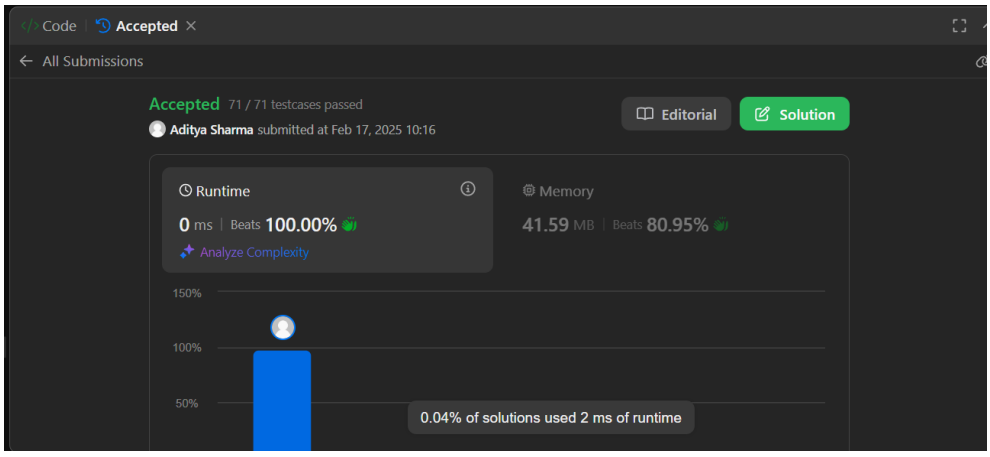
AP: ASSIGNMENT 3

NAME: Aditya Sharma	UID: 22BCS10116
SECTION: 22BSC_FL_IOT-601	GROUP: A

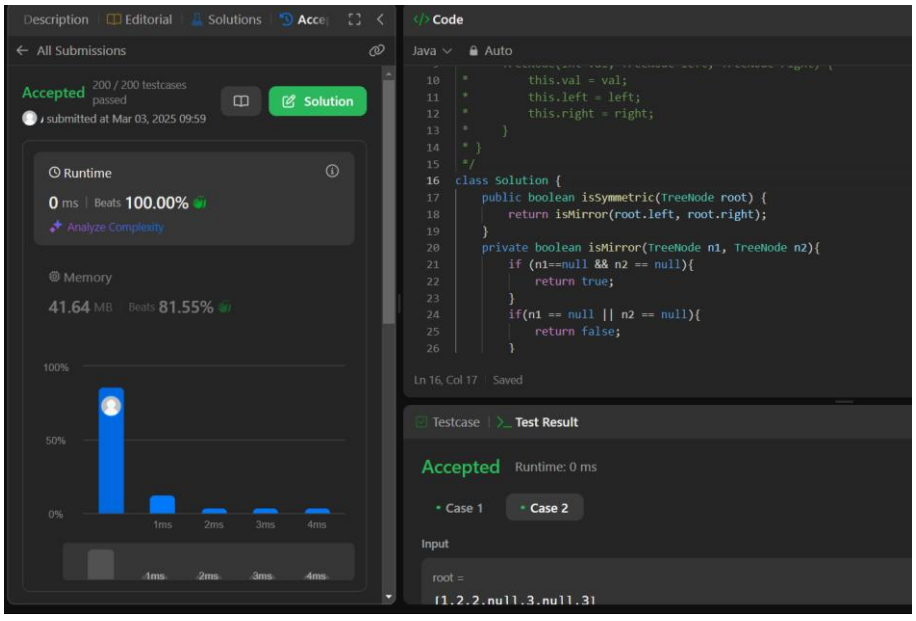
94.Binary Tree Inorder Traversal

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        traverse(root, res);
        return res;
    }
    void traverse(TreeNode root, List<Integer> res){
        if(root == null) return;
        traverse(root.left, res);
        res.add(root.val);
        traverse(root.right, res);
    }
}
```



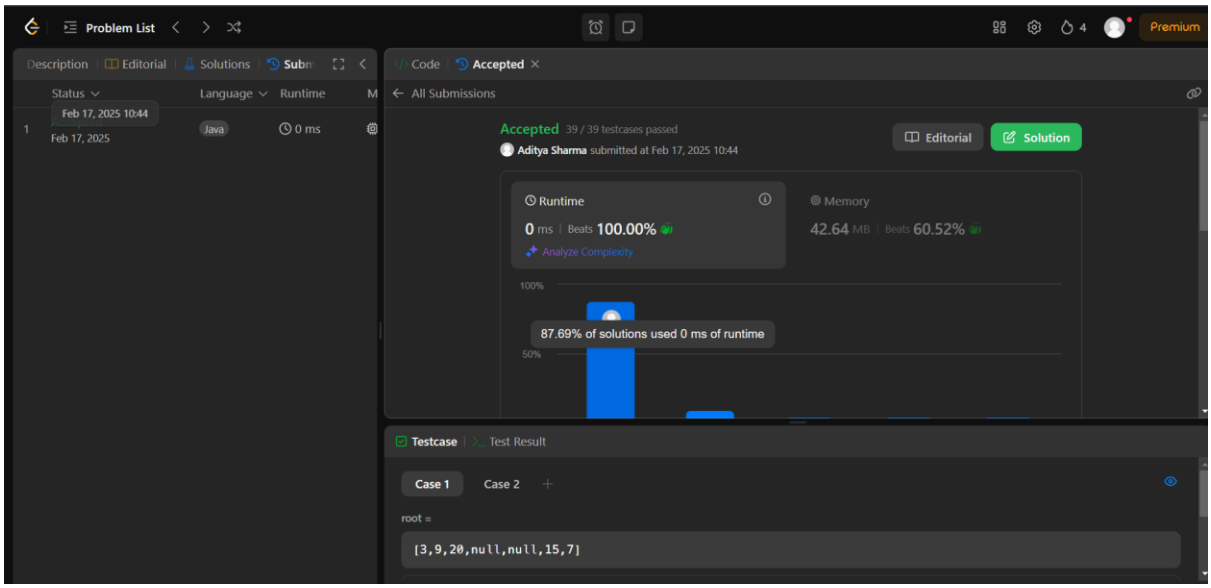
