

# DSA PRACTICEDAY 2

Name: Deepak S

Reg No.: 22IT018

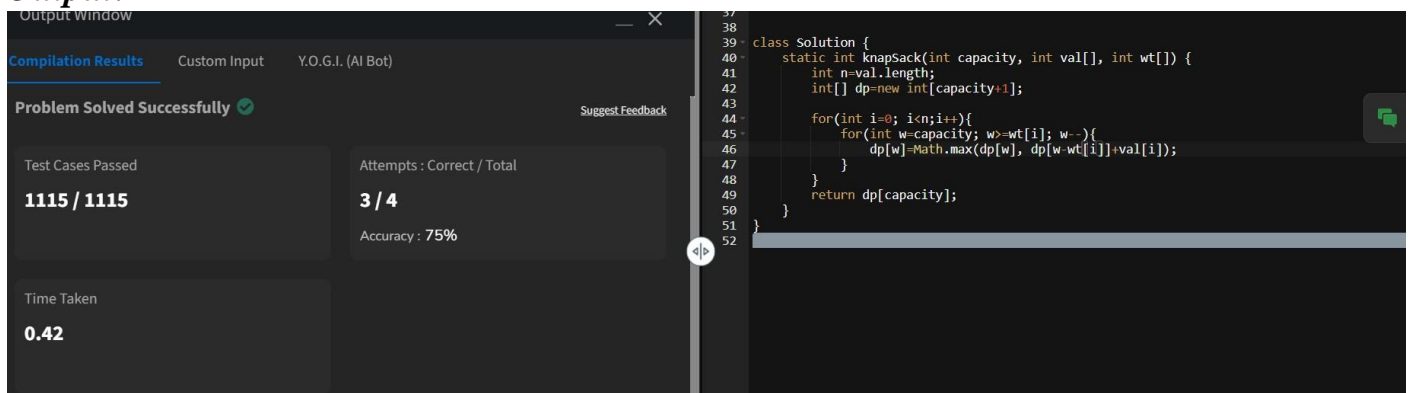
## 1. 0-1 knapsack problem

*Code Solution:*

```
public class knapsack{
    public static int knapsack(int capacity, int[] weight, val[]){
        int n=val.length;
        int[] dp=new int[capacity+1];

        for(int i=0; i<n;i++){
            for(int w=capacity; w>=weights[i]; w--){
                dp[w]=Math.max(dp[w], dp[w-weights[i]]+val[i]);
            }
        }
        return dp[capacity];
    }
}
```

*Output:*



*Time Complexity:*  $O(n \times \text{capacity})$

*Space Complexity:*  $O(\text{Capacity})$

## 2. Floor in sorted array

*Code Solution:*

```
class Solution {
    static int findFloor(int[] arr, int k) {
        int low=0;
        int high=arr.length-1;
        int floorIndex=-1;
```

```

while (low<=high) {
    int mid=low+(high-low)/2;

    if (arr[mid]==k) {
        return mid;
    } else if (arr[mid]<k) {
        floorIndex=mid;
        low=mid+1;
    } else {
        high=mid-1;
    }
}
return floorIndex;
}
}

```

### 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 1111 / 1111, 'Attempts' as 1 / 2, 'Accuracy' as 50%, 'Points Scored' as 2 / 2, and 'Time Taken' as 0.27. A 'Solve Next' button is at the bottom. On the right, a code editor shows the implementation of the 'findFloor' function in Java, which uses a binary search algorithm to find the floor index of a given element in a sorted array.

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

**Space Complexity:  $O(1)$**

### 3. Check equal arrays

#### Code Solution:

```

class Solution {
    public static boolean check(int[] arr1, int[] arr2) {
        if (arr1.length!=arr2.length)
            return false;

        Arrays.sort(arr1);
        Arrays.sort(arr2);
        for (int i=0; i<arr1.length; i++)

```

```

        if (arr1[i]!=arr2[i])
            return false;

    return true;
}
}

```

### Output:

The screenshot displays a coding platform interface. On the left, the 'Output Window' shows a 'Compilation Results' tab with a green checkmark indicating 'Problem Solved Successfully'. It lists 'Test Cases Passed' as 1116 / 1116, 'Attempts : Correct / Total' as 1 / 1, 'Accuracy : 100%', and 'Time Taken' as 0.19. Below this, it shows 'Points Scored' as 1 / 1 and 'Your Total Score: 11'. On the right, the code editor shows a Java solution for checking if two arrays are equal. The code includes a comment '// User function Template for Java' and a class 'Solution' with a method 'check' that sorts the arrays and compares them element by element.

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

*Space Complexity:  $O(1)$*

## 4. Palindrome linked list

### Code Solution:

```

class Solution {
    // Function to check whether the list is palindrome.
    boolean isPalindrome(Node head) {
        // Your code here
        if (head==null || head.next==null) return true;

        Node slow=head;
        Node fast=head;

        while (fast!=null && fast.next!=null) {
            slow=slow.next;
            fast=fast.next.next;
        }

        Node prev=null;
        Node curr=slow;
        while (curr!=null) {
            Node nextNode=curr.next;
            curr.next=prev;
            prev=curr;
        }
    }
}

```

```

        curr=nextNode;
    }

    Node first=head;
    Node second=prev;
    boolean isPalindrome=true;
    while (second!=null) {
        if (first.data!=second.data) {
            isPalindrome=false;
            break;
        }
        first=first.next;
        second=second.next;
    }

    return isPalindrome;
}
}

