

## PRACTICE SET 10 25.11.24

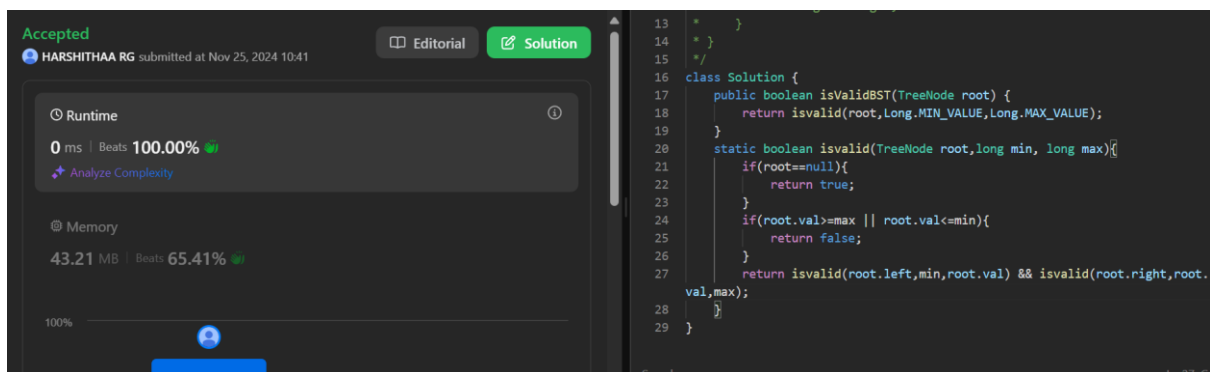
NAME: Harshithaa RG

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);  
    }  
}
```



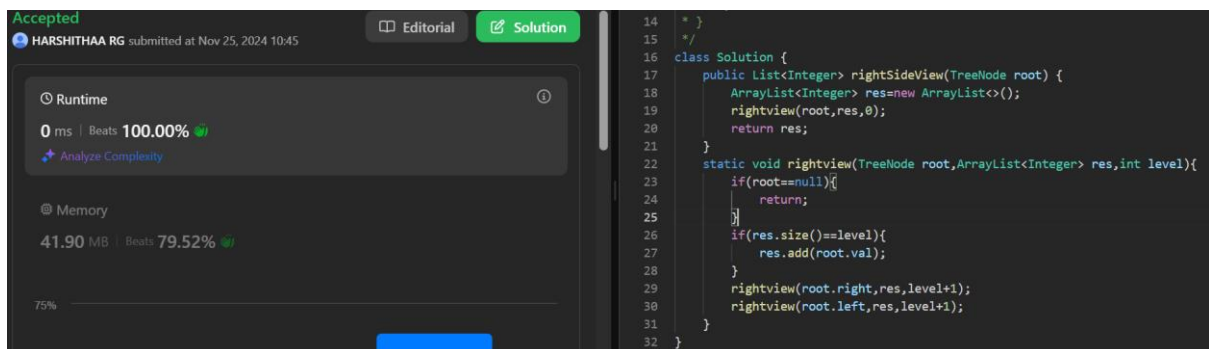
Time Complexity:  $O(N)$

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);  
    }  
}
```



The screenshot displays a code editor interface. On the left, a sidebar shows the submission status: 'Accepted' in green, followed by the user 'HARSHITHAA RG' and the submission time 'submitted at Nov 25, 2024 10:45'. Below this, the 'Runtime' section indicates '0 ms' and 'Beats 100.00%' with a green checkmark, and a link to 'Analyze Complexity'. The 'Memory' section shows '41.90 MB' and 'Beats 79.52%' with a green checkmark. On the right, the code editor shows the Java code for the 'rightSideView' method, which is identical to the code provided in the text above. The code is written in a dark-themed editor with syntax highlighting.

**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);  
    }  
}
```

The screenshot shows a coding competition interface. On the left, the 'Compilation Results' tab is active, displaying 'Problem Solved Successfully' with a green checkmark. Below this, statistics are shown: 'Test Cases Passed' as 1115 / 1115, 'Attempts: Correct / Total' as 1 / 1, 'Accuracy: 100%', 'Points Scored' as 2 / 2, and 'Time Taken' as 0.75. A 'Your Total Score: 95' is also visible. On the right, the code editor shows a Java solution for finding the left view of a binary tree. The code defines a `Node` class and a `Solution` class with a recursive method `leftsideview`.