101.Symmetric Tree

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isSymmetric(TreeNode root) {
        return isMirror(root.left, root.right);
    }
    private boolean isMirror(TreeNode n1, TreeNode n2){
        if (n1==null && n2 == null){
            return true;
        }
        if(n1 == null || n2 == null){
            return false;
        }
        return n1.val == n2.val && isMirror(n1.left, n2.right) &&
isMirror(n1.right, n2.left);
    }
}
```



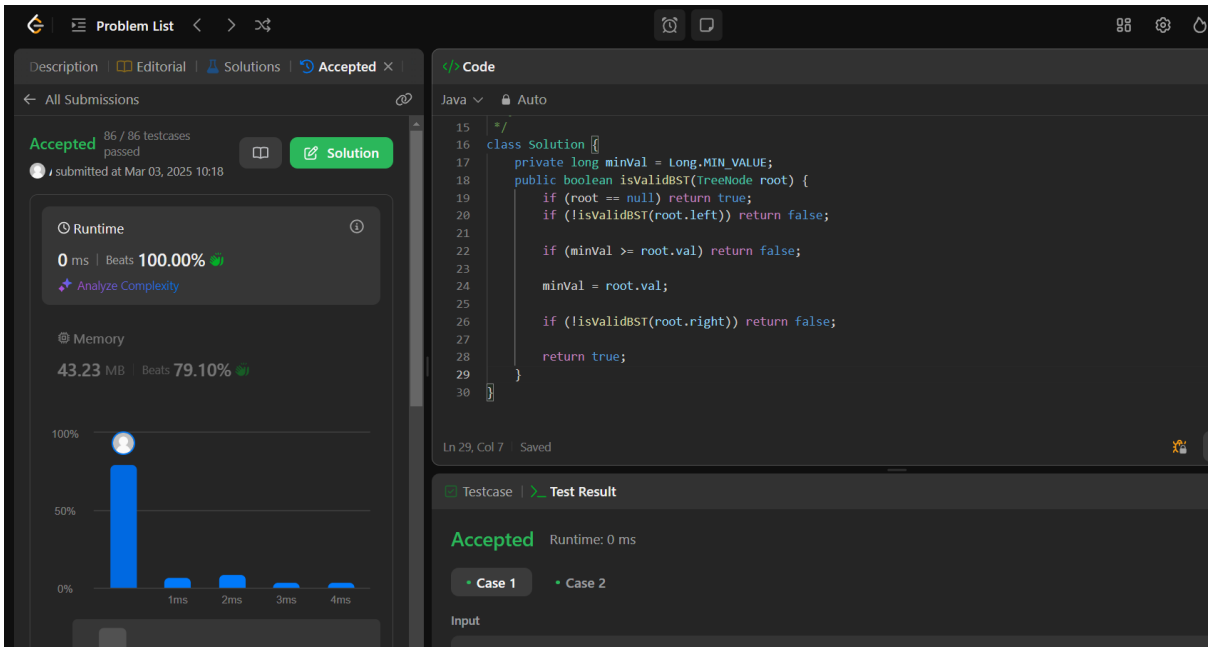
104.[Maximum Depth of Binary Tree]