```

### Output:

The screenshot shows a coding platform interface with two main panels. The left panel, titled 'Output Window', displays the results of a submission. It indicates that the problem was solved successfully, with 1112 out of 1112 test cases passed. The user's attempt was 1 out of 1, with 100% accuracy. The points scored were 4 out of 4, and the time taken was 1.85 seconds. Below this, there are suggestions for 'Solve Next' problems, including 'Intersection Point in Y Shaped Linked Lists', 'Flattening a Linked List', and 'Longest Palindrome in Linked List'. The right panel shows the Java code for the solution, which implements a function to check if a linked list is a palindrome using a two-pointer technique (slow and fast pointers) and a stack to store the values of the first half of the list.

```

79
80 class Solution {
81     // Function to check whether the list is palindrome.
82     boolean isPalindrome(Node head) {
83         // Your code here
84         if (head==null || head.next==null) return true;
85
86         Node slow=head;
87         Node fast=head;
88
89         while (fast!=null && fast.next!=null) {
90             slow=slow.next;
91             fast=fast.next.next;
92         }
93
94         Node prev=null;
95         Node curr=slow;
96         while (curr!=null) {
97             Node nextNode=curr.next;
98             curr.next=prev;
99             prev=curr;
100            curr=nextNode;
101        }
102
103        Node first=head;
104        Node second=prev;
105        boolean isPalindrome=true;
106        while (second!=null) {
107            if (first.data!=second.data) {
108                isPalindrome=false;
109                break;
110            }
111            first=first.next;
112            second=second.next;
113        }
114
115        return isPalindrome;
116    }
117 }
118

```

*Time Complexity:  $O(n)$*

*Space Complexity:  $O(1)$*

## 5. Balanced tree check

### *Code Solution:*

```
class Tree
{
    //Function to check whether a binary tree is balanced or not.
    boolean isBalanced(Node root)
    {
        if (root==null)
            return true;
        int lh=height(root.left);
        if (lh==-1)
            return false;
        int rh=height(root.right);
        if (rh==-1)
            return false;
        if (Math.abs(lh-rh)>1)
            return false;

        return true;
    }

    public static int height(Node root) {
        if (root==null)
            return 0;
        int leftHeight=height(root.left);
        if (leftHeight==-1)
            return -1;
        int rightHeight=height(root.right);
        if (rightHeight==-1)
            return -1;
        if (Math.abs(leftHeight-rightHeight)>1)
            return -1;

        return Math.max(leftHeight, rightHeight)+1;
    }
}
```

## Output:

The screenshot shows a coding platform interface with a dark theme. On the left, the 'Output Window' displays 'Compilation Results' for a problem solved successfully. It shows 'Test Cases Passed: 1120 / 1120', 'Attempts: 1 / 1', 'Accuracy: 100%', 'Points Scored: 2 / 2', and 'Time Taken: 0.44'. Below this, 'Solve Next' buttons are visible for 'Height of Binary Tree', 'Minimum Depth of a Binary Tree', and 'Array to BST'. On the right, the code editor shows a Java solution for checking if a binary tree is balanced. The code defines a 'Tree' class with an 'isBalanced' method and a static 'height' method. The 'isBalanced' method uses recursive calls to check the balance of the left and right subtrees. The 'height' method calculates the height of the tree. The code is in Java 1.8 and has an average time of 20m.

*Time Complexity:  $O(n)$*

*Space Complexity:  $O(n)$*

## 6. Triplet sum in array

### Code Solution:

```
class Solution {  
    // Should return true if there is a triplet with sum equal  
    // to x in arr[], otherwise false  
    public static boolean find3Numbers(int arr[], int n, int x) {  
        // Your code Here  
        Arrays.sort(arr);  
        for (int i=0; i<n-2; i++){  
            int l=i+1;  
            int r=n-1;  
            while(l<r){  
                int a=arr[i]+arr[l]+arr[r];  
                if(a==x){  
                    return true;  
                }  
                else if (a<x){  
                    l++;  
                }  
                else{  
                    r--;  
                }  
            }  
        }  
    }  
}
```

```

    }
    return false;
}
}

```

## Output:

The screenshot displays the 'Output Window' of a coding platform. The 'Compilation Results' tab is active, showing 'Problem Solved Successfully' with a green checkmark. The test results are as follows:

Test Cases Passed	Attempts : Correct / Total
125 / 125	1 / 1

Additional statistics shown include:

- Points Scored: 4 / 4
- Your Total Score: 19 (with an upward arrow indicating improvement)
- Time Taken: 0.16

At the bottom, there are buttons for 'Solve Next', 'Sort Elements by Decreasing Frequency', and 'Zero Sum Subarrays'.

On the right side of the screenshot, the Java code for the solution is visible. It includes a class named 'Solution' with a method 'find3Numbers' that implements a sorting-based approach to find three numbers in an array that sum up to a target value 'x'.

```

30
31
32 // User function Template for Java
33
34 class Solution {
35     // Should return true if there is a triplet with sum equal
36     // to x in arr[], otherwise false
37     public static boolean find3Numbers(int arr[], int n, int x) {
38         // Your code here
39         Arrays.sort(arr);
40         for (int i=0; i<n-2; i++){
41             int l=i+1;
42             int r=n-1;
43             while(l<r){
44                 int a=arr[i]+arr[l]+arr[r];
45                 if(a==x){
46                     return true;
47                 }
48                 else if (a<x){
49                     l++;
50                 }
51                 else{
52                     r--;
53                 }
54             }
55         }
56         return false;
57     }
58 }
59

```

**Time Complexity:**  $O(n^2)$

**Space Complexity:**  $O(1)$