```

119 Node left, right;
120
121 Node(int item)
122 {
123     data = item;
124     left = right = null;
125 }
126
127 class Solution {
128     // Function to return list containing elements of left view of binary
129     ArrayList<Integer> leftView(Node root) {
130         ArrayList<Integer> res=new ArrayList<>();
131         leftsideview(root,res,0);
132         return res;
133     }
134     static void leftsideview(Node root, ArrayList<Integer> res, int level
135     {
136         if(root==null){
137             return;
138         }
139         if(res.size()==level){
140             res.add(root.data);
141         }
142         leftsideview(root.left,res,level+1);
143         leftsideview(root.right,res,level+1);
144     }
145 }

```

**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:  $O(\log N)$**

**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;
}
}

```

The screenshot displays a coding platform interface. On the left, the 'Output Window' shows 'Compilation Results' for a custom input by 'Y.O.G.I. (AI Bot)'. It confirms the 'Problem Solved Successfully' with 1111 test cases passed out of 1111, 1 attempt out of 1, 100% accuracy, 4 points scored out of 4, and a time taken of 0.77. Below this, it suggests solving the next problem, 'Bottom View of Binary Tree'. On the right, the Java code for the solution is shown, including a 'Pair' class and a 'Solution' class with a 'topView' method that uses a map and a queue to traverse the tree and collect node values by line.

**Output Window**

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

Problem Solved Successfully

Suggest Feedback

Test Cases Passed <b>1111 / 1111</b>	Attempts : Correct / Total <b>1 / 1</b> Accuracy : 100%
Points Scored <b>4 / 4</b> Your Total Score: 99	Time Taken <b>0.77</b>

**Solve Next**

Bottom View of Binary Tree Maximum difference between node and its ancestor

Extreme nodes in alternate order

Kick start your career with GfG 160!

```

25 - class Pair{
26     Node node;
27     int line;
28     Pair(Node node,int line){
29         this.node=node;
30         this.line=line;
31     }
32 }
33
34
35 - class Solution {
36     static ArrayList<Integer> topView(Node root) {
37         ArrayList<Integer> arr=new ArrayList<>();
38         if(root==null){
39             return arr;
40         }
41         Map<Integer,Integer> map=new TreeMap<>();
42         Queue<Pair> q=new LinkedList<>();
43         q.add(new Pair(root,0));
44         while(!q.isEmpty()){
45             Pair pair=q.poll();
46             Node nd=pair.node;
47             int lin=pair.line;
48             if(!map.containsKey(lin)){
49                 map.put(lin,nd.data);
50             }
51             if(nd.left!=null){
52                 q.add(new Pair(nd.left,lin-1));
53             }
54             if(nd.right!=null){
55                 q.add(new Pair(nd.right,lin+1));
56             }
57         }
58         for(int value:map.values()){
59             arr.add(value);
60         }
61         return arr;
62     }
63 }

```

**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;
}
}

```

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: Correct / Total' as 1 / 1, 'Accuracy: 100%', 'Points Scored' as 4 / 4, and 'Time Taken' as 1.16. Below this, 'Solve Next' suggests problems like 'Connect Nodes of Levels', 'Maximum difference between node and its ancestor', and 'Vertical Tree Traversal'. On the right, the code editor shows the Java solution, which includes a 'Pair' class, a 'Solution' class with a 'bottomView' method, and the implementation of the BFS algorithm using a queue and a map to store node data by line.

```

120 - class Pair{
121     Node node;
122     int line;
123     Pair(Node node,int line){
124         this.node=node;
125         this.line=line;
126     }
127 }
128
129 class Solution
130 {
131     //Function to return a list containing the bottom view of the given tree.
132     public ArrayList<Integer> bottomView(Node root)
133     {
134         ArrayList<Integer> arr=new ArrayList<>();
135         if(root==null){
136             return arr;
137         }
138         Map<Integer,Integer> map=new TreeMap<>();
139         Queue<Pair> q=new LinkedList<>();
140         q.add(new Pair(root,0));
141         while(!q.isEmpty()){
142             Pair pair=q.poll();
143             Node node=pair.node;
144             int line=pair.line;
145             map.put(line,node.data);
146             if(node.left!=null){
147                 q.add(new Pair(node.left,line-1));
148             }
149             if(node.right!=null){
150                 q.add(new Pair(node.right,line+1));
151             }
152         }
153         for(int value:map.values()){
154             arr.add(value);
155         }
156         return arr;
157     }
158 }

```

**Time Complexity:  $O(N)$**

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