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public int maxDepth(TreeNode root) {
        return traverse(root);
    }
    int traverse(TreeNode root){
        if(root == null) return 0;
        int r = traverse(root.left);
        int l = traverse(root.right);
       return Math.max(r, l)+1;
    }
}
```



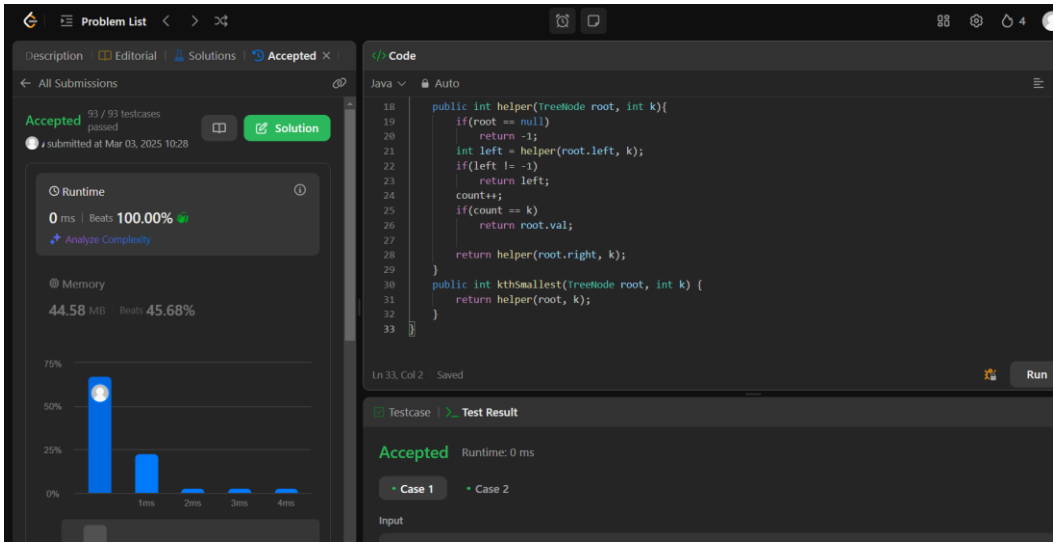
98. Validate Binary Search Tree

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    private long minVal = Long.MIN_VALUE;
    public boolean isValidBST(TreeNode root) {
        if (root == null) return true;
        if (!isValidBST(root.left)) return false;

        if (minVal >= root.val) return false;

        minVal = root.val;

        if (!isValidBST(root.right)) return false;

        return true;
    }
}
```



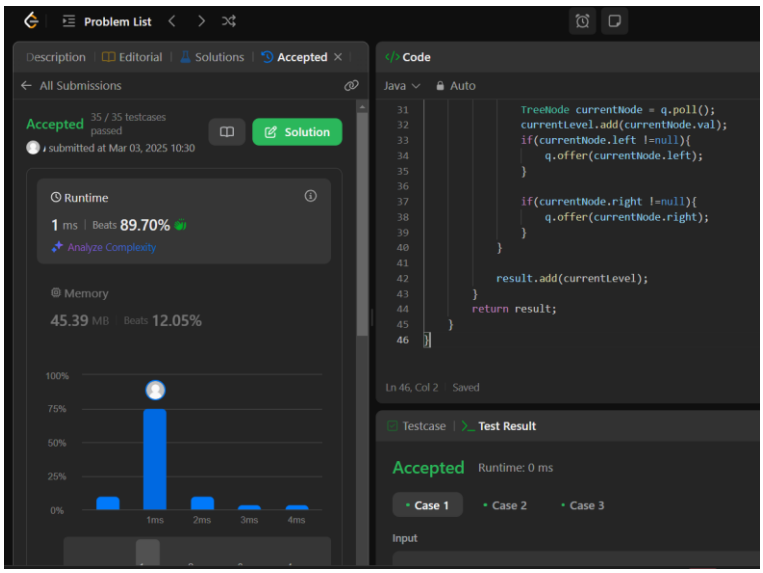
230. [Kth Smallest Element in a BST]

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    int count = 0;
    public int helper(TreeNode root, int k){
        if(root == null)
            return -1;
        int left = helper(root.left, k);
        if(left != -1)
            return left;
        count++;
        if(count == k)
            return root.val;

        return helper(root.right, k);
    }
    public int kthSmallest(TreeNode root, int k) {
        return helper(root, k);
    }
}
```



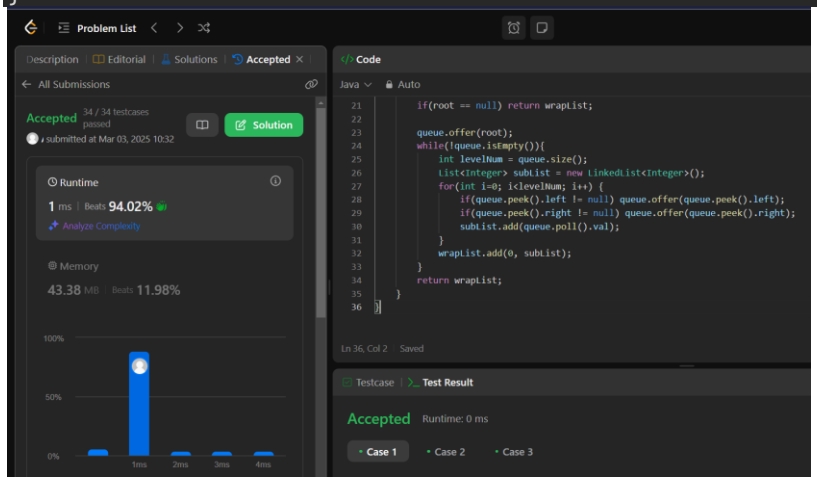
102. [Binary Tree Level Order Traversal]

