Experiment 2

Student Name: Garv Gupta
UID: 22BCS11750
Branch: BE-CSE
Section/Group: 22BCS_IOT-638/B
Semester: 6th
Date of Performance: 21/01/2025
Subject Name: AP Lab-II
Subject Code: 22CSP-351

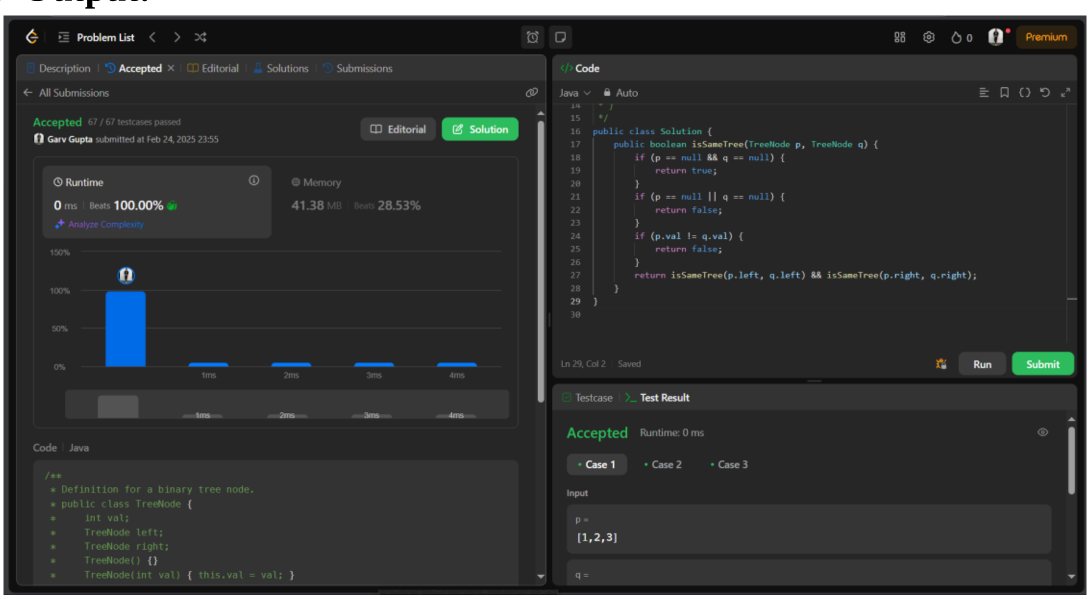
A. Same Tree

1. **Aim:** Given the roots of two binary trees p and q, write a function to check if they are the same or not.Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

2. **Code:**

```
public 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);
    }
}
```

