Experiment 3

Student Name: Khushmn Sangha
Branch: BE-CSE
Semester: 6th
Subject Name: Advanced Programming - II

UID: 22BCS14585
Section/Group: 602/A
Date of Performance: 14-02-25
Subject Code: 22CSP-351

Aim: To solve the following problems on Leetcode, with the goal of optimizing solutions in terms of time complexity and space efficiency:

(I) 94.Binary Tree Inorder Traversal
(II) 101.Symmetric Tree
(III) 104.Maximum Depth of Binary Tree
(IV) 98.Validate Binary Search Tree
(V) 230.Kth Smallest Element in a BST
(VI) 102. Binary Tree Level Order Traversal
(VII) 107.Binary Tree Level Order Traversal II
(VII) 103.Binary Tree Zigzag Level Order Traversal
(VIII) 199.Binary Tree Right Side View
(IX) 106.Construct Binary Tree from Inorder and Postorder Traversal
(X) 513.Find Bottom Left Tree Value
(XI) 124. Binary Tree Maximum Path Sum
(XII) 987.Vertical Order Traversal of a Binary Tree

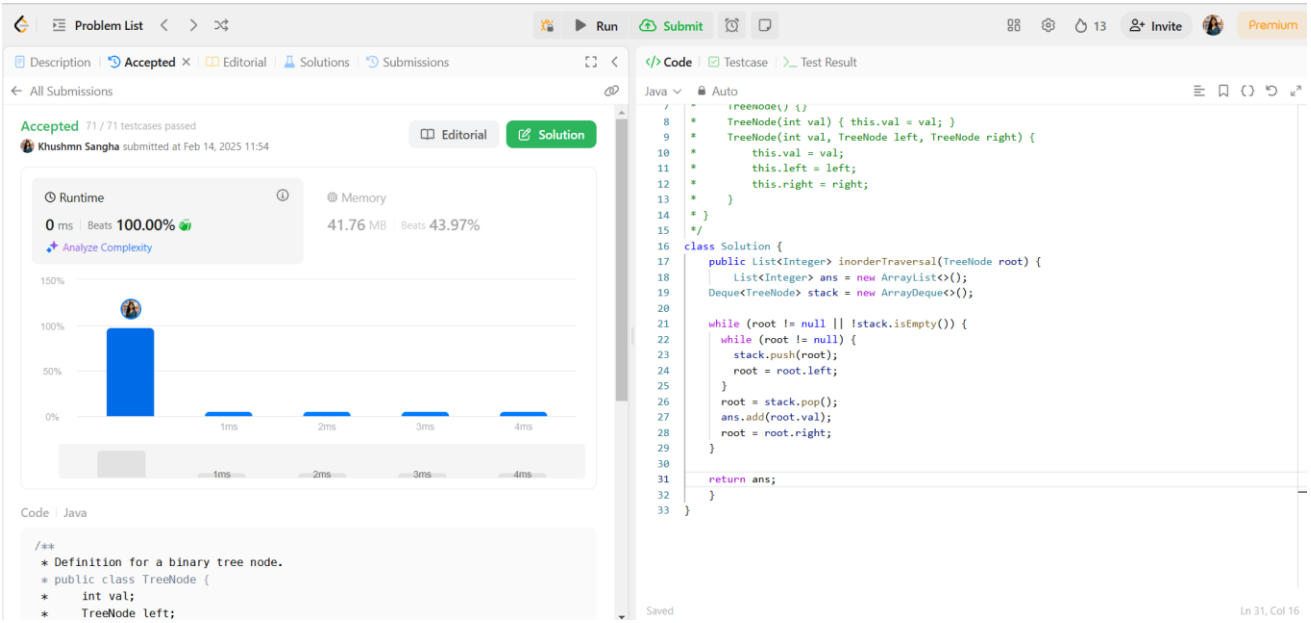
1. **Objective:** To efficiently solve a set of algorithmic problems on Leetcode, optimizing for time and space complexity, by implementing advanced techniques such as bit manipulation, binary search, sorting algorithms, and dynamic programming, with the goal of enhancing problem-solving skills and achieving optimal solutions for real-world applications.

2. Implementation/Code:

(VIII) ***Problem No.*** 94.Binary Tree Inorder Traversal

Code:
```
class Solution {
   public List<Integer>
inorderTraversal(TreeNode root) {
```