```java
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {

        List<List<Integer>> result=new ArrayList<>();
        if(root==null)
        return result;

        Queue<TreeNode> q=new LinkedList<>();
        q.offer(root);
        while(!q.isEmpty()){
            int levelSize=q.size();
            List<Integer> currentLevel = new ArrayList<>();

            for(int i=0;i<levelSize;i++){

                TreeNode currentNode = q.poll();
                currentLevel.add(currentNode.val);
                if(currentNode.left !=null){
                    q.offer(currentNode.left);
                }

                if(currentNode.right !=null){
                    q.offer(currentNode.right);
                }
            }

            result.add(currentLevel);
        }
        return result;
    }
}
```



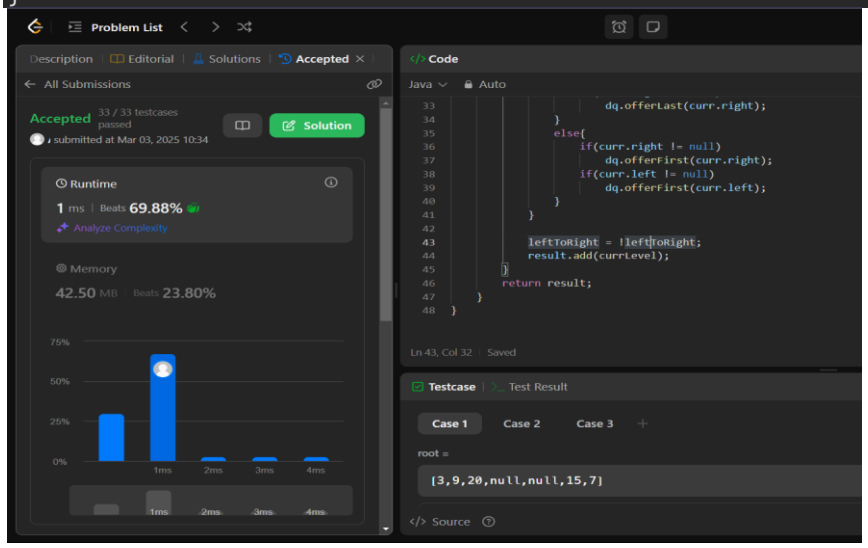
107.[Binary Tree Level Order Traversal II]

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        List<List<Integer>> wrapList = new LinkedList<List<Integer>>();

        if(root == null) return wrapList;

        queue.offer(root);
        while(!queue.isEmpty()){
            int levelNum = queue.size();
            List<Integer> subList = new LinkedList<Integer>();
            for(int i=0; i<levelNum; i++) {
                if(queue.peek().left != null) queue.offer(queue.peek().left);
                if(queue.peek().right != null) queue.offer(queue.peek().right);
                subList.add(queue.poll().val);
            }
            wrapList.add(0, subList);
        }
        return wrapList;
    }
}
```