3. **Output:**



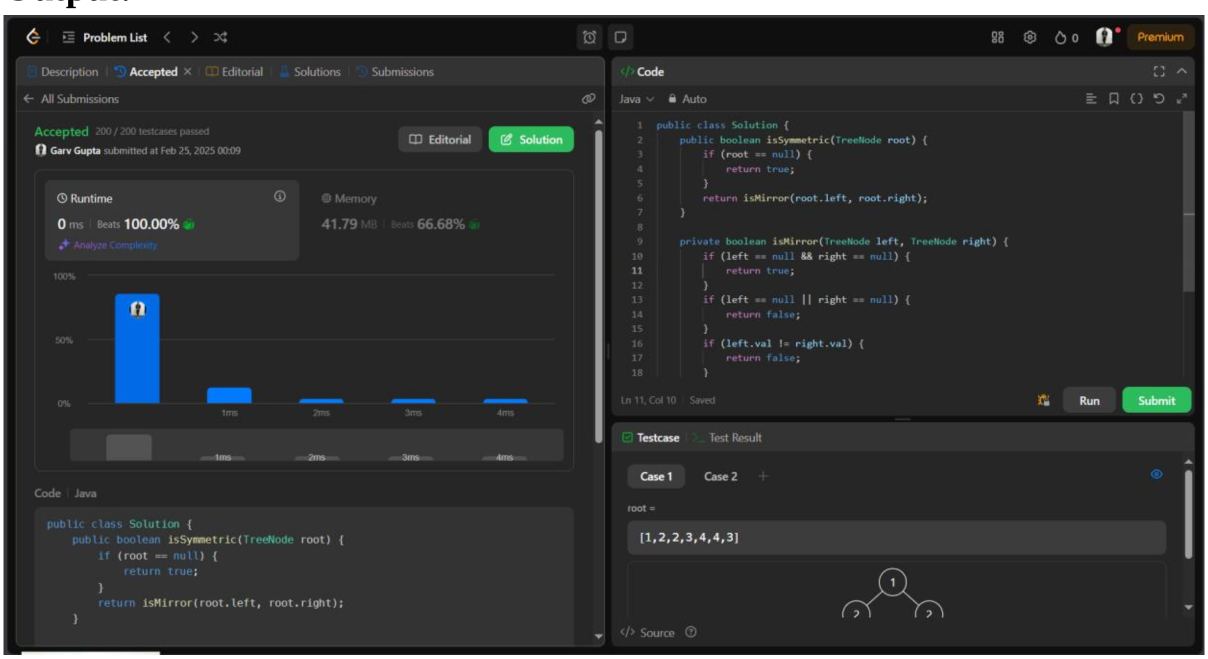
B. Symmetric Tree

1. **Aim:** Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center)

2. **Code:**

```
public class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null) {
            return true;
        }
        return isMirror(root.left, root.right);
    }
    private boolean isMirror(TreeNode left, TreeNode right) {
        if (left == null && right == null) {
            return true;
        }
        if (left == null || right == null) {
            return false;
        }
        if (left.val != right.val) {
            return false;
        }
        return isMirror(left.left, right.right) && isMirror(left.right, right.left);
    }
}
```

3. **Output:**



C. Balanced Binary Tree

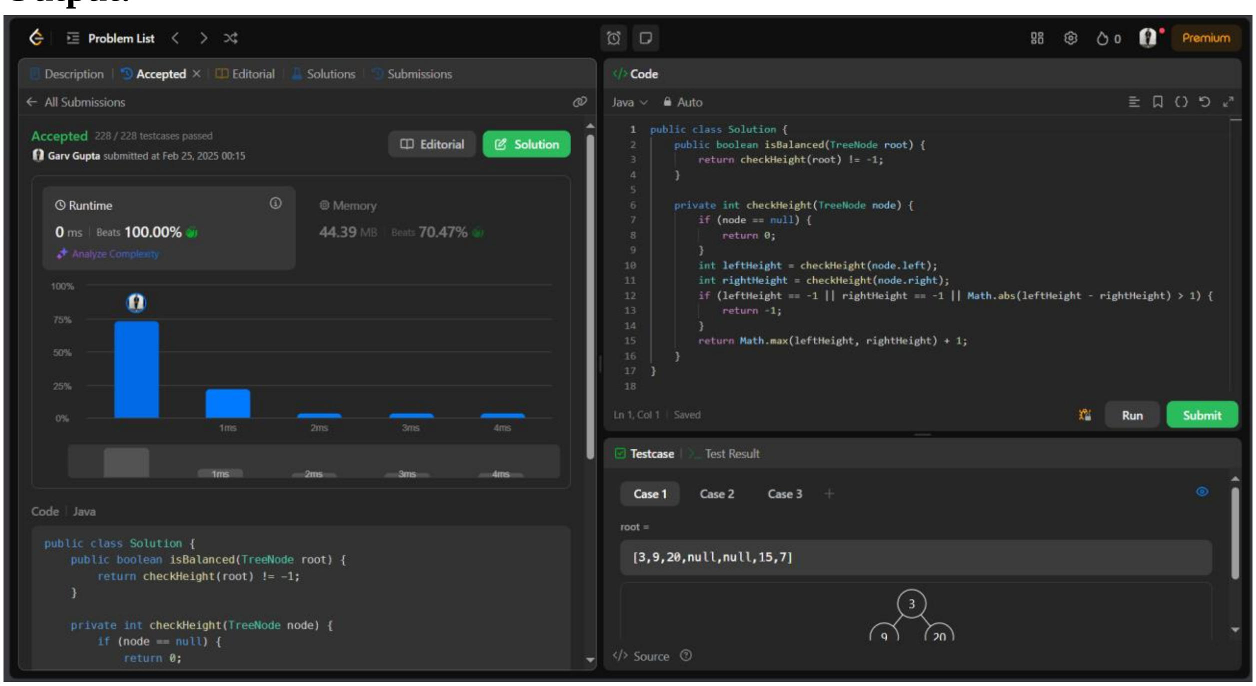
1. Aim: Given a binary tree, determine if it is height-balanced.

