Experiment-6

Student Name: Ankit | **UID: 22BET10181**
Branch: BE-IT | **Section/Group: 22BET_IOT-702/B**
Semester: 6th | **Date of Performance: 28th Feb, 2025**
Subject Name: AP- 2 | **Code: 22ITP-351**

Problem-1

1. **Aim:** Given the root of a binary tree, return its maximum depth.

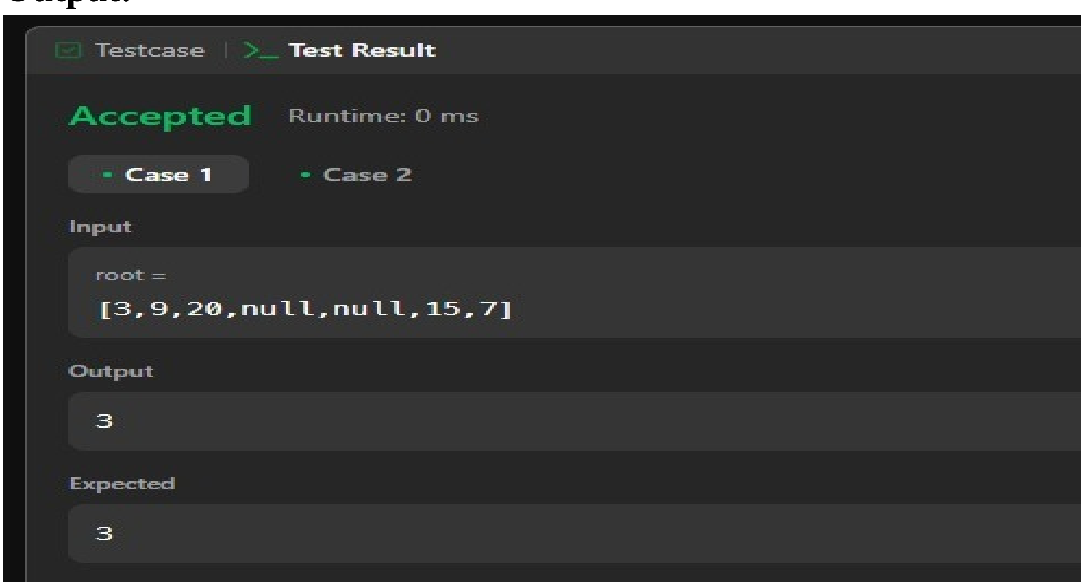
2. **Code:**

```
import java.util.*;


class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) return result;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        boolean leftToRight = true;

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            LinkedList<Integer> level = new LinkedList<>();

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

```
                TreeNode node = queue.poll();

                if (leftToRight) {
                    level.addLast(node.val);
                } else {
                    level.addFirst(node.val);
                }

                if (node.left != null) queue.offer(node.left);
                if (node.right != null) queue.offer(node.right);
            }

            result.add(level);
            leftToRight = !leftToRight;
        }

        return result;
    }
}

// Definition for a binary tree node
class TreeNode {
    int val;
    TreeNode left, right;

    TreeNode(int val) {
        this.val = val;
    }
```

```
}

// Test the code
public class Main {
    public static void main(String[] args) {
        Solution solution = new Solution();

        TreeNode root = new TreeNode(3);
        root.left = new TreeNode(9);
        root.right = new TreeNode(20);
        root.right.left = new TreeNode(15);
        root.right.right = new TreeNode(7);