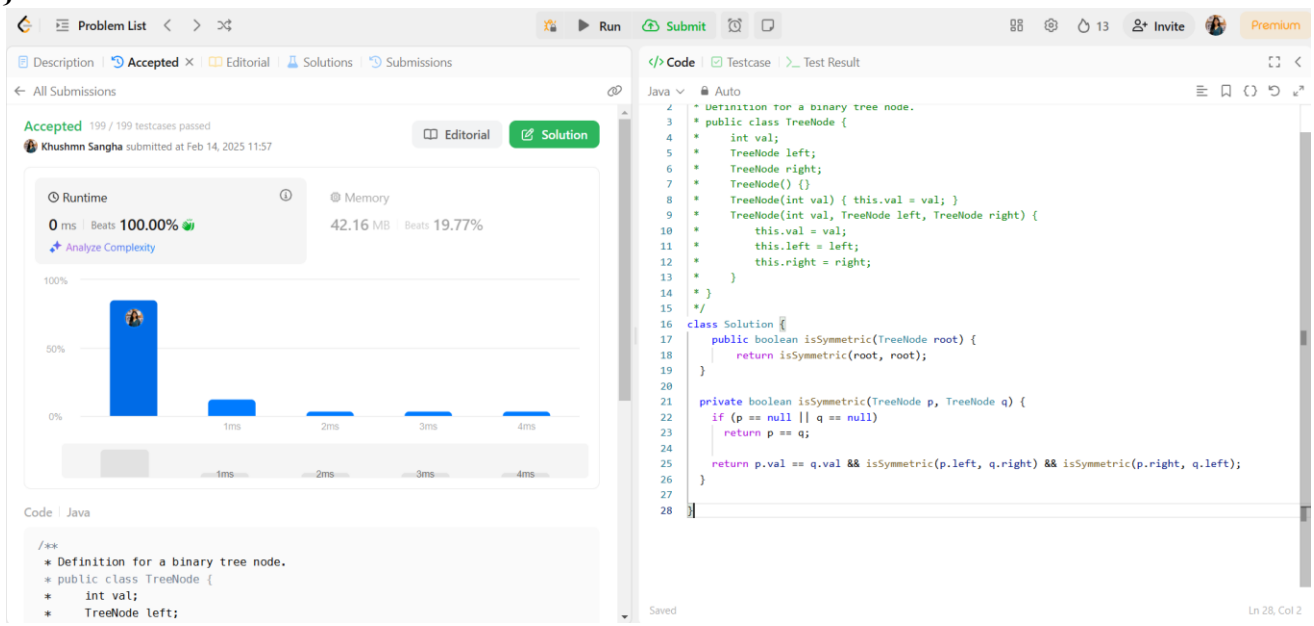
```
        List<Integer> ans = new
ArrayList<>();
    Deque<TreeNode> stack = new
ArrayDeque<>();

    while (root != null ||
!stack.isEmpty()) {
      while (root != null) {
        stack.push(root);
        root = root.left;
      }
      root = stack.pop();
      ans.add(root.val);
      root = root.right;
    }

    return ans;
    }
}
```



(IX) ***Problem No.*** 101.Symmetric Tree

Code: class Solution {

```
    public boolean isSymmetric(TreeNode root) {
        return isSymmetric(root, root);
    }

    private boolean isSymmetric(TreeNode p, TreeNode q) {
      if (p == null || q == null)
        return p == q;

      return p.val == q.val && isSymmetric(p.left, q.right) && isSymmetric(p.right, q.left);
    }

}
```

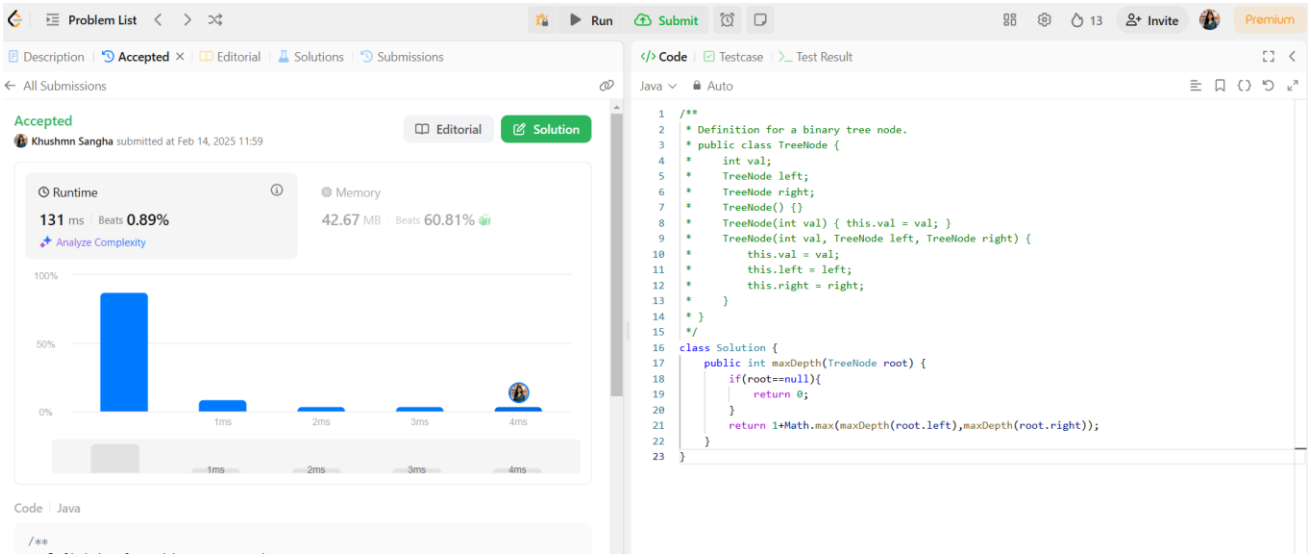


(X) ***Problem No.*** 104.Maximum Depth of Binary Tree

Code:

```
class Solution {
    public int maxDepth(TreeNode root) {
        if(root==null){
            return 0;
        }
        return 1+Math.max(maxDepth(root.left),maxDepth(root.right));
```

```
    }
}
```