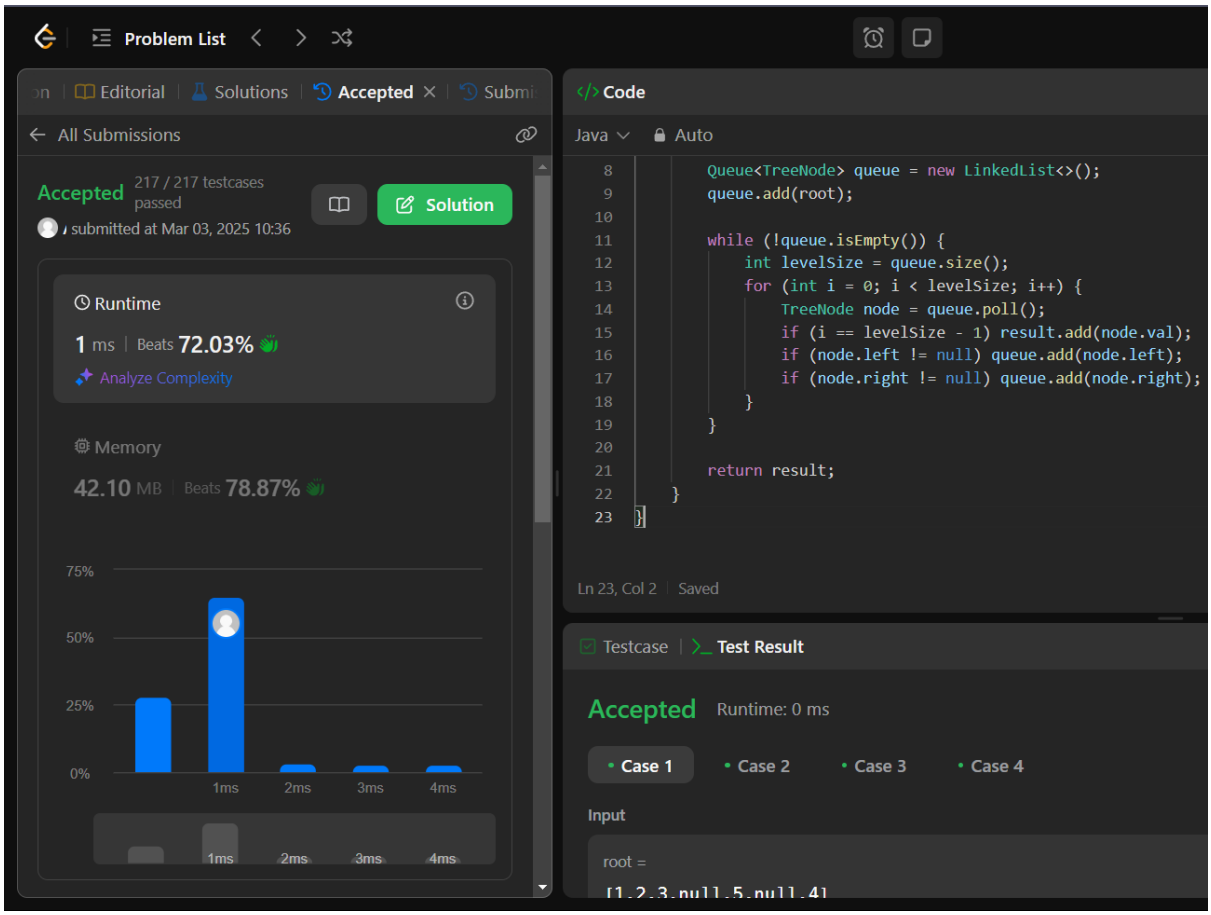
103.Binary Tree Zigzag Level Order Traversal

```java
class Solution {

    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        if(root == null)return new ArrayList<>();
        ArrayDeque<TreeNode> dq = new ArrayDeque<>();
        dq.offer(root);
        List<List<Integer>> result = new ArrayList<>();
        boolean leftToRight = true;

        while(!dq.isEmpty()){
            List<Integer> currLevel = new ArrayList<>();
            for(int i = dq.size(); i > 0; i--){
                TreeNode curr = (leftToRight)?dq.pollFirst():dq.pollLast();
                currLevel.add(curr.val);
                if(leftToRight){
                    if(curr.left != null)
                        dq.offerLast(curr.left);
                    if(curr.right != null)
                        dq.offerLast(curr.right);
                }
                else{
                    if(curr.right != null)
                        dq.offerFirst(curr.right);
                    if(curr.left != null)
                        dq.offerFirst(curr.left);
                }
            }

            leftToRight = !leftToRight;
            result.add(currLevel);
        }
        return result;
    }
}
```



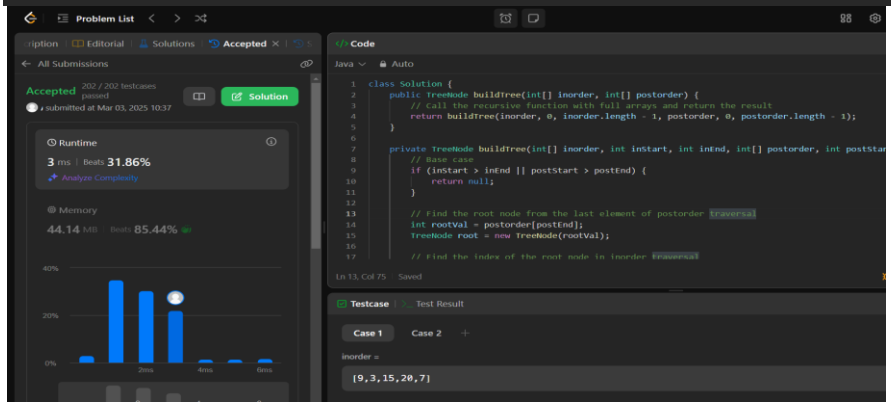
199.Binary Tree Right Side View

```java
import java.util.*;
// takesoumen collection
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) return result;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            for (int i = 0; i < levelSize; i++) {
                TreeNode node = queue.poll();
                if (i == levelSize - 1) result.add(node.val);
                if (node.left != null) queue.add(node.left);
                if (node.right != null) queue.add(node.right);
            }
        }

        return result;
    }
}
```



