Experiment 5

Student Name: Payal **UID: 22BCS15824**
Branch: CSE **Section/Group: 637-B**
Semester: 6th **Date of Performance:20/2/25**
Subject Name: Advanced Programming - 2 **Subject Code: 22CSH-351**

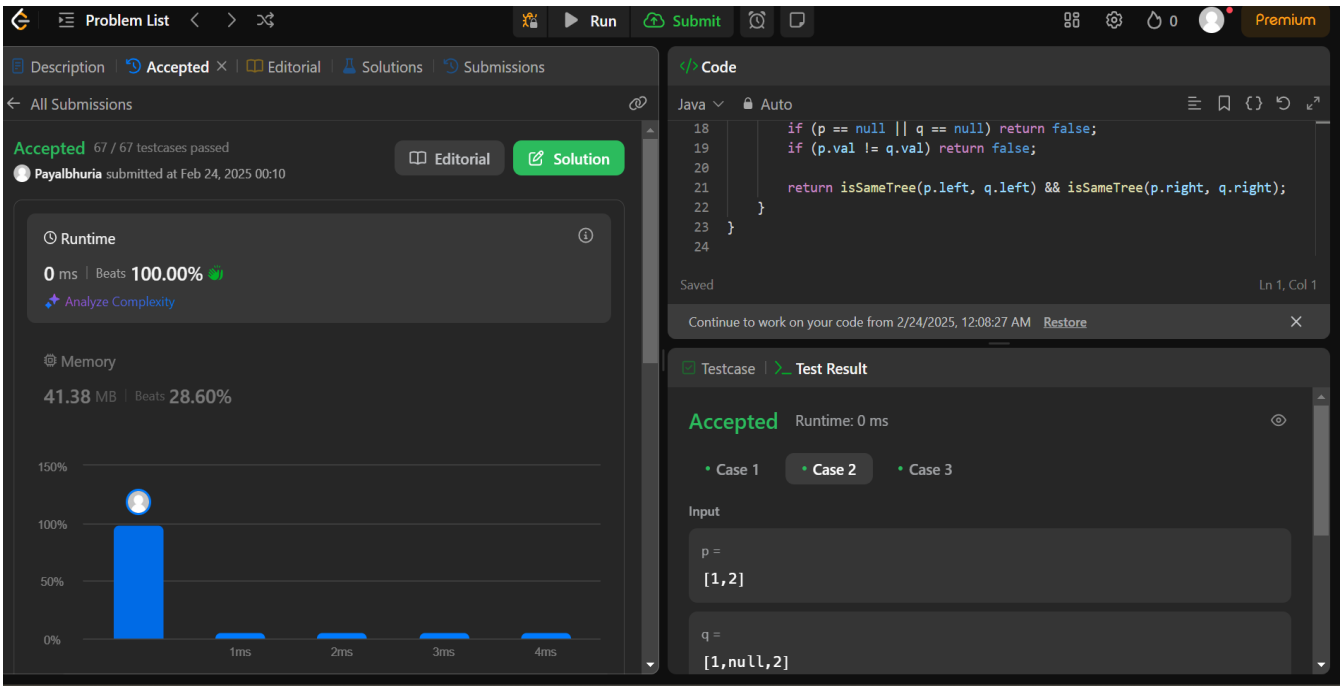
Ques 1:

Aim: Same Tree

Code:

```
// Definition for a binary tree node.
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 isSameTree(TreeNode p, TreeNode q) {
        if (p == null && q == null) return true;
        if (p == null || q == null) return false;
        if (p.val != q.val) return false;

        return isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
    }
}
```

Submission Screenshot:



Ques 2:

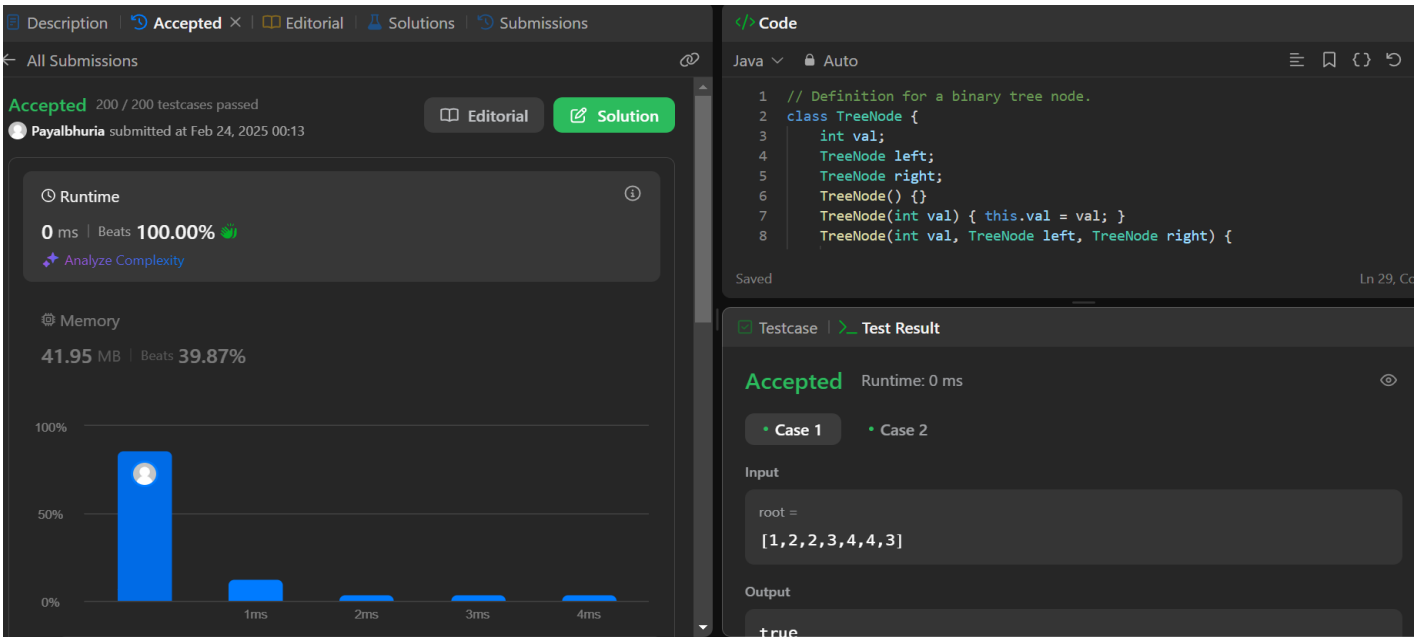
Aim: Symmetric Tree

Code:

```
// Definition for a binary tree node.
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) {
                if (root == null) return true;
                return isMirror(root.left, root.right);
            }

            private boolean isMirror(TreeNode t1, TreeNode t2) {
                if (t1 == null && t2 == null) return true;
                if (t1 == null || t2 == null) return false;
                if (t1.val != t2.val) return false;

                return isMirror(t1.left, t2.right) && isMirror(t1.right, t2.left);
            }
}
```

Submission Screenshot:



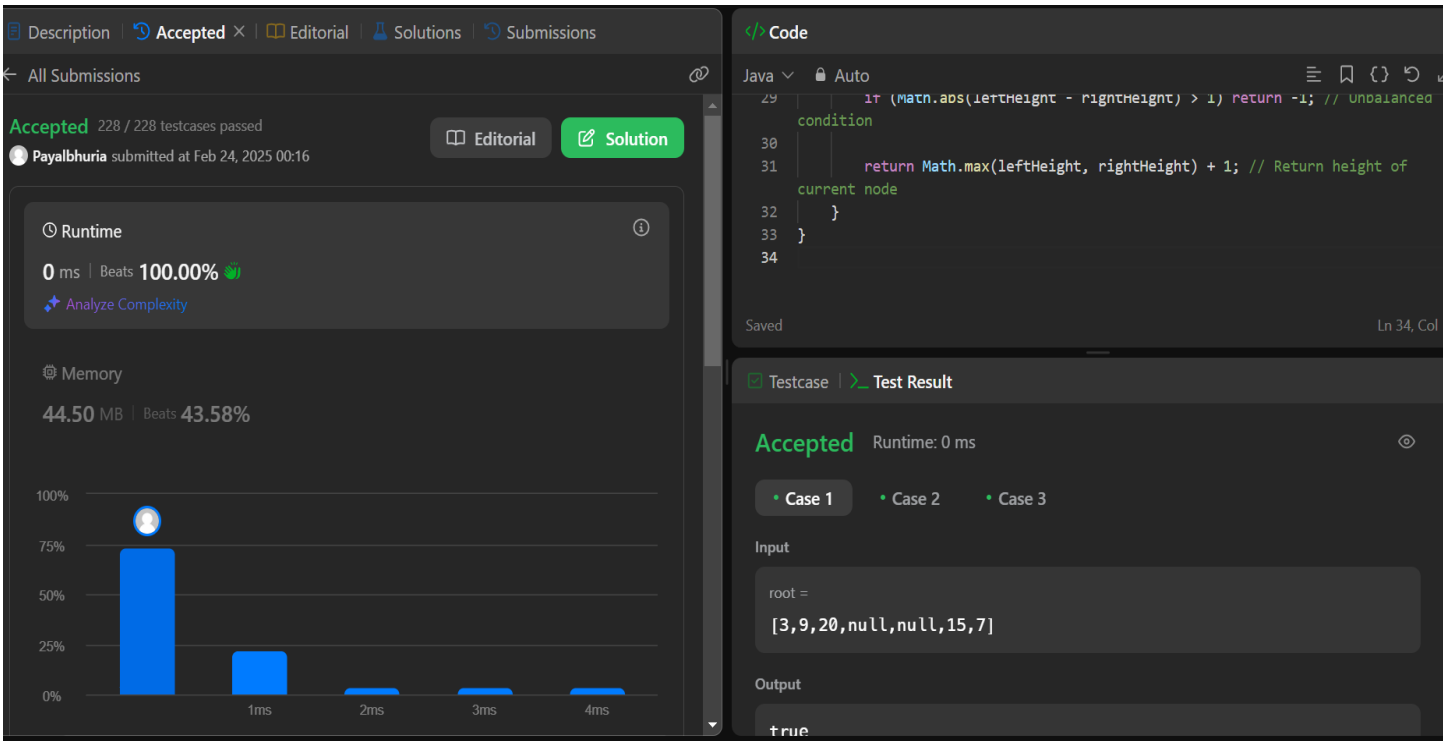
Ques 3:

Aim: Balanced Binary Tree

Code:

```
// Definition for a binary tree node.
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 isBalanced(TreeNode root) {
        return checkHeight(root) != -1;
    }

    private int checkHeight(TreeNode node) {
        if (node == null) return 0; // Base case: Null tree is balanced

        int leftHeight = checkHeight(node.left);
        if (leftHeight == -1) return -1; // If left subtree is unbalanced, return immediately

        int rightHeight = checkHeight(node.right);
        if (rightHeight == -1) return -1; // If right subtree is unbalanced, return immediately

        if (Math.abs(leftHeight - rightHeight) > 1) return -1; // Unbalanced condition

        return Math.max(leftHeight, rightHeight) + 1; // Return height of current node
    }
}
```

Submission Screenshot:



Ques 4:

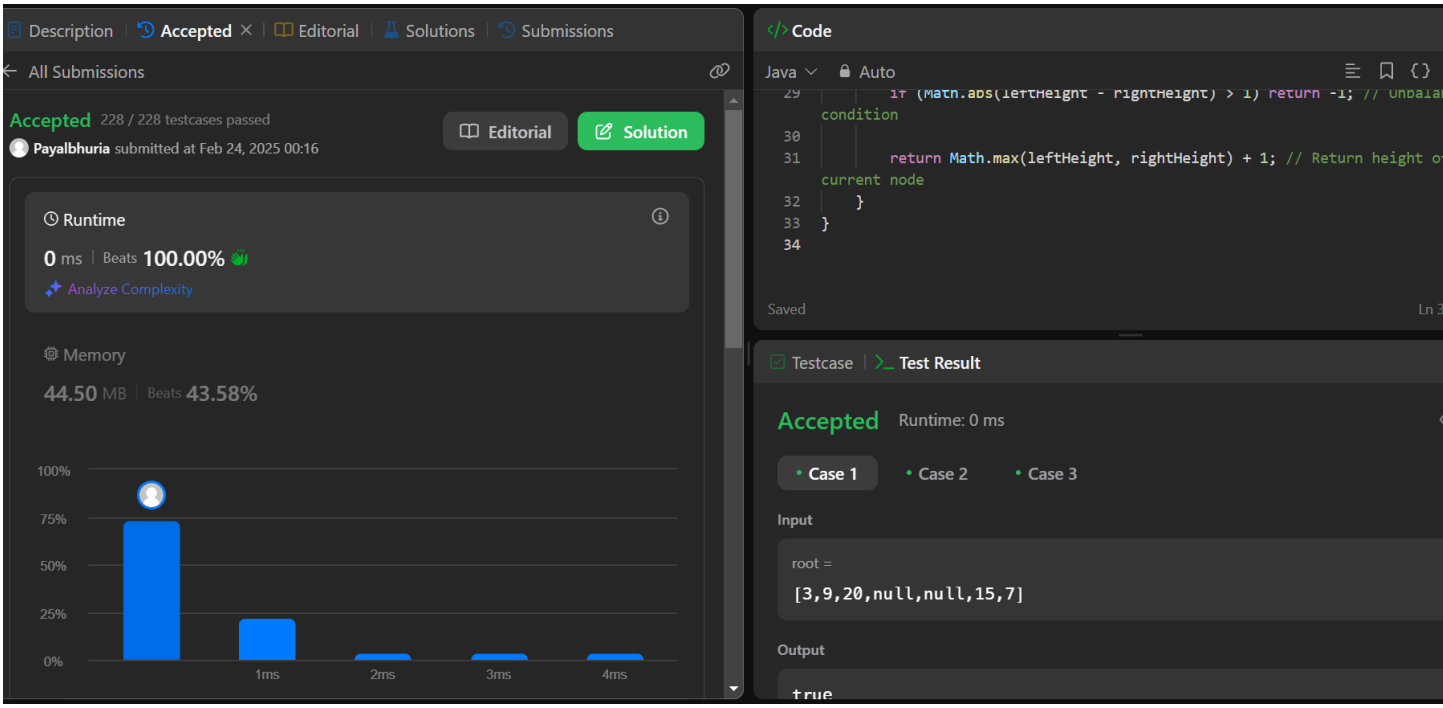
Aim: Path Sum

Code:

```
// Definition for a binary tree node.
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 hasPathSum(TreeNode root, int targetSum) {
            if (root == null) return false; // Empty tree case

            // If it's a leaf node, check if the sum matches
            if (root.left == null && root.right == null && root.val == targetSum) {
                return true;
            }

            // Recursive check for left and right subtrees with updated sum
            int newSum = targetSum - root.val;
            return hasPathSum(root.left, newSum) || hasPathSum(root.right, newSum);
        }
}
```

Submission Screenshot:



Ques 5:

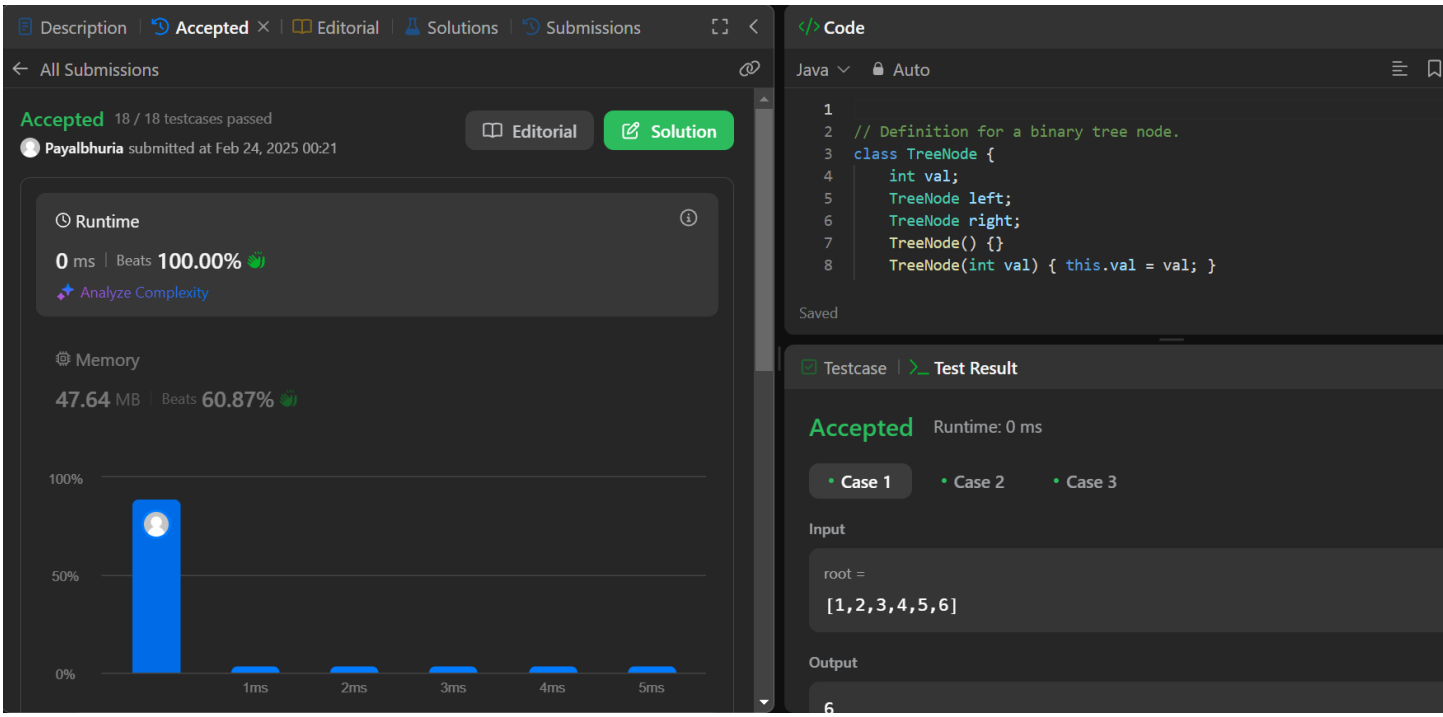
Aim: Count Complete Tree Nodes

Code:

```
// Definition for a binary tree node.
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 countNodes(TreeNode root) {
        if (root == null) return 0; // Base case: Empty tree

        int leftHeight = getLeftHeight(root);
        int rightHeight = getRightHeight(root);

        if (leftHeight == rightHeight) {
            // Perfect binary tree formula: (2^h) - 1
            return (1 << leftHeight) - 1;
        }

        // If not perfect, count recursively
        return 1 + countNodes(root.left) + countNodes(root.right);
    }
```

```
    private int getLeftHeight(TreeNode node) {
        int height = 0;
        while (node != null) {
            height++;
            node = node.left;
        }
        return height;
    }

    private int getRightHeight(TreeNode node) {
        int height = 0;
        while (node != null) {
            height++;
            node = node.right;
        }
        return height;
    }
}
```

Submission Screenshot:



Ques 6:

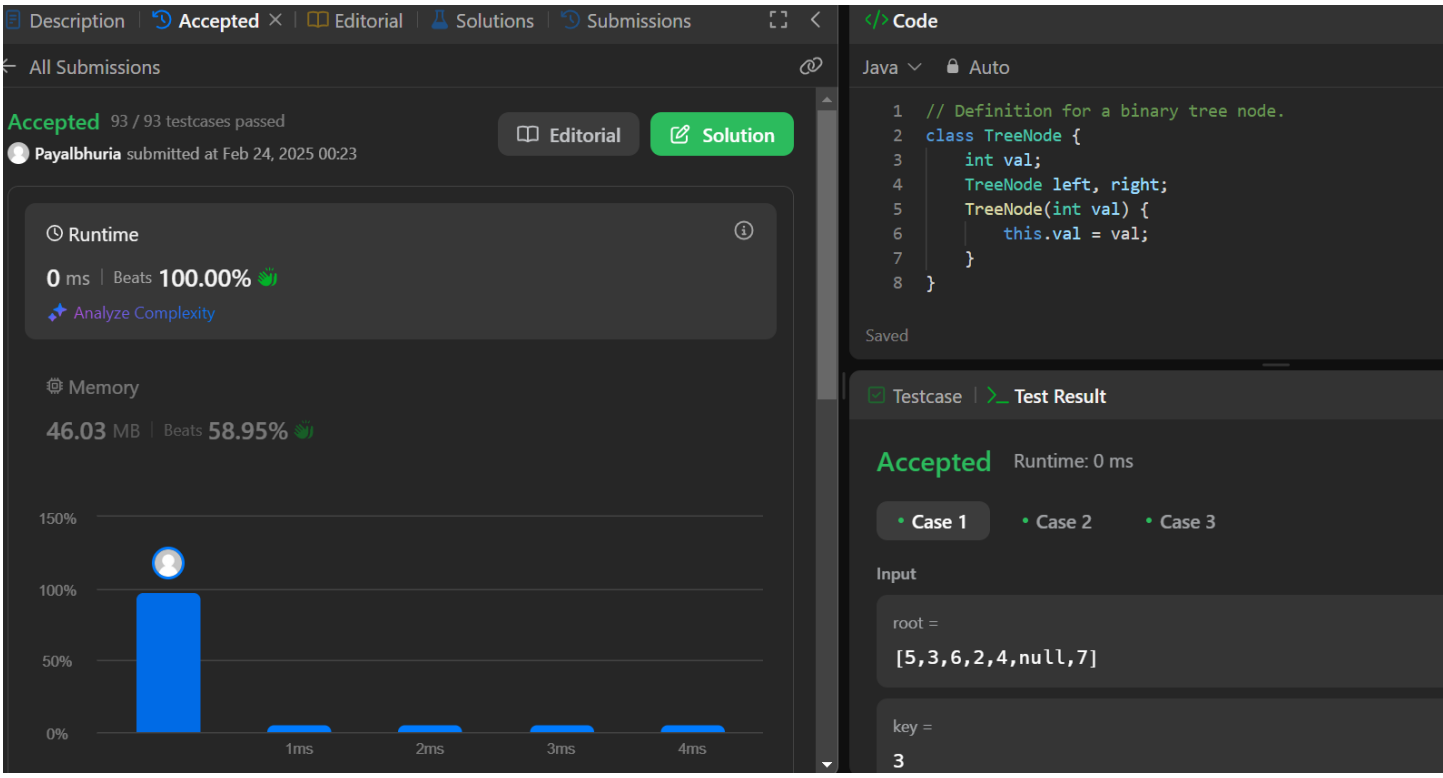
Aim: Delete node in a BST

Code:

```
// Definition for a binary tree node.
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int val) {
        this.val = val;
    }
}

class Solution {
    public TreeNode deleteNode(TreeNode root, int key) {
        if (root == null) return null; // Base case: empty tree
```

```
        if (key < root.val) {
            root.left = deleteNode(root.left, key); // Search in left subtree
        } else if (key > root.val) {
            root.right = deleteNode(root.right, key); // Search in right subtree
        } else {
            // Node to be deleted found
            if (root.left == null) return root.right; // Case 1 & 2: No child / One child (right)
            if (root.right == null) return root.left; // Case 2: One child (left)

            // Case 3: Node has two children, find inorder successor (smallest in right subtree)
            TreeNode successor = findMin(root.right);
            root.val = successor.val; // Replace value
            root.right = deleteNode(root.right, successor.val); // Delete successor
        }
        return root;
    }

    // Helper function to find the minimum value node in BST
    private TreeNode findMin(TreeNode node) {
        while (node.left != null) {
            node = node.left;
        }
        return node;
    }
}
```

Submission Screenshot:



Ques 7:

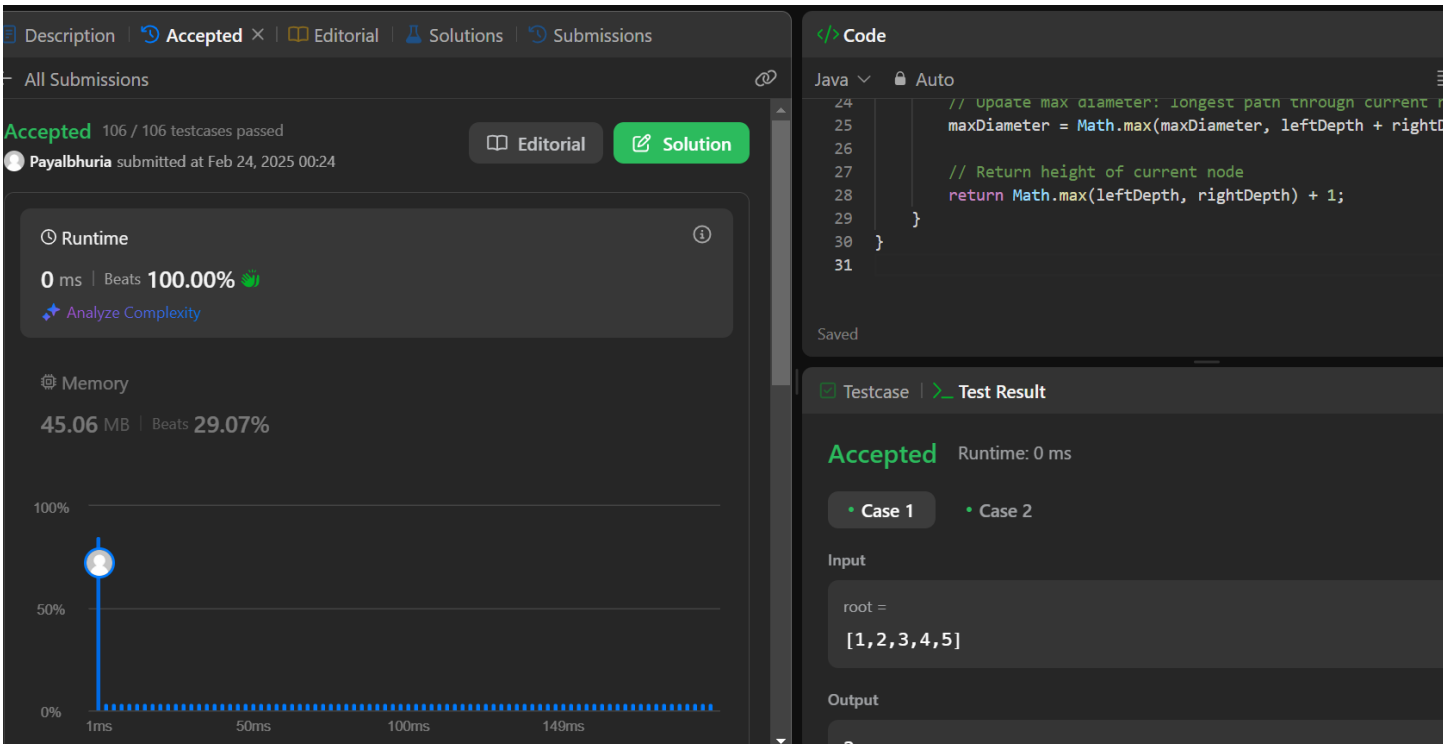
Aim: Diameter of Binary Tree

Code:

```
// Definition for a binary tree node.
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int val) {
        this.val = val;
    }
}

class Solution {
    private int maxDiameter = 0; // Stores the max diameter

    public int diameterOfBinaryTree(TreeNode root) {
```

```
        depth(root);
        return maxDiameter;
    }

    private int depth(TreeNode node) {
        if (node == null) return 0; // Base case

        int leftDepth = depth(node.left);   // Height of left subtree
        int rightDepth = depth(node.right); // Height of right subtree

        // Update max diameter: longest path through current node
        maxDiameter = Math.max(maxDiameter, leftDepth + rightDepth);

        // Return height of current node
        return Math.max(leftDepth, rightDepth) + 1;
    }
}
```

Submission Screenshot:


Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 106 / 106 testcases passed
Payalbhuria submitted at Feb 24, 2025 00:24
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
45.06 MB | Beats 29.07%
100%
50%
0%
1ms
50ms
100ms
149ms
Code
Java
Auto
maxDiameter = Math.max(maxDiameter, leftDepth + rightD
// Return height of current node
return Math.max(leftDepth, rightDepth) + 1;
}
}
Saved
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[1,2,3,4,5]
Output