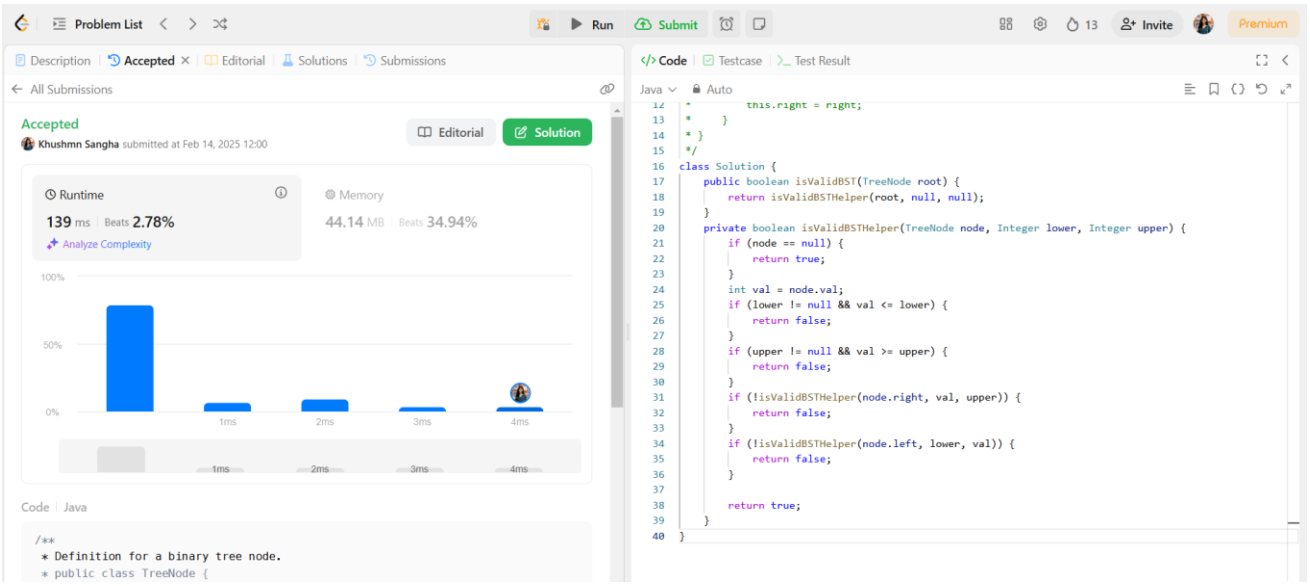


(XI) ***Problem No.*** 98.Validate Binary Search Tree

Code:

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return isValidBSTHelper(root, null, null);
    }
    private boolean isValidBSTHelper(TreeNode node, Integer lower, Integer upper) {
        if (node == null) {
            return true;
        }
        int val = node.val;
        if (lower != null && val <= lower) {
            return false;
        }
        if (upper != null && val >= upper) {
            return false;
        }
        if (!isValidBSTHelper(node.right, val, upper)) {
            return false;
        }
```

```
            if (!isValidBSTHelper(node.left, lower, val)) {
                return false;
            }

            return true;
        }
    }
```

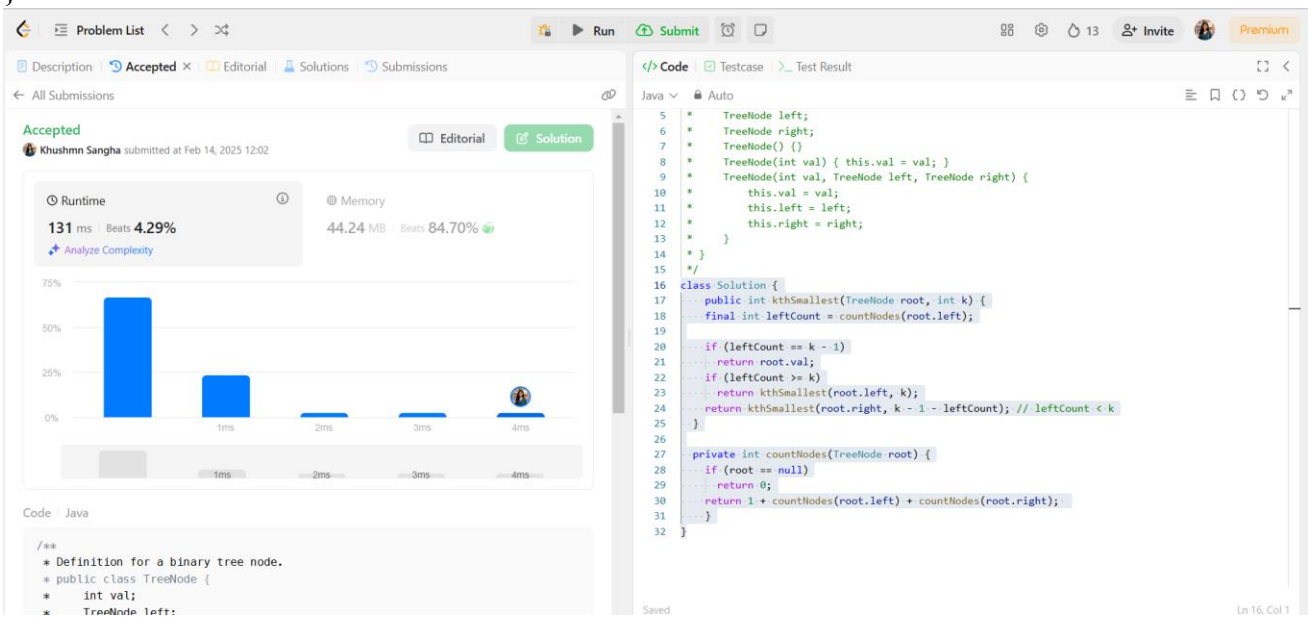


(XII) ***Problem No.*** 230.Kth Smallest Element in a BST

Code:

```
class Solution {
  public int kthSmallest(TreeNode root, int k) {
  final int leftCount = countNodes(root.left);

  if (leftCount == k - 1)
    return root.val;
  if (leftCount >= k)
    return kthSmallest(root.left, k);
  return kthSmallest(root.right, k - 1 - leftCount); // leftCount < k
}

private int countNodes(TreeNode root) {
```

```
    if (root == null)
      return 0;
    return 1 + countNodes(root.left) + countNodes(root.right);
    }
}
```

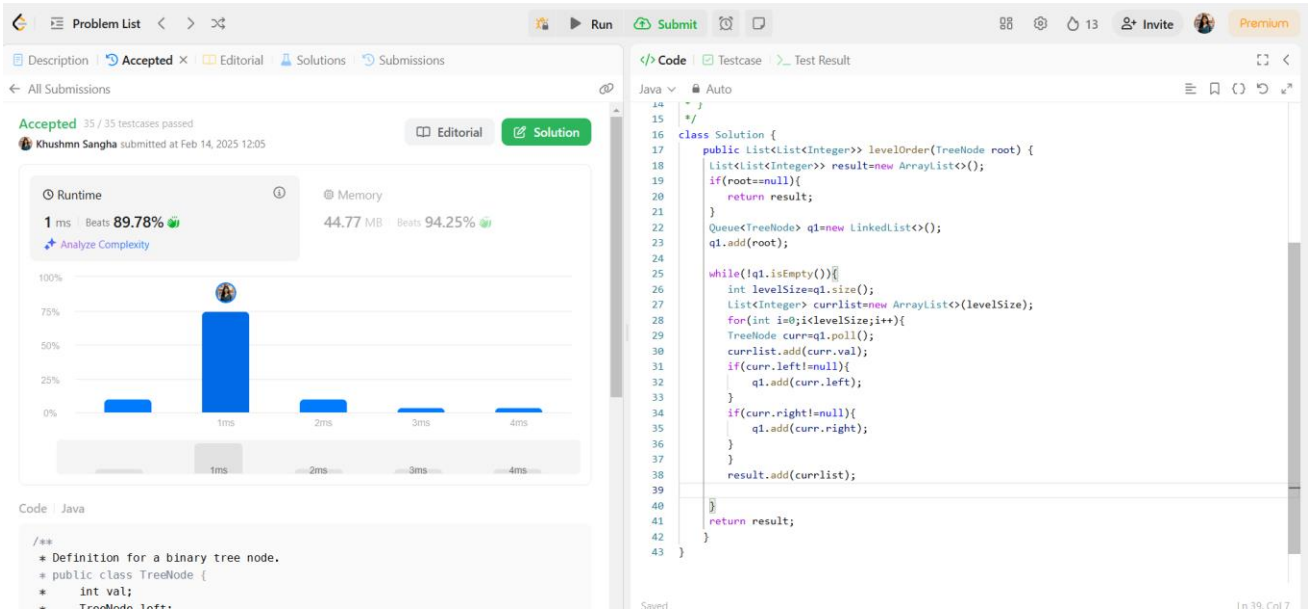


(XIII) ***Problem No.:*** 102. Binary Tree Level Order Traversal

Code:

```
class Solution {
  public List<List<Integer>> levelOrder(TreeNode root) {
   List<List<Integer>> result=new ArrayList<>();
   if(root==null){
     return result;
   }
   Queue<TreeNode> q1=new LinkedList<>();
   q1.add(root);
```

```
        while(!q1.isEmpty()){
          int levelSize=q1.size();
          List<Integer> currlist=new ArrayList<>(levelSize);
          for(int i=0;i<levelSize;i++){
          TreeNode curr=q1.poll();
          currlist.add(curr.val);
          if(curr.left!=null){
              q1.add(curr.left);
          }
          if(curr.right!=null){
              q1.add(curr.right);
          }
          }
          result.add(currlist);