        System.out.println(solution.zigzagLevelOrder(root));
    }
}
```

3. Output:



Problem-2

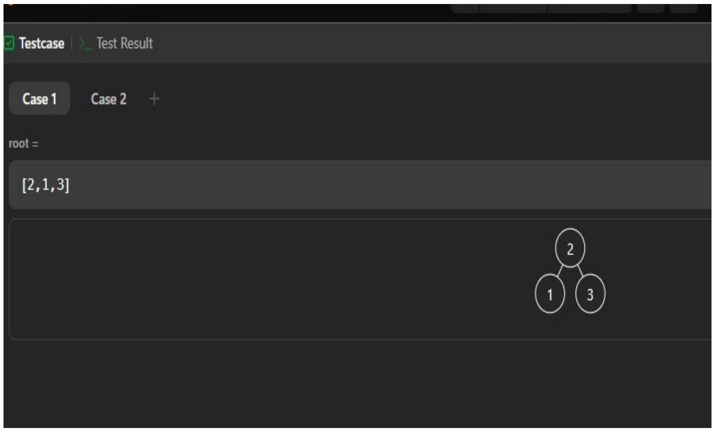
1. **Aim:** Given the root of a binary tree, determine if it is a valid binary search tree (BST).

2. **Code:**

```
// Definition for a binary tree node.
class TreeNode {
    int val;
    TreeNode left, right;

    TreeNode(int val) {
        this.val = val;
    }
}

public class Solution {

    public boolean isValidBST(TreeNode root) {
        return validate(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    private boolean validate(TreeNode node, long min, long max) {
        if (node == null) return true;
        if (node.val <= min || node.val >= max) return false;

        return validate(node.left, min, node.val) && validate(node.right, node.val, max);
    }
```

```
    public TreeNode buildSampleTree(boolean valid) {
        if (valid) {
            TreeNode root = new TreeNode(2);
            root.left = new TreeNode(1);
            root.right = new TreeNode(3);
            return root;
        } else {
            TreeNode root = new TreeNode(5);
            root.left = new TreeNode(1);
            root.right = new TreeNode(4);
            root.right.left = new TreeNode(3);
            root.right.right = new TreeNode(6);
            return root;
        }
    }

    public static void main(String[] args) {
        Solution solution = new Solution();

        TreeNode validBST = solution.buildSampleTree(true);
        System.out.println("Valid BST: " + solution.isValidBST(validBST));

        TreeNode invalidBST = solution.buildSampleTree(false);
        System.out.println("Invalid BST: " + solution.isValidBST(invalidBST));
    }
}
```

3. Output:



Problem-3

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

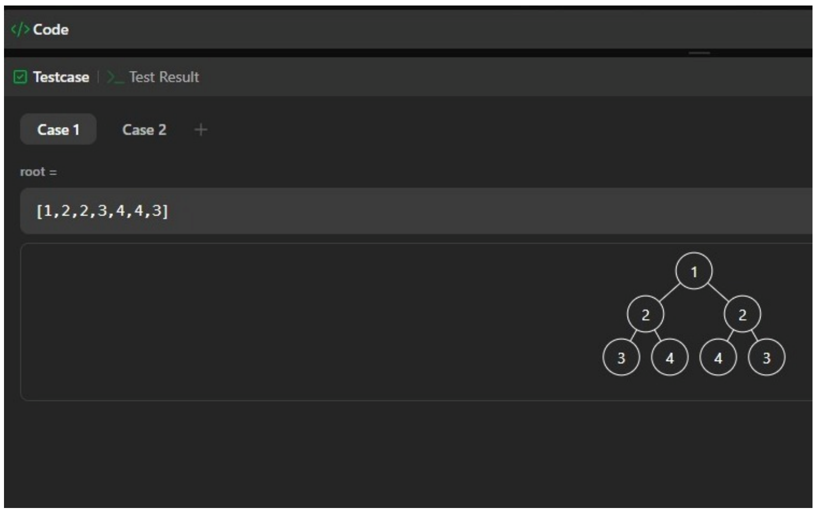
2. **Code:**

```
// Definition for a binary tree node.
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class Solution {

    // Main function to check if tree is symmetric
    public boolean isSymmetric(TreeNode root) {
        if (root == null) {
            return true; // An empty tree is symmetric
```