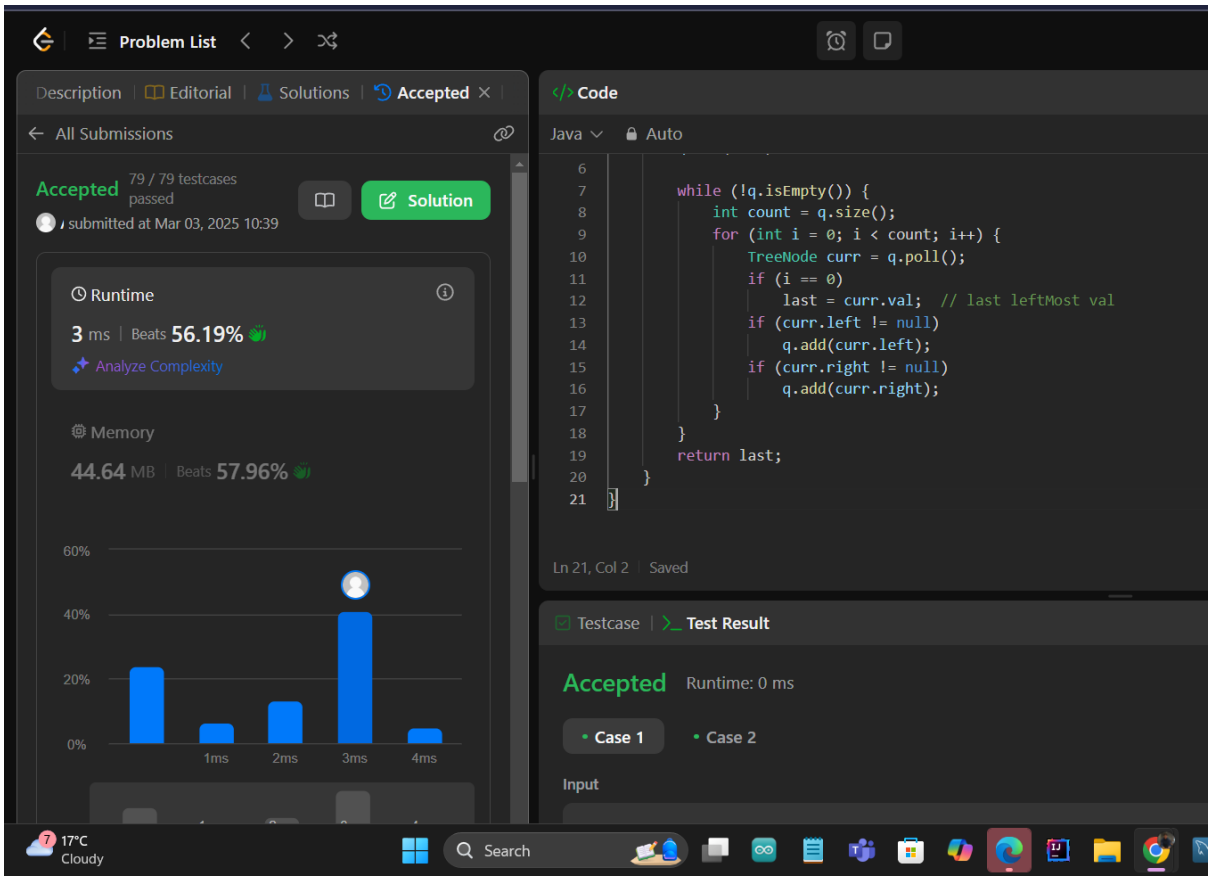
106.Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution {

    public TreeNode buildTree(int[] inorder, int[] postorder) {
        // Call the recursive function with full arrays and return the result
        return buildTree(inorder, 0, inorder.length - 1, postorder, 0,
postorder.length - 1);
    }

    private TreeNode buildTree(int[] inorder, int inStart, int inEnd, int[]
postorder, int postStart, int postEnd) {
        // Base case
        if (inStart > inEnd || postStart > postEnd) {
            return null;
        }

        // Find the root node from the last element of postorder traversal
        int rootVal = postorder[postEnd];
        TreeNode root = new TreeNode(rootVal);

        // Find the index of the root node in inorder traversal
        int rootIndex = 0;
        for (int i = inStart; i <= inEnd; i++) {
            if (inorder[i] == rootVal) {
                rootIndex = i;
                break;
            }
        }