        }
        return result;
       }
}
```



(XIV) ***Problem No.*** 107.Binary Tree Level Order Traversal II

Code: class Solution {

public List<List<Integer>>

levelOrderBottom(TreeNode root) {

List<List<Integer>> result=new

ArrayList<>();

List<List<Integer>> result1=new

ArrayList<>();

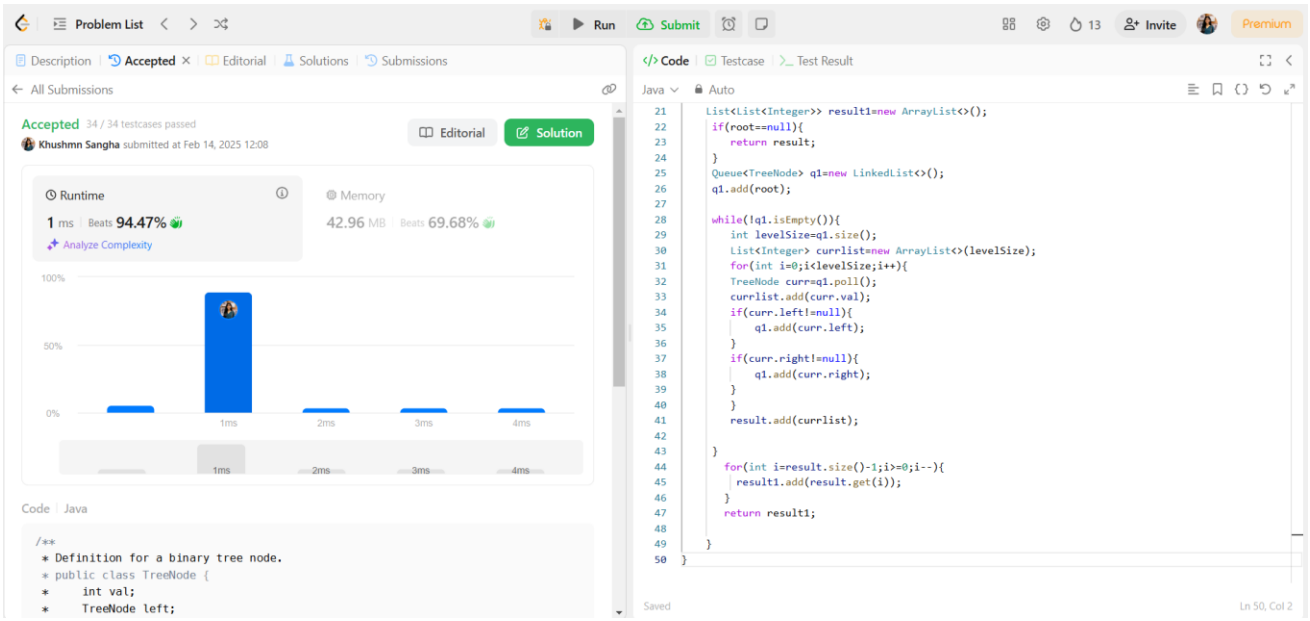
if(root==null){

return result;

}

```
    Queue<TreeNode> q1=new
LinkedList<>();
    q1.add(root);

    while(!q1.isEmpty()){
      int levelSize=q1.size();
      List<Integer> currlist=new
ArrayList<>(levelSize);
      for(int i=0;i<levelSize;i++){
      TreeNode curr=q1.poll();
      currlist.add(curr.val);
      if(curr.left!=null){
         q1.add(curr.left);
      }
      if(curr.right!=null){
         q1.add(curr.right);
      }
      }
      result.add(currlist);

    }
     for(int i=result.size()-1;i>=0;i--){
      result1.add(result.get(i));
     }
     return result1;
```

```
    }
}
```



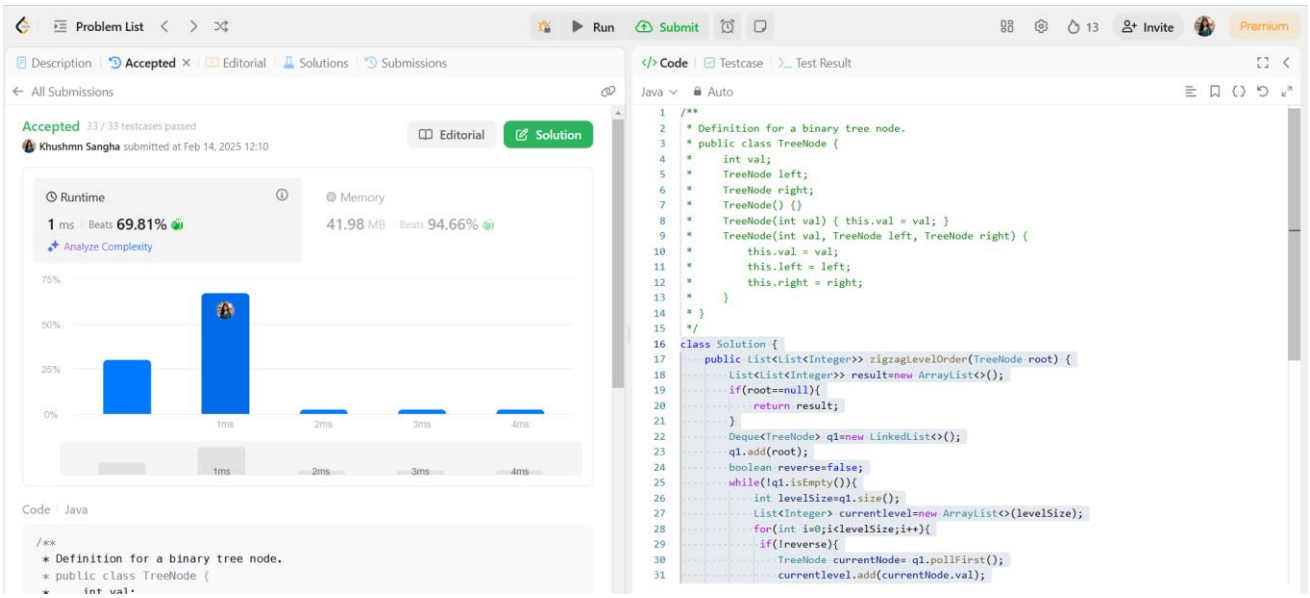
(XIII) ***Prolem No.*** 103.[Binary Tree Zigzag Level Order Traversal]

Code: class Solution {

```
    public List<List<Integer>>
zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> result=new ArrayList<>();
        if(root==null){
            return result;
        }
        Deque<TreeNode> q1=new LinkedList<>();
        q1.add(root);
        boolean reverse=false;
```