```
    }
    return isMirror(root.left, root.right);
}
// Helper function to check if two trees are mirror images
private boolean isMirror(TreeNode t1, TreeNode t2) {
    if (t1 == null && t2 == null) {
        return true; // Both are null, symmetric
    }
    if (t1 == null || t2 == null) {
        return false; // One is null, not symmetric
    }
    // Check values and cross symmetry of subtrees
    return (t1.val == t2.val)
            && isMirror(t1.left, t2.right)
            && isMirror(t1.right, t2.left);
}
// Sample tree builder for [1,2,2,3,4,4,3] (symmetric tree)
public TreeNode buildSampleTree1() {
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(2);
    root.left.left = new TreeNode(3);
    root.left.right = new TreeNode(4);
    root.right.left = new TreeNode(4);
    root.right.right = new TreeNode(3);
    return root;
}
// Sample tree builder for [1,2,2,null,3,null,3] (asymmetric tree)
public TreeNode buildSampleTree2() {
    TreeNode root = new TreeNode(1);
    root.left = new TreeNode(2);
    root.right = new TreeNode(2);
    root.left.right = new TreeNode(3);
    root.right.right = new TreeNode(3);
    return root;
}
public static void main(String[] args) {
    Solution solution = new Solution();
```

```
        TreeNode root1 = solution.buildSampleTree1();
        System.out.println("Tree 1 is symmetric: " + solution.isSymmetric(root1)); // true
        TreeNode root2 = solution.buildSampleTree2();
        System.out.println("Tree 2 is symmetric: " + solution.isSymmetric(root2)); // false
    }
}
```

3. Output:



Problem-4

1. Aim: Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

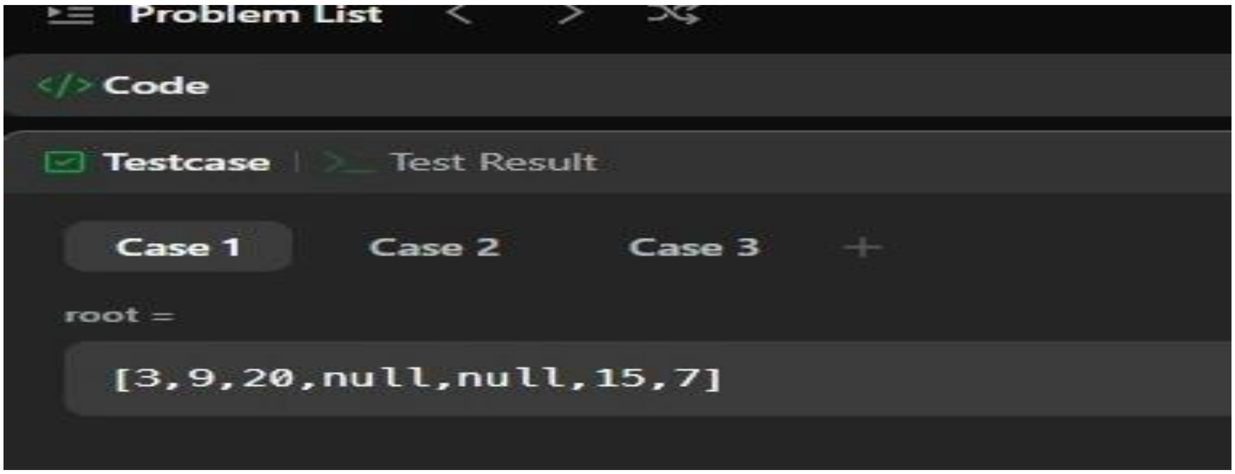
2. Code:

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left, right;

    TreeNode(int val) {
        this.val = val;
    }
}
```

```
public class Solution {

    public List<List<Integer>> levelOrder(TreeNode root) {
        List<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();
            List<Integer> currentLevel = new ArrayList<>();

            for (int i = 0; i < levelSize; i++) {
                TreeNode node = queue.poll();
                currentLevel.add(node.val);

                if (node.left != null) queue.add(node.left);
                if (node.right != null) queue.add(node.right);
            }

            result.add(currentLevel);
        }

        return result;
    }

    public TreeNode buildSampleTree() {
        TreeNode root = new TreeNode(3);
        root.left = new TreeNode(9);
        root.right = new TreeNode(20);
        root.right.left = new TreeNode(15);
        root.right.right = new TreeNode(7);
        return root;
    }

    public static void main(String[] args) {
```

```
        Solution solution = new Solution();
        TreeNode root = solution.buildSampleTree();
        System.out.println("Level Order Traversal: " + solution.levelOrder(root));
    }
}
```

