DSA PRACTICE
DAY 2

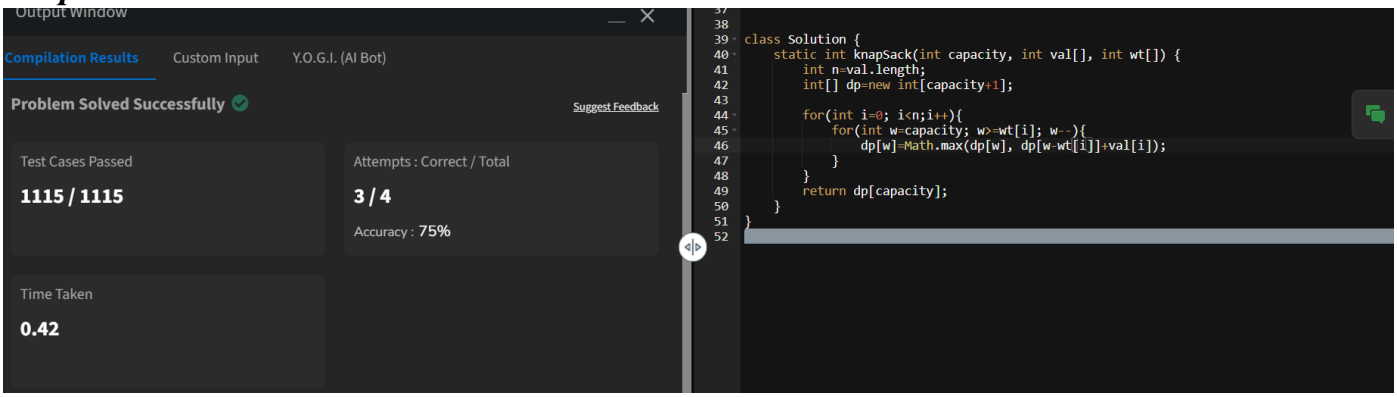
Name: Dhejan R
Reg No.: 22IT022
Date: 11/11/2024

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*capacity)
Space Complexity: O(Capacity)