```
        while(!q1.isEmpty()){
            int levelSize=q1.size();
            List<Integer> currentlevel=new
ArrayList<>(levelSize);
            for(int i=0;i<levelSize;i++){
             if(!reverse){
                TreeNode currentNode= q1.pollFirst();
                currentlevel.add(currentNode.val);
                if(currentNode.left!=null){
                    q1.addLast(currentNode.left);
                }if(currentNode.right!=null){
                    q1.addLast(currentNode.right);
                }

             }else{
             TreeNode currentNode= q1.pollLast();
             currentlevel.add(currentNode.val);
            if(currentNode.right!=null){
                    q1.addFirst(currentNode.right);
            }if(currentNode.left!=null){
                    q1.addFirst(currentNode.left);
                }
             }

            }
```

```
        result.add(currentlevel);reverse=!reverse;
        }
        return result;
    }
}
```

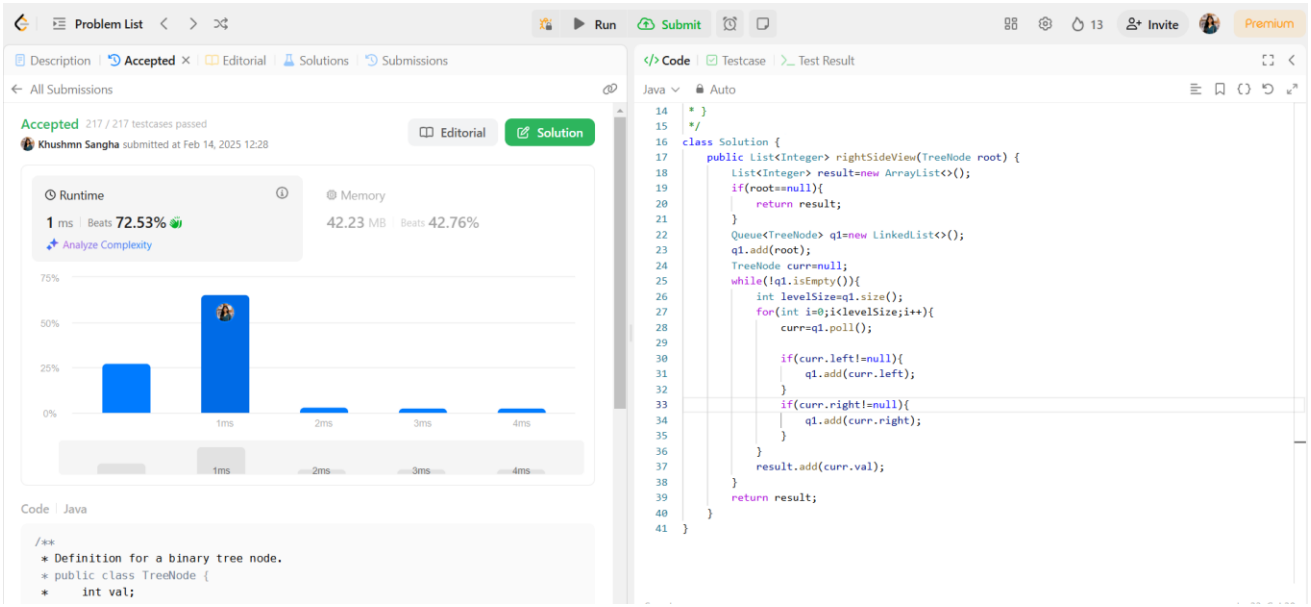


(XIV) ***Problem No.*** 199.Binary Tree Right Side View

Code:

```
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result=new ArrayList<>();
        if(root==null){
            return result;
        }
        Queue<TreeNode> q1=new LinkedList<>();
```

```
        q1.add(root);
        TreeNode curr=null;
        while(!q1.isEmpty()){
            int levelSize=q1.size();
            for(int i=0;i<levelSize;i++){
                curr=q1.poll();