Output:



Problem-5

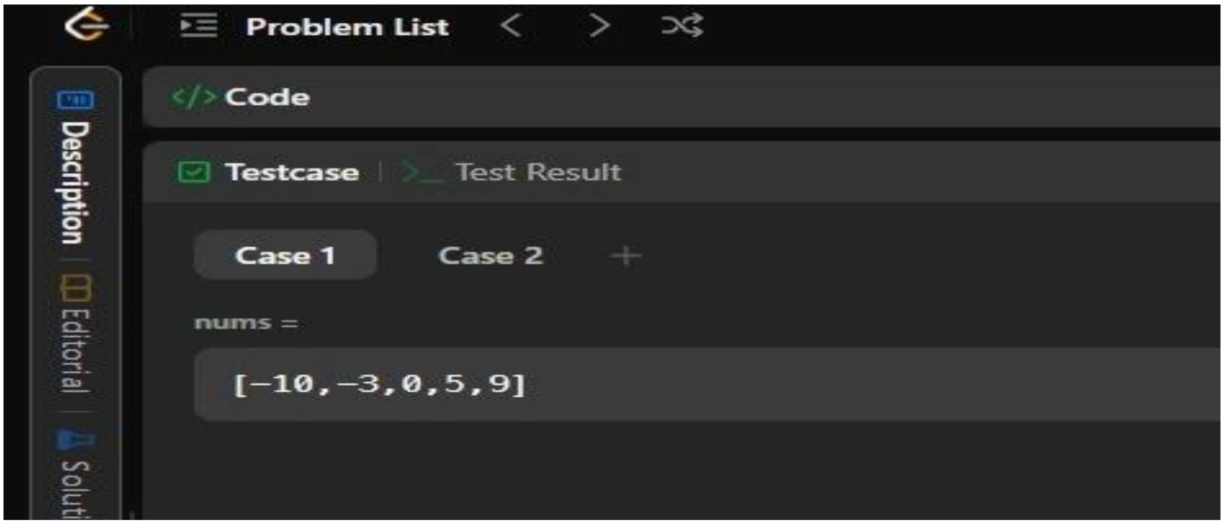
1. **Aim:** Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree
2. **Code:**

```
// Definition for a binary tree node.
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}
```

```java
public class Solution {

    // Main function to convert sorted array to height-balanced BST
    public TreeNode sortedArrayToBST(int[] nums) {
        if (nums == null || nums.length == 0) {
            return null;
        }
        return buildBST(nums, 0, nums.length - 1);
    }
    // Helper function using recursive approach (divide and conquer)
    private TreeNode buildBST(int[] nums, int left, int right) {
        if (left > right) {
            return null; // Base case: no elements to form a tree
        }
        int mid = left + (right - left) / 2; // Find middle element
        TreeNode root = new TreeNode(nums[mid]); // Make mid element root
        // Recursively build left and right subtrees
        root.left = buildBST(nums, left, mid - 1);
        root.right = buildBST(nums, mid + 1, right);
        return root;
    }
    // Inorder traversal to print tree nodes
    public void inorderTraversal(TreeNode root) {
        if (root == null) {
            return;
        }
        inorderTraversal(root.left);
        System.out.print(root.val + " ");
        inorderTraversal(root.right);
    }
    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] nums1 = {-10, -3, 0, 5, 9};
        TreeNode root1 = solution.sortedArrayToBST(nums1);
        System.out.print("Inorder Traversal of BST (Example 1): ");
        solution.inorderTraversal(root1);
        System.out.println();
```

```
        int[] nums2 = {1, 3};
        TreeNode root2 = solution.sortedArrayToBST(nums2);
        System.out.print("Inorder Traversal of BST (Example 2): ");
        solution.inorderTraversal(root2);
    }
}
```

