PRACTICE SET 10 25.11.24

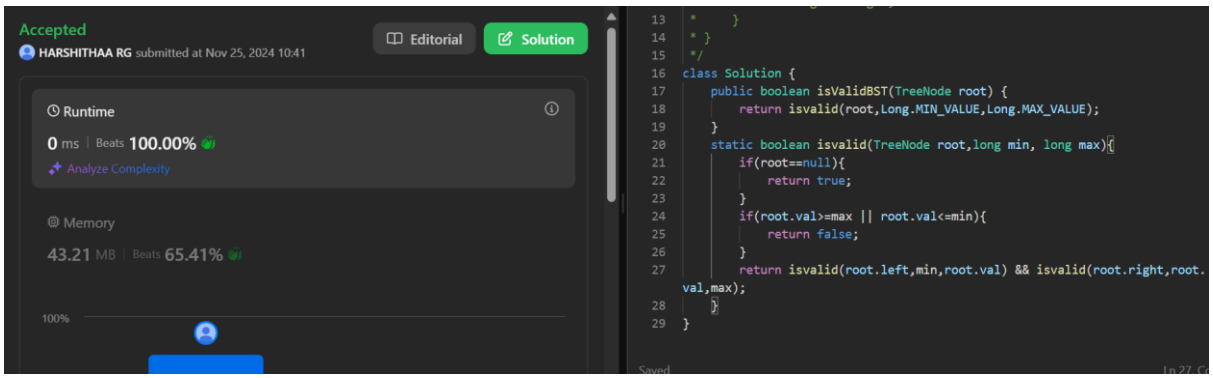
NAME: Harshithaa RG

REG. NO. 22IT035

1.Validate Binary Search Tree

Code:

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return isvalid(root,Long.MIN_VALUE,Long.MAX_VALUE);
    }
    static boolean isvalid(TreeNode root,long min, long max){
        if(root==null){
            return true;
        }
        if(root.val>=max || root.val<=min){
            return false;
        }
        return isvalid(root.left,min,root.val) && isvalid(root.right,root.val,max);
    }
}
```



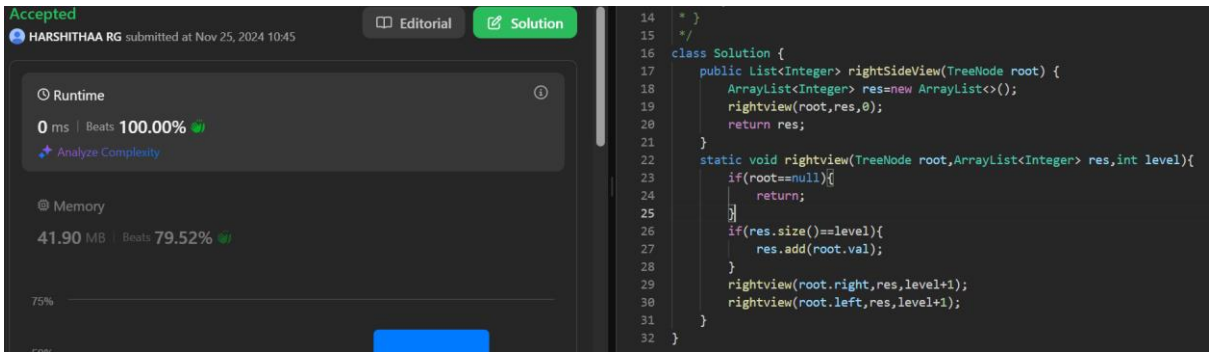
Time Complexity: O(N)

Space Complexity: O(1)

2.Right view of a binary search tree

Code:

```
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        ArrayList<Integer> res=new ArrayList<>();
        rightview(root,res,0);
        return res;
    }
    static void rightview(TreeNode root,ArrayList<Integer> res,int level){
        if(root==null){
            return;
        }
        if(res.size()==level){
            res.add(root.val);
        }
        rightview(root.right,res,level+1);
        rightview(root.left,res,level+1);
    }
}
```



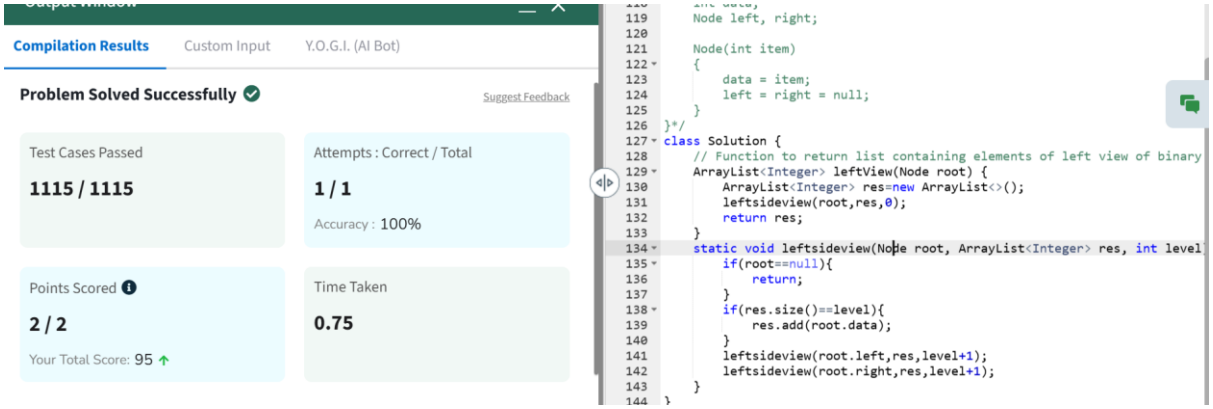
Time Complexity: O(N)

Space Complexity: O(1)

3.Left view of a BST

Code:

```
class Solution {
    // Function to return list containing elements of left view of binary tree.
    ArrayList<Integer> leftView(Node root) {
        ArrayList<Integer> res=new ArrayList<>();
        leftsideview(root,res,0);
        return res;
    }
    static void leftsideview(Node root, ArrayList<Integer> res, int level){
        if(root==null){
            return;
        }
        if(res.size()==level){
            res.add(root.data);
        }
        leftsideview(root.left,res,level+1);
        leftsideview(root.right,res,level+1);
    }
}
```



Time Complexity: O(N)

Space Complexity: O(1)

4. Binary search tree implementation (Recursion)

Code:

```
class Node{
    int data;
    Node left;
    Node right;
    public Node(int val){
        data=val;
        left=null;
        right=null;
    }
}
class Main {
    public static Node insert(Node root,int val){
        if(root==null){
            return new Node(val);
        }
        if(val<root.data){
```

```
        root.left=insert(root.left,val);
    }
    else{
        root.right=insert(root.right,val);
    }
    return root;

}
public static void inorder(Node root){
    if(root==null){
        return;
    }
    inorder(root.left);
    System.out.print(root.data+" ");
    inorder(root.right);
}
public static void main(String[] args) {
    Node root=null;
    root=insert(root,50);
    root=insert(root,30);
    root=insert(root,20);
    root=insert(root,40);
    root=insert(root,70);
    root=insert(root,60);
    root=insert(root,80);
    inorder(root);
}
```

}

```
20 30 40 50 60 70 80
=== Code Execution Successful ===
```

Time Complexity: O(N)

Space Complexity: O(1)

5.Binary Search Tree Implementation (Using collections)

Code:

```
import java.util.TreeSet;
class Main {
    public static void main(String[] args) {
        TreeSet<Integer> bst=new TreeSet<>();
        bst.add(50);
        bst.add(30);
        bst.add(20);
        bst.add(70);
        bst.add(40);
        bst.add(80);
        bst.add(60);
        System.out.println(bst);
    }
}
```

```
[20, 30, 40, 50, 60, 70, 80]

=== Code Execution Successful ===
```

Time Complexity: O(logN)

Space Complexity: O(N)

6. Top view of a BST

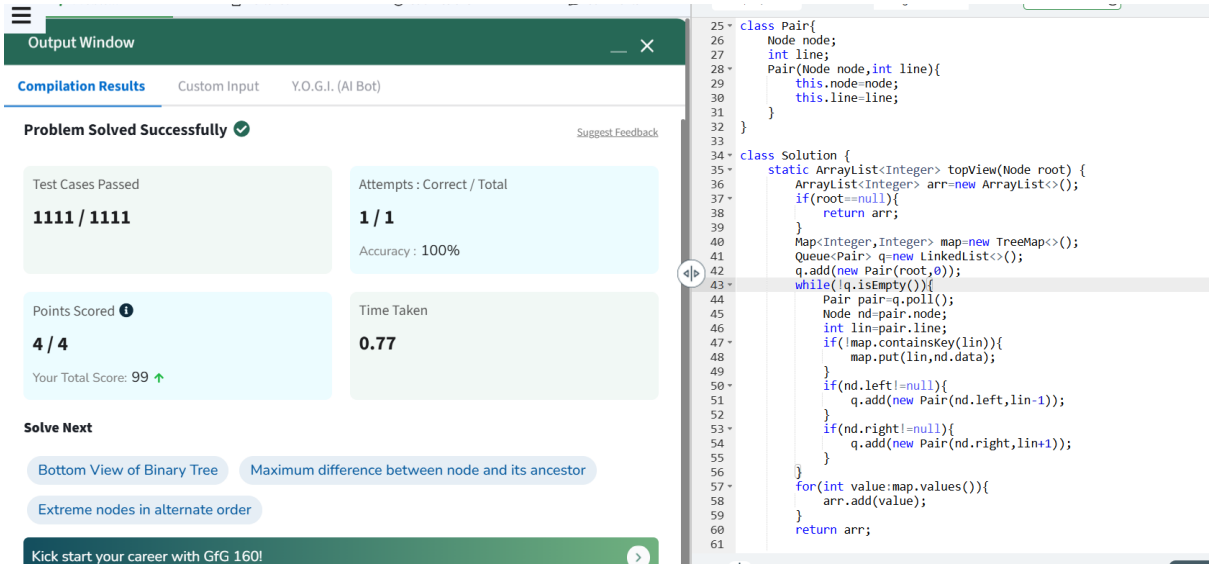
Code:

```
class Pair{
    Node node;
    int line;
    Pair(Node node,int line){
        this.node=node;
        this.line=line;
    }
}

class Solution {
    static ArrayList<Integer> topView(Node root) {
        ArrayList<Integer> arr=new ArrayList<>();
        if(root==null){
            return arr;
        }
        Map<Integer,Integer> map=new TreeMap<>();
        Queue<Pair> q=new LinkedList<>();
        q.add(new Pair(root,0));
        while(!q.isEmpty()){
            Pair pair=q.poll();
            Node nd=pair.node;
            int lin=pair.line;
```

```
            if(!map.containsKey(lin)){
                map.put(lin,nd.data);
            }
            if(nd.left!=null){
                q.add(new Pair(nd.left,lin-1));
            }
            if(nd.right!=null){
                q.add(new Pair(nd.right,lin+1));
            }
        }
        for(int value:map.values()){
            arr.add(value);
        }
        return arr;

    }
}
```



Time Complexity: O(N)

Space Complexity: **O(N)**

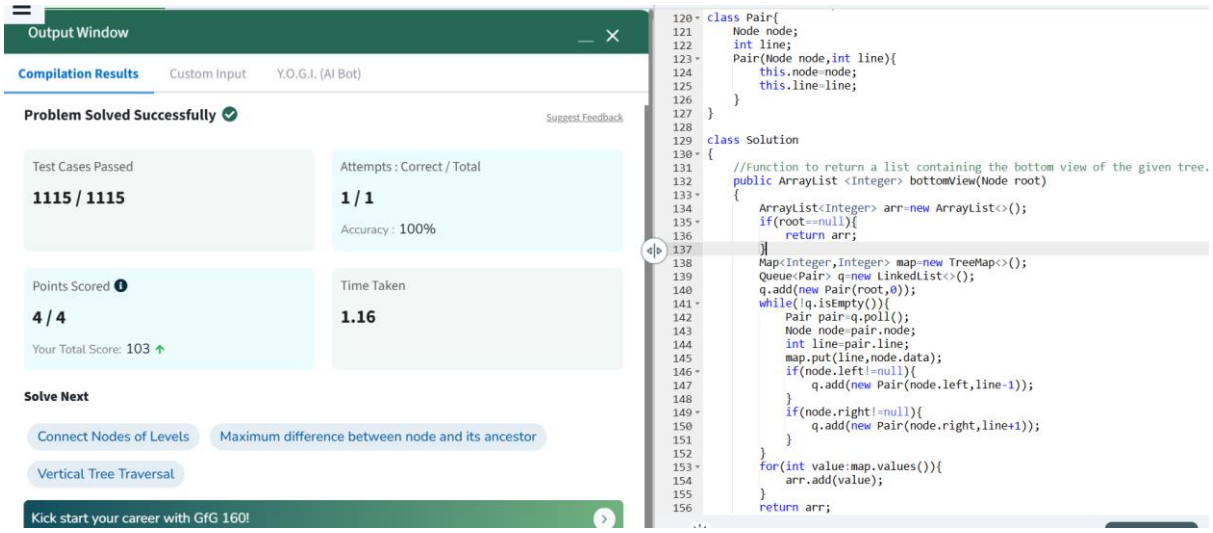
7.Bottom View of a Binary Tree

Code:

```
class Pair{
    Node node;
    int line;
    Pair(Node node,int line){
        this.node=node;
        this.line=line;
    }
}

class Solution
{
    //Function to return a list containing the bottom view of the given tree.
    public ArrayList <Integer> bottomView(Node root)
    {
        ArrayList<Integer> arr=new ArrayList<>();
        if(root==null){
            return arr;
        }
        Map<Integer,Integer> map=new TreeMap<>();
        Queue<Pair> q=new LinkedList<>();
        q.add(new Pair(root,0));
        while(!q.isEmpty()){
            Pair pair=q.poll();
            Node node=pair.node;
            int line=pair.line;
```

```
            map.put(line,node.data);
            if(node.left!=null){
                q.add(new Pair(node.left,line-1));
            }
            if(node.right!=null){
                q.add(new Pair(node.right,line+1));
            }
        }
        for(int value:map.values()){
            arr.add(value);
        }
        return arr;
    }
}
```



Time Complexity: **O(N)**

Space Complexity: O(N)