2. Code:

```
public class Solution {
    public boolean isBalanced(TreeNode root) {
        return checkHeight(root) != -1;
    }

    private int checkHeight(TreeNode node) {
        if (node == null) {
            return 0;
        }
        int leftHeight = checkHeight(node.left);
        int rightHeight = checkHeight(node.right);
        if (leftHeight == -1 || rightHeight == -1 || Math.abs(leftHeight - rightHeight) > 1) {
            return -1;
        }
        return Math.max(leftHeight, rightHeight) + 1;
    }
}
```

3. Output:



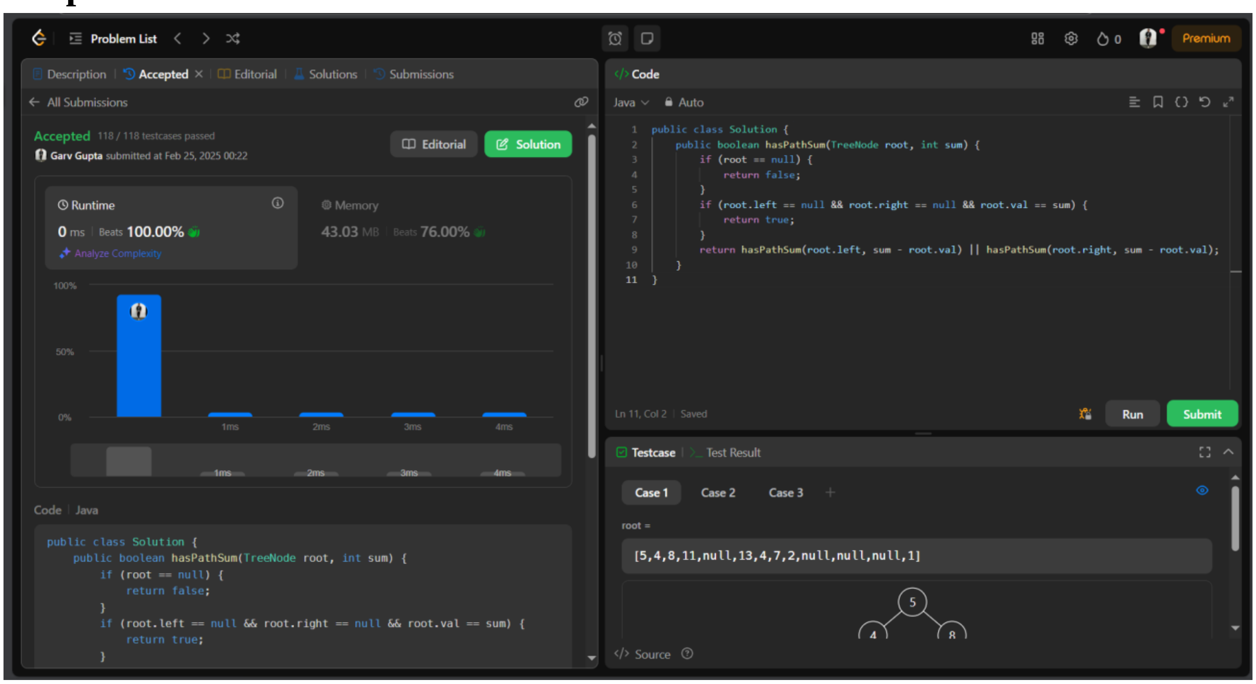
D. Path Sum

1. **Aim:** Given the root of a binary tree and an integer target Sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals target Sum. A leaf is a node with no children.