        // Recursively build the left and right subtrees
        int leftSize = rootIndex - inStart;
        int rightSize = inEnd - rootIndex;
        root.left = buildTree(inorder, inStart, rootIndex - 1, postorder,
postStart, postStart + leftSize - 1);
        root.right = buildTree(inorder, rootIndex + 1, inEnd, postorder, postEnd -
rightSize, postEnd - 1);

        return root;
    }
}
```



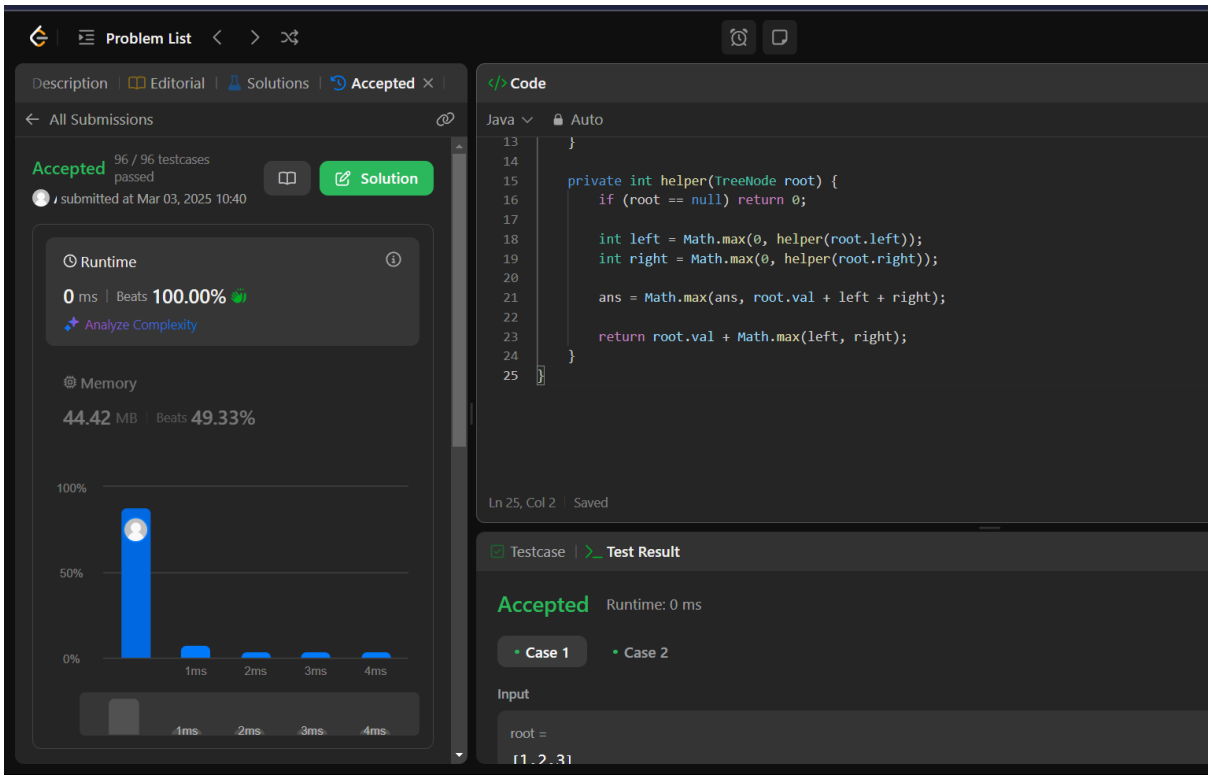
513.[Find Bottom Left Tree Value]

```java
public class Solution {
    public int findBottomLeftValue(TreeNode root) {
        int last = 0;
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);