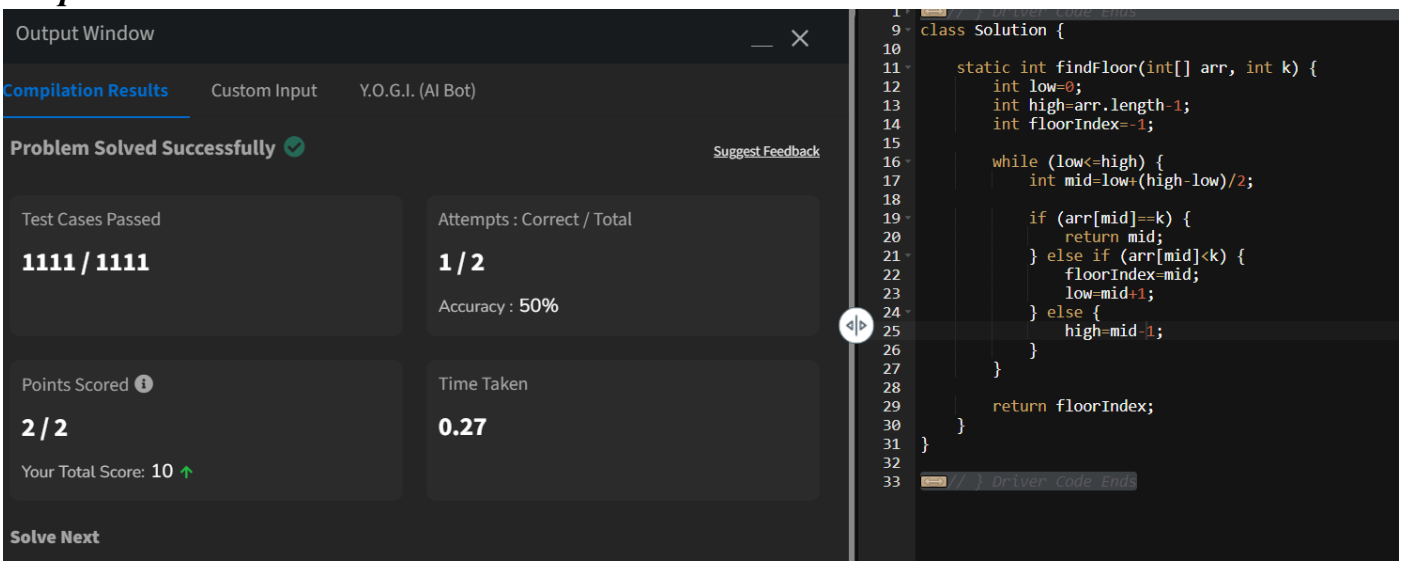
2. Floor in sorted array

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:



Time Complexity: O(Logn)
Space Complexity: O(1)

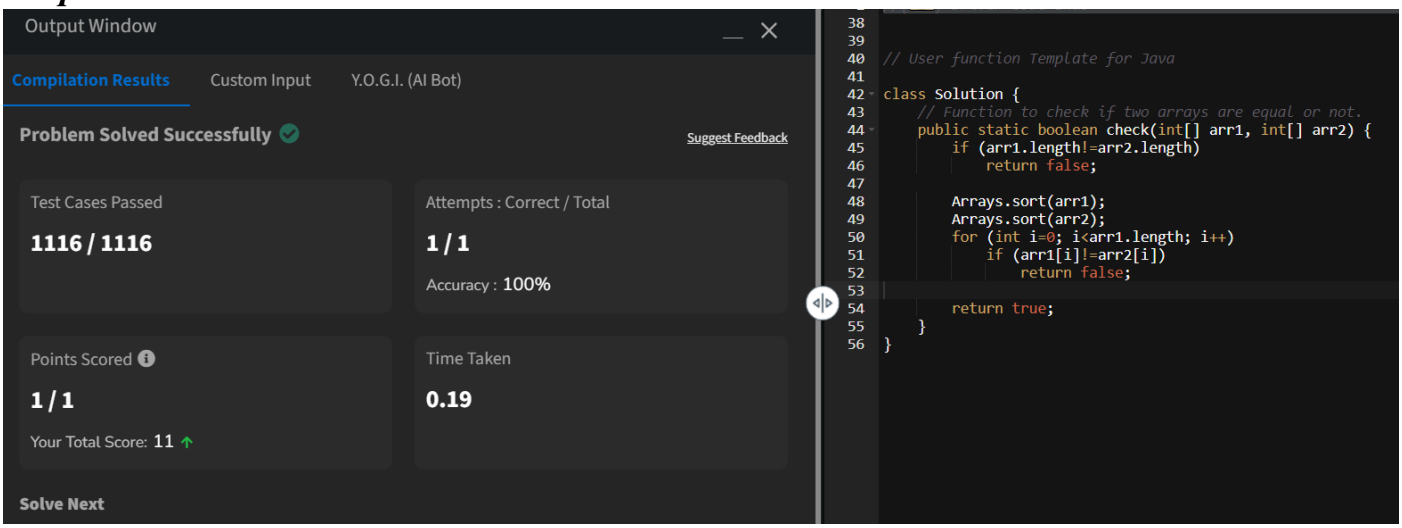
3. Check equal arrays

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:



Time Complexity: O(n logn)
Space Complexity: O(1)

