=low+(high-low)/2;
44             if (arr[mid]==k){
45                 return mid;
46             }
47             else if (arr[mid]>k){
48                 high=mid-1;
49             }
50             else{
51                 low=mid+1;
52             }
53         }
54         return -1;
55     }
56 }
```

Time Complexity: $O(\log n)$

Space Complexity: $O(1)$

6. Next Greater Element

Code Solution:

```
class Solution {
    // Function to find the next greater element for each element of the array.
    public ArrayList<Integer> nextLargerElement(int[] arr) {
        // code here
        int n=arr.length;
        ArrayList<Integer> result = new ArrayList<>(n);
        for (int i=0; i<n; i++) {
            result.add(-1);
        }

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

Output:

The screenshot displays a coding platform interface. On the left, the 'Output Window' shows 'Compilation Results' with a green checkmark indicating 'Problem Solved Successfully'. It also displays 'Test Cases Passed' as 1110 / 1110, 'Attempts : Correct / Total' as 1 / 1, 'Accuracy : 100%', 'Points Scored' as 4 / 4, and 'Time Taken' as 1.51. On the right, the code editor shows the same Java solution as provided in the text block, with line numbers 36 to 62 visible.

Time Complexity: $O(n)$

Space Complexity: $O(n)$

7. Union of 2 Arrays with Duplicate elements

Code Solution

```
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:

The screenshot displays a coding platform interface. On the left, the problem title is "Union of Two Arrays with Duplicate Elements". Below the title, it shows "Difficulty: Easy", "Accuracy: 42.22%", "Submissions: 387K+", and "Points: 2". The problem description states: "Given two arrays **a[]** and **b[]**, the task is to find the number of elements in the union between these two arrays." It further explains: "The Union of the two arrays can be defined as the set containing distinct elements from both arrays. If there are repetitions, then only one element occurrence should be there in the union." A note says: "Note: Elements are not necessarily distinct." Below this is an "Output Window" section. It shows "Compilation Results" with "Custom Input" and "Y.O.G.I. (AI Bot)" selected. A green checkmark indicates "Problem Solved Successfully". It also shows "Test Cases Passed: 1111 / 1111" and "Attempts : Correct / Total: 1 / 1" with "Accuracy : 100%". On the right, a code editor shows the Java solution for the problem, which matches the code provided in the "Code Solution" section.

Time Complexity: $O(n+m)$

Space Complexity: $O(n+m)$