        while (!q.isEmpty()) {
            int count = q.size();
            for (int i = 0; i < count; i++) {
                TreeNode curr = q.poll();
                if (i == 0)
                    last = curr.val;  // last leftMost val
                if (curr.left != null)
                    q.add(curr.left);
                if (curr.right != null)
                    q.add(curr.right);
            }
        }
        return last;
    }
}
```



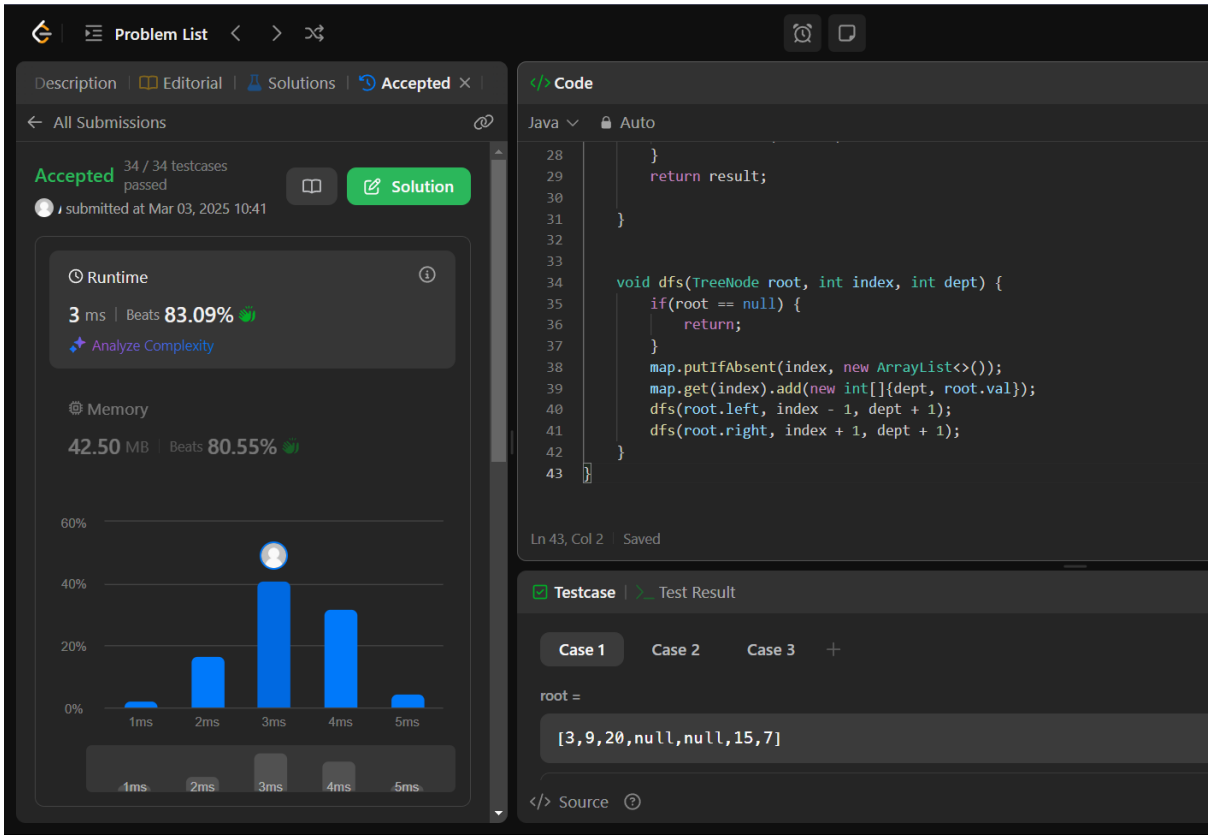
124. [Binary Tree Maximum Path Sum]

```java
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int val) { this.val = val; }
}

class Solution {
    private int ans = Integer.MIN_VALUE;

    public int maxPathSum(TreeNode root) {
        helper(root);
        return ans;
    }

    private int helper(TreeNode root) {
        if (root == null) return 0;

        int left = Math.max(0, helper(root.left));
        int right = Math.max(0, helper(root.right));

        ans = Math.max(ans, root.val + left + right);

        return root.val + Math.max(left, right);
    }
}
```



987.[Vertical Order Traversal of a Binary Tree]

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    Map<Integer, ArrayList<int[]>> map = new TreeMap<>();
    public List<List<Integer>> verticalTraversal(TreeNode root) {
        dfs(root, 0, 0);
        List<List<Integer>> result = new ArrayList<>();
        for(ArrayList<int[]> list: map.values()) {
           Collections.sort(list, (a, b) -> a[0] == b[0] ? Integer.compare(a[1], 
b[1]) : Integer.compare(a[0], b[0]));
            ArrayList<Integer> current = new ArrayList<>();
            for(int[] num :  list) {
                current.add(num[1]);
            }
            result.add(current);
        }
        return result;

    }


    void dfs(TreeNode root, int index, int dept) {
        if(root == null) {
            return;
        }
        map.putIfAbsent(index, new ArrayList<>());
        map.get(index).add(new int[]{dept, root.val});
        dfs(root.left, index - 1, dept + 1);
        dfs(root.right, index + 1, dept + 1);
    }
}
```


Problem List
Description
Editorial
Solutions
Accepted
All Submissions
Accepted
34 / 34 testcases passed
submitted at Mar 03, 2025 10:41
Solution
Runtime
3 ms | Beats 83.09%
Analyze Complexity
Memory
42.50 MB | Beats 80.55%
60%
40%
20%
0%
1ms
2ms
3ms
4ms
5ms
Code
Java
Auto
28 }
29 return result;
30
31 }
32
33
34 void dfs(TreeNode root, int index, int dept) {
35 if(root == null) {
36 return;
37 }
38 map.putIfAbsent(index, new ArrayList<>());
39 map.get(index).add(new int[]{dept, root.val});
40 dfs(root.left, index - 1, dept + 1);
41 dfs(root.right, index + 1, dept + 1);
42 }
43 }
Ln 43, Col 2
Saved
Testcase
Test Result
Case 1
Case 2
Case 3
root =
[3,9,20,null,null,15,7]
Source