

DSA PRACTICE – DAY 4

Name: Dhejan R

Reg No: 22IT022

Date: 13/11/2024

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 displays the output of a coding problem solution. On the left, the 'Output Window' shows 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Below this, statistics are shown: 'Test Cases Passed' as 1110 / 1110, 'Attempts: Correct / Total' as 1 / 3, 'Accuracy: 33%', 'Points Scored' as 4 / 4, and 'Time Taken' as 0.31. At the bottom, there are buttons for 'Solve Next' and 'Smallest Positive Missing Number', 'Valid Pair Sum', and 'Optimal Array'. On the right, the code editor shows the Java code for the 'kthSmallest' function, which matches the code provided in the 'Code Solution' section.

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 lists 'Test Cases Passed' as 1115 / 1115, 'Attempts' as 1 / 2, 'Accuracy' as 50%, 'Points Scored' as 4 / 4, and 'Time Taken' as 0.69. Below this, it says 'Your Total Score: 8' with an upward arrow. At the bottom, there are buttons for 'Solve Next' and 'Minimum Jumps', 'A difference of values and indexes', and 'Minimize the Heights I'. 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: $O(n \log n)$

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 interface. On the left, the 'Output Window' shows 'Compilation Results' and 'Problem Solved Successfully' with a green checkmark. It also displays 'Test Cases Passed: 1115 / 1115', 'Attempts: Correct / Total: 2 / 2', 'Accuracy: 100%', and 'Time Taken: 0.4'. On the right, the code editor shows the Java code for the Solution class, which is identical to the code provided in the 'Code Solution' section. The code is written in a dark-themed editor with line numbers visible on the left side of the code block.

Time complexity: $O(n)$

Space Complexity: $O(n)$

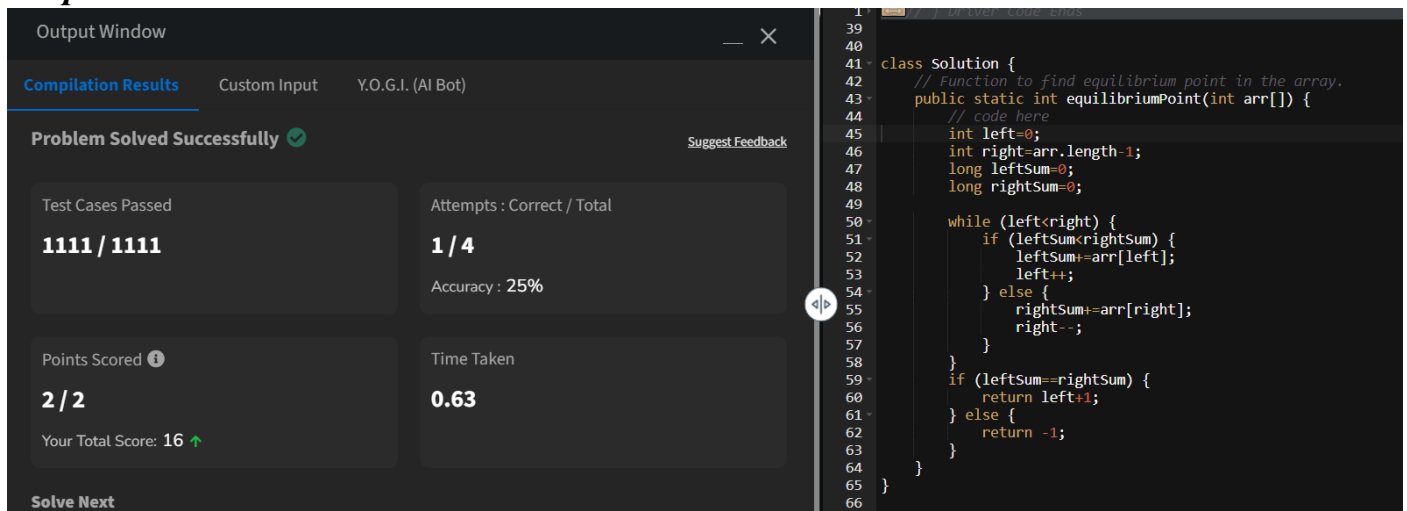
4. Equilibrium Point

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 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)
- Test Cases Passed:** 1111 / 1111
- Attempts:** 1 / 4
- Accuracy:** 25%
- Points Scored:** 2 / 2
- Time Taken:** 0.63
- Your Total Score:** 16 (with a green up arrow)
- Solve Next** button

On the right, the code editor shows the following Java code:

```
39 // Driver Code Ends
40
41 class Solution {
42     // Function to find equilibrium point in the array.
43     public static int equilibriumPoint(int arr[]) {
44         // code here
45         int left=0;
46         int right=arr.length-1;
47         long leftSum=0;
48         long rightSum=0;
49
50         while (left<right) {
51             if (leftSum<rightSum) {
52                 leftSum+=arr[left];
53                 left++;
54             } else {
55                 rightSum+=arr[right];
56                 right--;
57             }
58         }
59         if (leftSum==rightSum) {
60             return left+1;
61         } else {
62             return -1;
63         }
64     }
65 }
66
```

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 'Compilation Results' section shows 'Problem Solved Successfully' with 1115 test cases passed out of 1115, 1 correct attempt out of 2, and 50% accuracy. On the right, a code editor shows the Java solution for the problem, which matches the code provided in the 'Code Solution' section.

Binary Search

Difficulty: Easy Accuracy: 44.32% Submissions: 530K+ Points: 2

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.

Examples:

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

Output: 3

Output Window

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

Problem Solved Successfully

[Suggest Feedback](#)

Test Cases Passed: **1115 / 1115**

Attempts : Correct / Total: **1 / 2**

Accuracy : 50%

```
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 shows a coding platform interface. On the left, the 'Output Window' displays '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: 1110 / 1110', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 4 / 4', and 'Time Taken: 1.51'. At the bottom, it says 'Your Total Score: 22' and 'Solve Next'. On the right, the code editor shows the Java solution for the 'Next Greater Element' problem, which matches the code provided in the previous block. The code is numbered from 1 to 62.

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. At the top, 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. Under "Compilation Results", it says "Problem Solved Successfully" with a green checkmark. A table shows "Test Cases Passed" as "1111 / 1111" and "Attempts : Correct / Total" as "1 / 1". The "Accuracy" is listed as "100%". On the right side of the interface, the Java code solution is visible, matching the code provided in the "Code Solution" section.

Time Complexity: $O(n+m)$

Space Complexity: $O(n+m)$