2. **Code:**

```
public class Solution {
    public boolean hasPathSum(TreeNode root, int sum) {
        if (root == null) {
            return false;
        }
        if (root.left == null && root.right == null && root.val == sum) {
            return true;
        }
        return hasPathSum(root.left, sum - root.val) || hasPathSum(root.right, sum -
root.val);
    }
}
```

3. **Output:**



E. Count Complete Tree

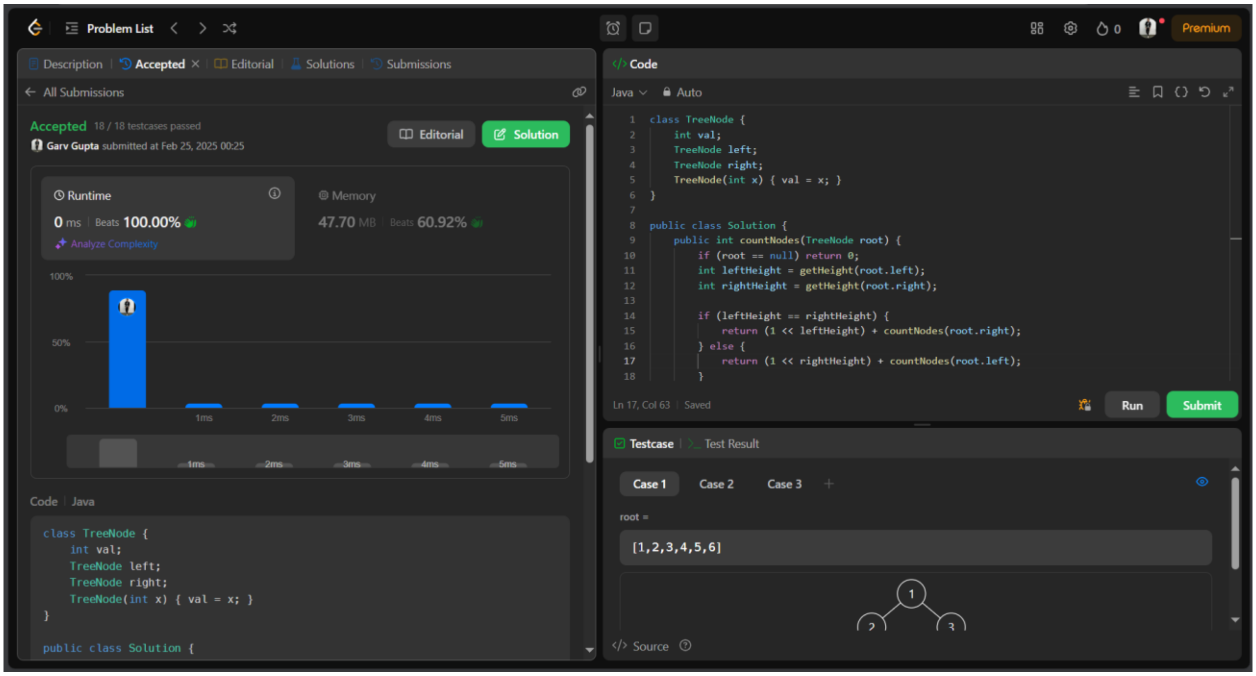
1. **Aim:** Given the root of a complete binary tree, return the number of the nodes in the tree. According to Wikipedia, every level, except possibly the last, is completely filled in a complete binary tree, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes CO3inclusive at the last level h. Design an algorithm that runs in less than O(n) time complexity.

2. **Code:**

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}
public class Solution {
    public int countNodes(TreeNode root) {
        if (root == null) return 0;
        int leftHeight = getHeight(root.left);
        int rightHeight = getHeight(root.right);