                if(curr.left!=null){
                    q1.add(curr.left);
                }
                if(curr.right!=null){
                    q1.add(curr.right);
                }
            }
            result.add(curr.val);
        }
        return result;
    }
}
```



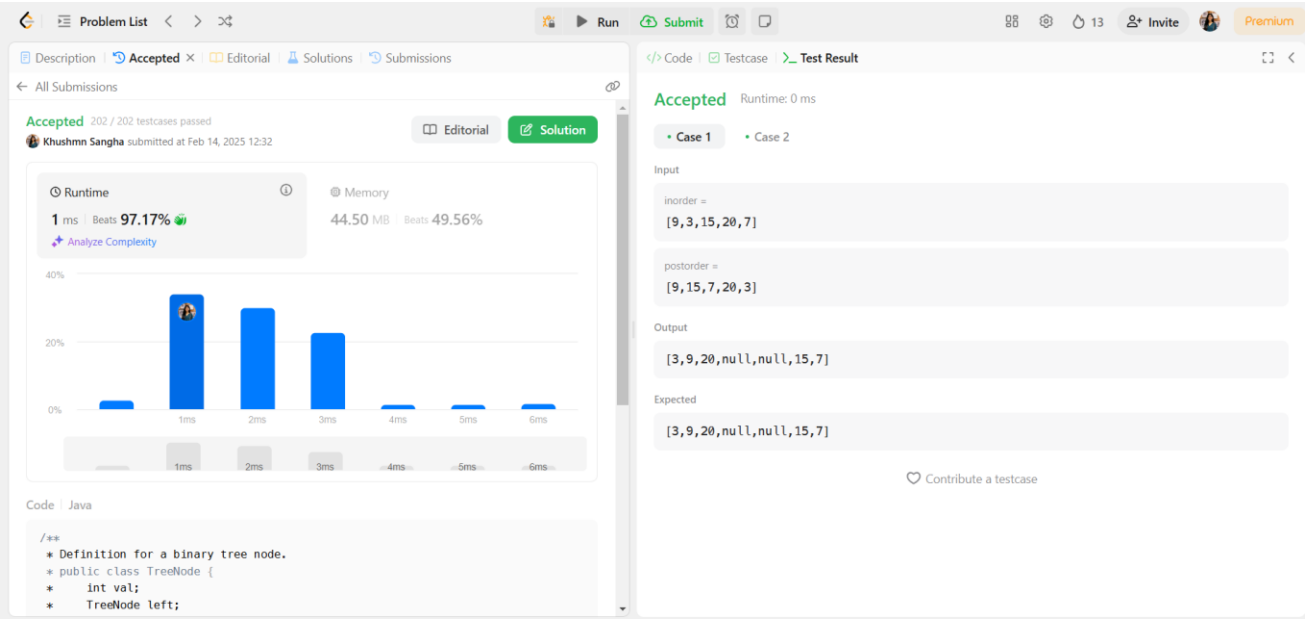
(XV) ***Problem No.*** 106.Construct Binary Tree from Inorder and Postorder Traversal

Code: class Solution {

```
    private int postIndex;

    private Map<Integer, Integer>
inorderMap;


    public TreeNode buildTree(int[]
inorder, int[] postorder) {
        postIndex = postorder.length - 1;
        inorderMap = new HashMap<>();


        for (int i = 0; i < inorder.length;
i++) {
```

```
            inorderMap.put(inorder[i], i);
        }

        return constructTree(postorder, 0,
inorder.length - 1);
    }
    private TreeNode constructTree(int[]
postorder, int left, int right) {
        if (left > right) return null;

        int rootValue =
postorder[postIndex--];
        TreeNode root = new
TreeNode(rootValue);

        int inorderIndex =
inorderMap.get(rootValue);

        root.right =
constructTree(postorder, inorderIndex
+ 1, right);
        root.left =
constructTree(postorder, left,
inorderIndex - 1);

        return root;
```

```
    }
}
```

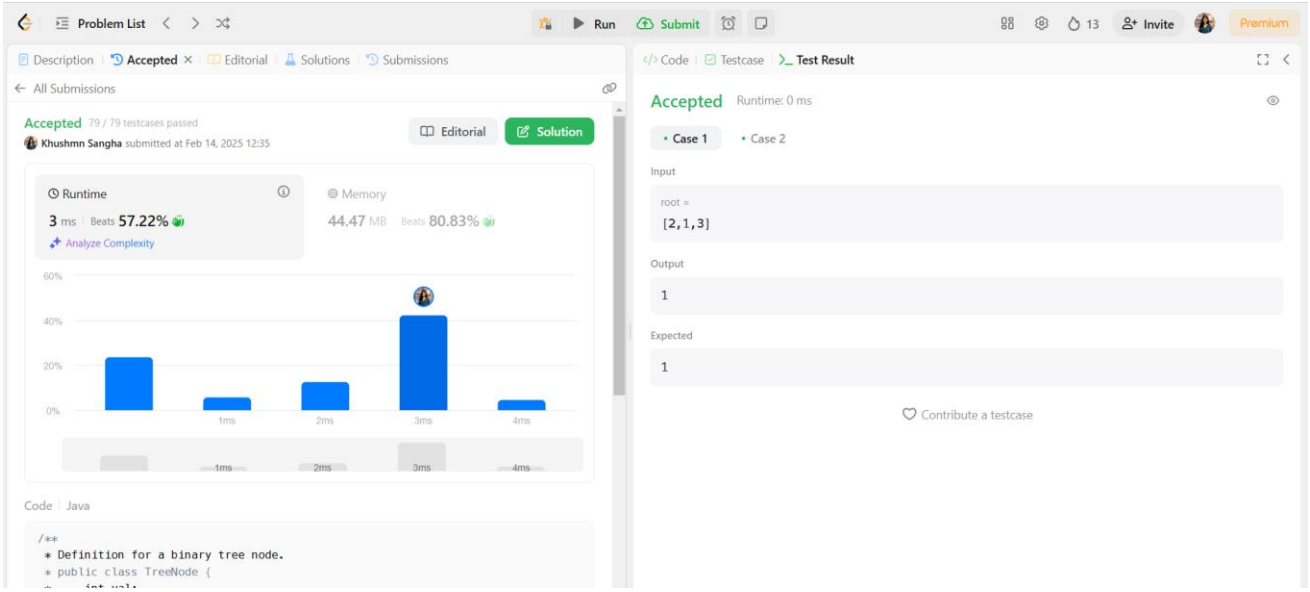


(XVI) ***Problem No.*** 513.Find Bottom Left Tree Value

Code: class Solution {

```
    public int
findBottomLeftValue(TreeNode root) {
        Queue<TreeNode> q = new
ArrayDeque<>();
        q.offer(root);
        int ans = 0;
        while (!q.isEmpty()) {
            ans = q.peek().val;
            for (int i = q.size(); i > 0; --i) {
```