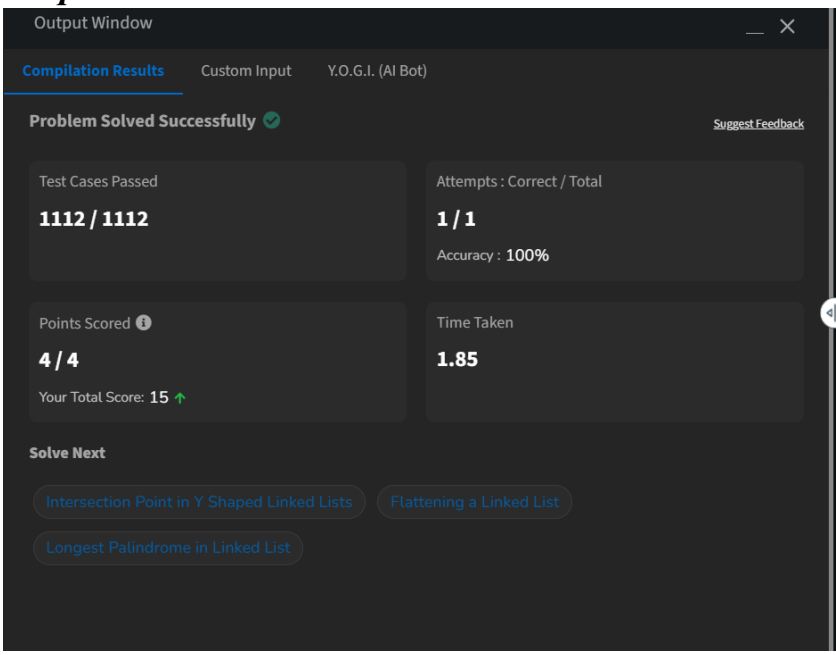
4. Palindrome linked list

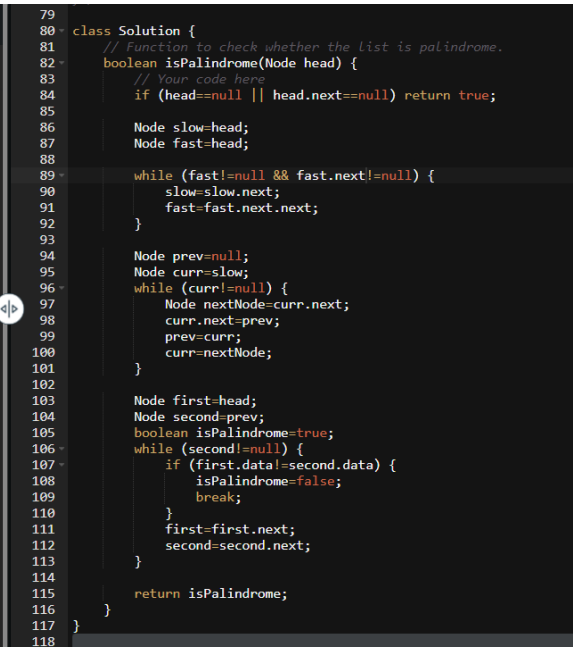
Code Solution:

```
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:





```
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;
    }
}
```

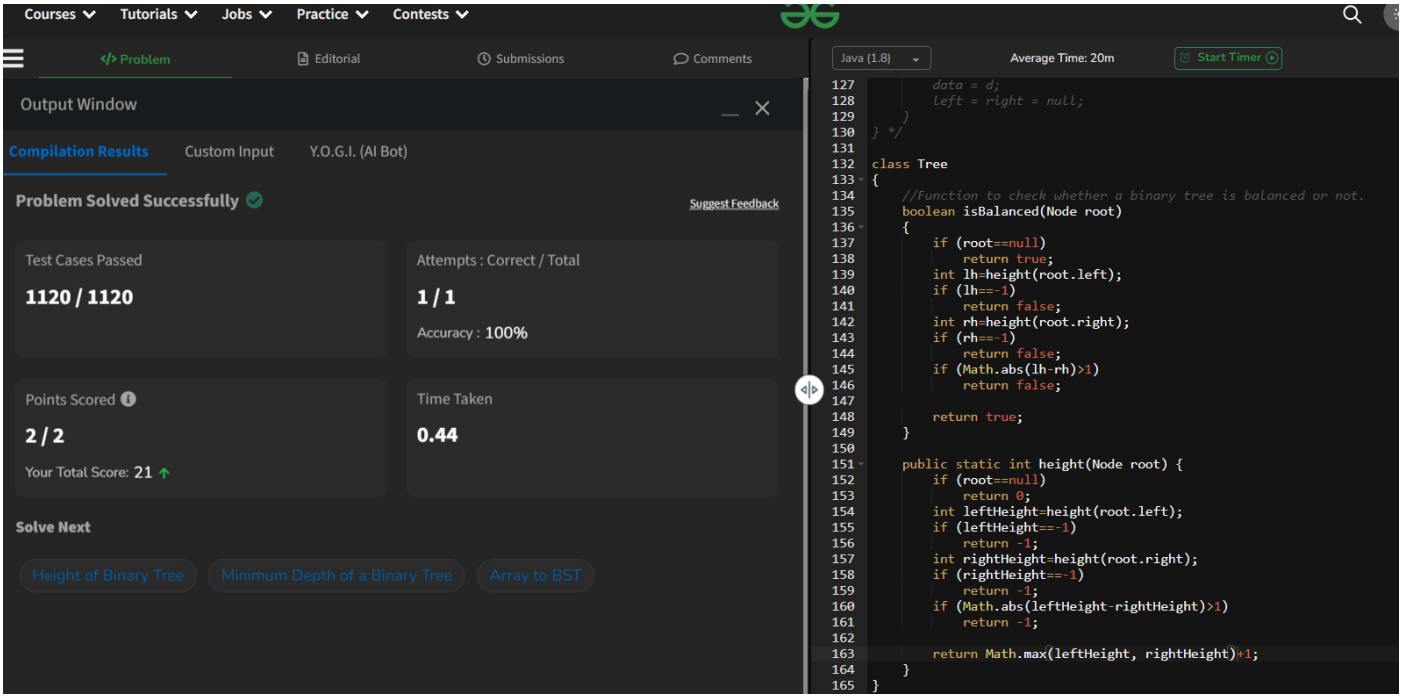
Time Complexity: O(n)
Space Complexity: O(1)

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:



Time Complexity: O(n)
Space Complexity: O(n)

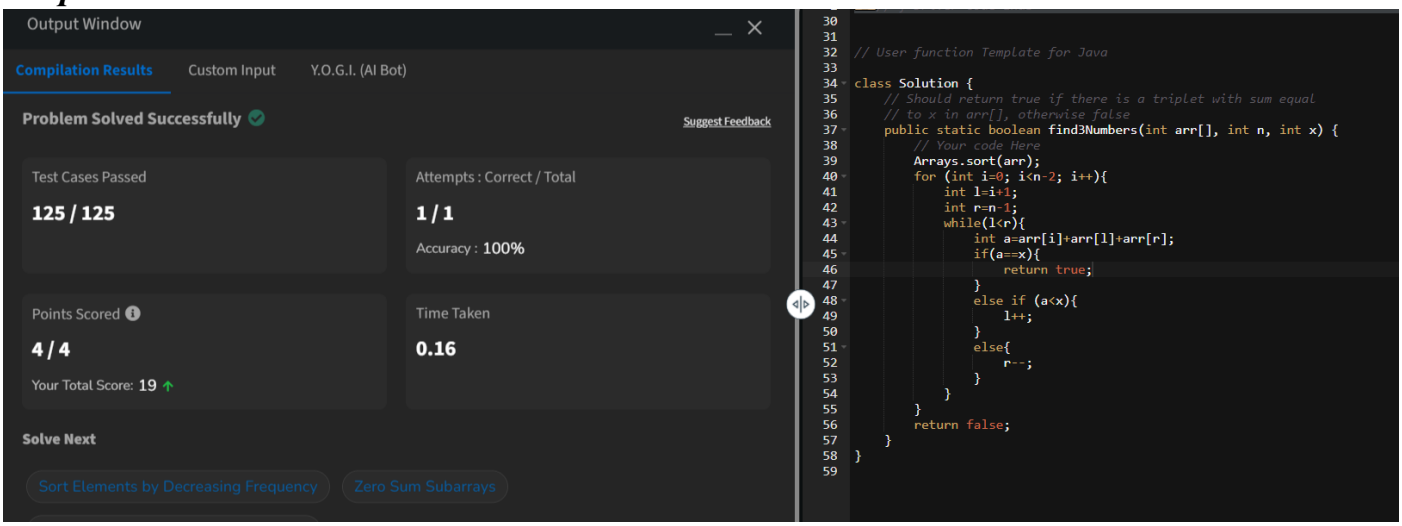
6. Triplet sum in array

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:



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