        if (leftHeight == rightHeight) {
            return (1 << leftHeight) + countNodes(root.right);
        } else {
            return (1 << rightHeight) + countNodes(root.left);
        }
    }
    private int getHeight(TreeNode node) {
        int height = 0;
        while (node != null) {
            height++;
            node = node.left;
        }
        return height;
    }
}
```

3. Output:



F. Delete Node in a BST

1. **Aim:** Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST. Basically, the deletion can be divided into two stages: Search for a node to remove. If the node is found, delete the node.

2. **Code:**

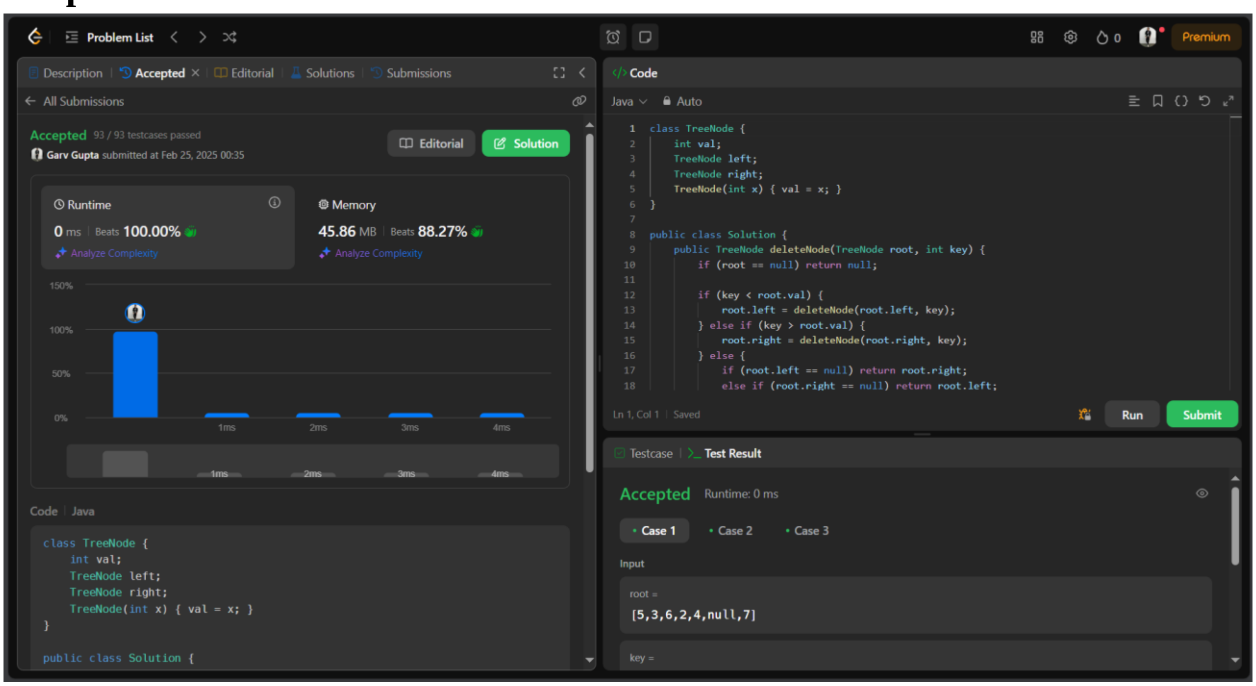
```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}
public class Solution {
    public TreeNode deleteNode(TreeNode root, int key) {
        if (root == null) return null;

        if (key < root.val) {
            root.left = deleteNode(root.left, key);
        } else if (key > root.val) {
            root.right = deleteNode(root.right, key);
```

```
        } else {
            if (root.left == null) return root.right;
            else if (root.right == null) return root.left;