```
            TreeNode node = q.poll();
            if (node.left != null) {
                q.offer(node.left);
            }
            if (node.right != null) {
                q.offer(node.right);
            }
        }
    }
    return ans;
  }
}
```

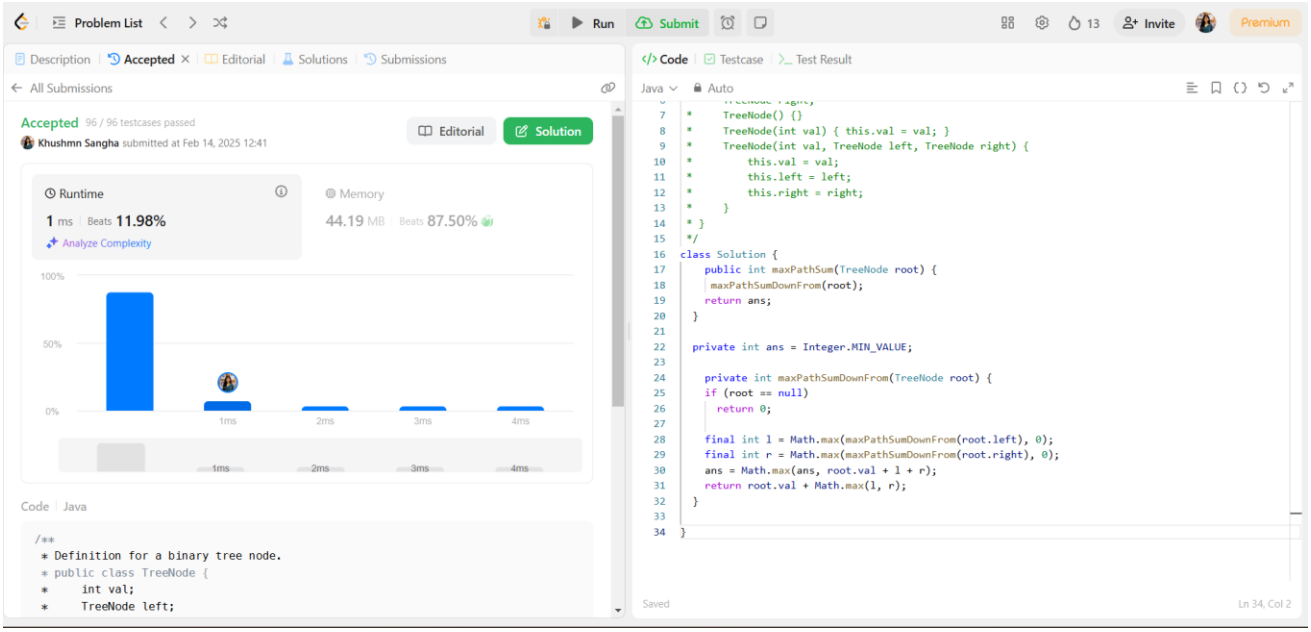


(XVII) ***Problem No.*** 124. Binary Tree Maximum Path Sum

Code:

```
class Solution {
   public int maxPathSum(TreeNode root)
{
    maxPathSumDownFrom(root);
   return ans;
  }

 private int ans = Integer.MIN_VALUE;

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

   final int l =
Math.max(maxPathSumDownFrom(root.l
eft), 0);
   final int r =
Math.max(maxPathSumDownFrom(root.r
ight), 0);
   ans = Math.max(ans, root.val + l + r);
   return root.val + Math.max(l, r);
  }
```

}



(XVIII) ***Problem No. 987:*** .Vertical Order Traversal of a Binary Tree

Code: class Solution {

public List<List<Integer>>

verticalTraversal(TreeNode root) {

// TreeMap to store nodes by

vertical level

TreeMap<Integer,

TreeMap<Integer,

PriorityQueue<Integer>>> map =

new TreeMap<>();

Queue<Tuple> queue = new

LinkedList<>();

```
        // Start BFS
        queue.offer(new Tuple(root, 0,
0));

        while (!queue.isEmpty()) {
            Tuple tuple = queue.poll();
            TreeNode node = tuple.node;
            int x = tuple.x; // Vertical
level
            int y = tuple.y; // Horizontal
level

            // Add node value to TreeMap
            map.putIfAbsent(x, new
TreeMap<>());
            map.get(x).putIfAbsent(y,
new PriorityQueue<>());
            map.get(x).get(y).offer(node.
val);

            // Traverse left and right
children
            if (node.left != null) {
                queue.offer(new
Tuple(node.left, x - 1, y + 1));
            }
```

```
            if (node.right != null) {
                queue.offer(new
Tuple(node.right, x + 1, y + 1));
            }
        }

        // Prepare result list
        List<List<Integer>> result =
new ArrayList<>();
        for (TreeMap<Integer,
PriorityQueue<Integer>> vertical :
map.values()) {
            List<Integer> verticalList =
new ArrayList<>();
            for (PriorityQueue<Integer>
nodes : vertical.values()) {
                while (!nodes.isEmpty()) {
                    verticalList.add(nodes.p
oll());
                }
            }
            result.add(verticalList);
        }

        return result;
    }
```

```
}
class Tuple {
    TreeNode node;
    int x; // Vertical level
    int y; // Horizontal level

    public Tuple(TreeNode node, int
x, int y) {
        this.node = node;
        this.x = x;
        this.y = y;
    }
}
```