            TreeNode minNode = findMin(root.right);
            root.val = minNode.val;
            root.right = deleteNode(root.right, root.val);
        }
        return root;
    }
    private TreeNode findMin(TreeNode node) {
        while (node.left != null) {
            node = node.left;
        }
        return node;
    }
}
```

3. Output:



G. Diameter of Binary Tree

1. **Aim:** Given the root of a binary tree, return the length of the diameter of the tree. The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root. The length of a path between two nodes is represented by the number of edges between them.

2. **Code:**

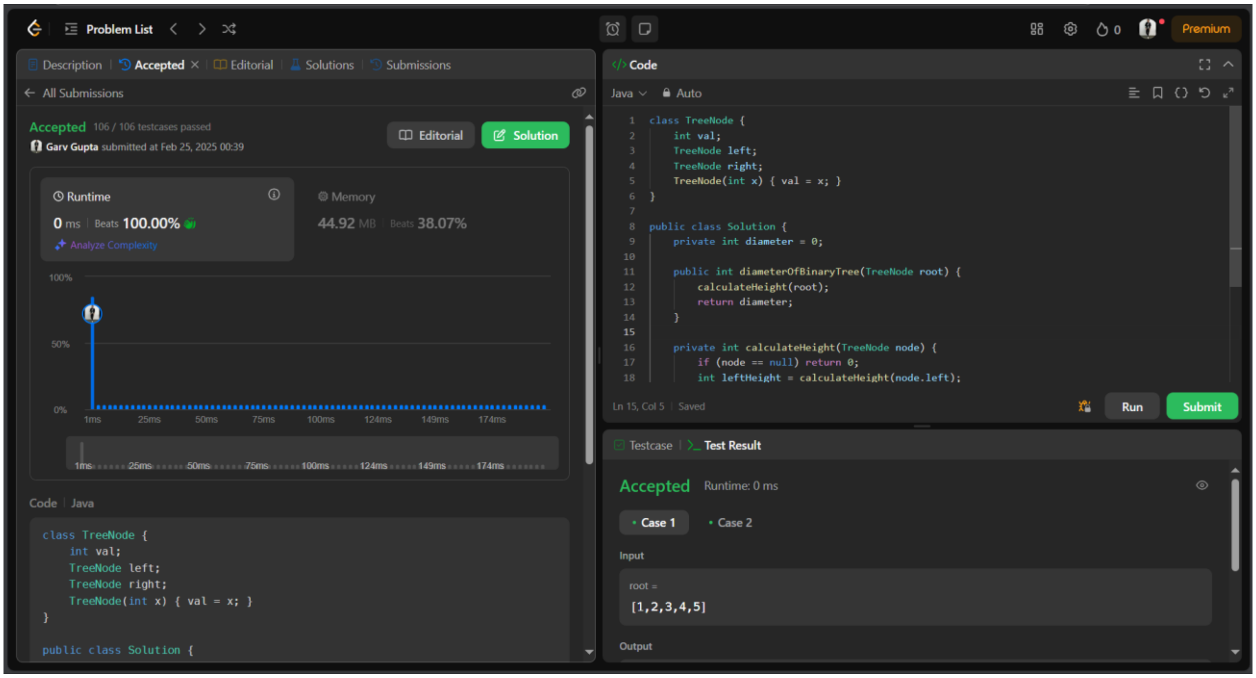
```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

public class Solution {
    private int diameter = 0;

    public int diameterOfBinaryTree(TreeNode root) {
        calculateHeight(root);
        return diameter;
    }

    private int calculateHeight(TreeNode node) {
        if (node == null) return 0;
        int leftHeight = calculateHeight(node.left);
        int rightHeight = calculateHeight(node.right);
        diameter = Math.max(diameter, leftHeight + rightHeight);
        return Math.max(leftHeight, rightHeight) + 1;
    }
}
```

3. Output:


Problem List
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 106 / 106 testcases passed
Garv Gupta submitted at Feb 25, 2025 00:39
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
44.92 MB | Beats 38.07%
100%
50%
0%
1ms 25ms 50ms 75ms 100ms 124ms 149ms 174ms
Code | Java
class TreeNode {
int val;
TreeNode left;
TreeNode right;
TreeNode(int x) { val = x; }
}
public class Solution {
Code
Java
Auto
class TreeNode {
int val;
TreeNode left;
TreeNode right;
TreeNode(int x) { val = x; }
}
public class Solution {
private int diameter = 0;
public int diameterOfBinaryTree(TreeNode root) {
calculateHeight(root);
return diameter;
}
private int calculateHeight(TreeNode node) {
if (node == null) return 0;
int leftHeight = calculateHeight(node.left);
Ln 15, Col 5 | Saved
Run
Submit
Testcase
Test Result
Accepted Runtime: 0 ms
Case 1
Case 2
Input
root =
[1,2,3,4,5]
Output