3. **Output:**



Problem-6

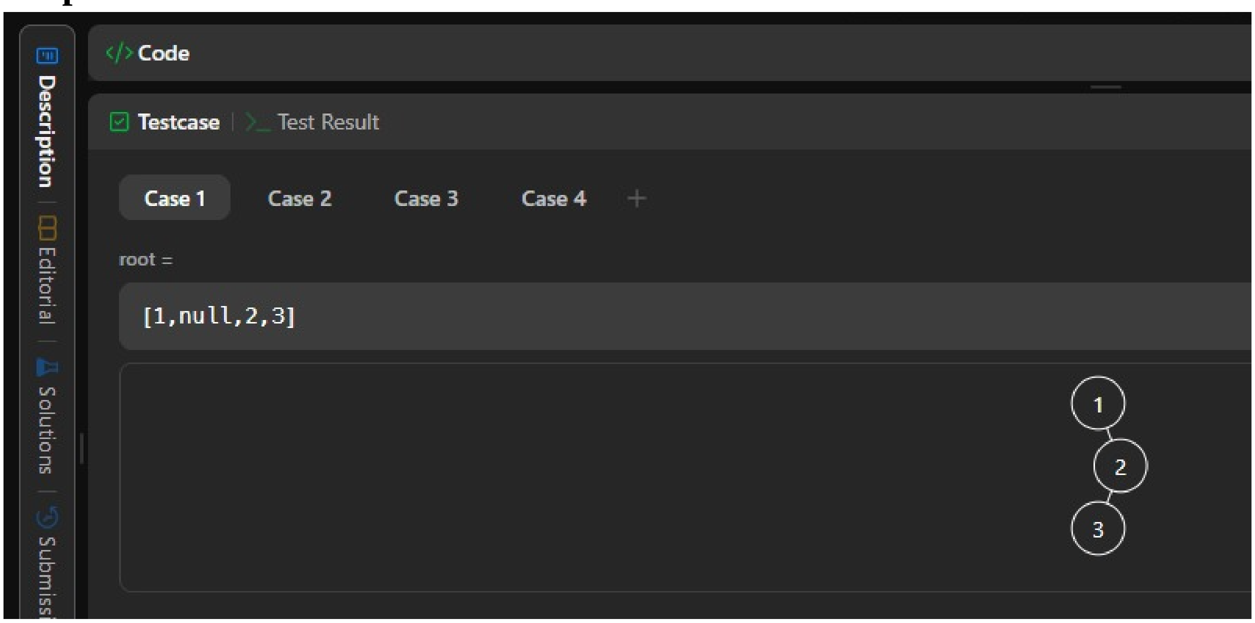
1. **Aim:** Given the root of a binary tree, return the inorder traversal of its nodes' values.
2. **Code:**

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left, right;

    TreeNode(int val) {
        this.val = val;
    }
}
```

```
public class Solution {

    public List<Integer> inorderTraversalRecursive(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        inorderHelper(root, result);
        return result;
    }

    private void inorderHelper(TreeNode node, List<Integer> result) {
        if (node == null) return;
        inorderHelper(node.left, result);
        result.add(node.val);
        inorderHelper(node.right, result);
    }

    public List<Integer> inorderTraversalIterative(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        Stack<TreeNode> stack = new Stack<>();
        TreeNode current = root;

        while (current != null || !stack.isEmpty()) {
            while (current != null) {
                stack.push(current);
                current = current.left;
            }
            current = stack.pop();
            result.add(current.val);
            current = current.right;
        }
        return result;
    }

    public TreeNode buildSampleTree() {
        TreeNode root = new TreeNode(1);
        root.right = new TreeNode(2);
        root.right.left = new TreeNode(3);
        return root;
```

```
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        TreeNode root = solution.buildSampleTree();

        System.out.println("Recursive Inorder Traversal: " +
solution.inorderTraversalRecursive(root));
        System.out.println("Iterative Inorder Traversal: " +
solution.inorderTraversalIterative(root));
    }
}
```

3. Output:



<u>Problem-7</u>

1. **Aim:** Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary tree.
2. **Code:**

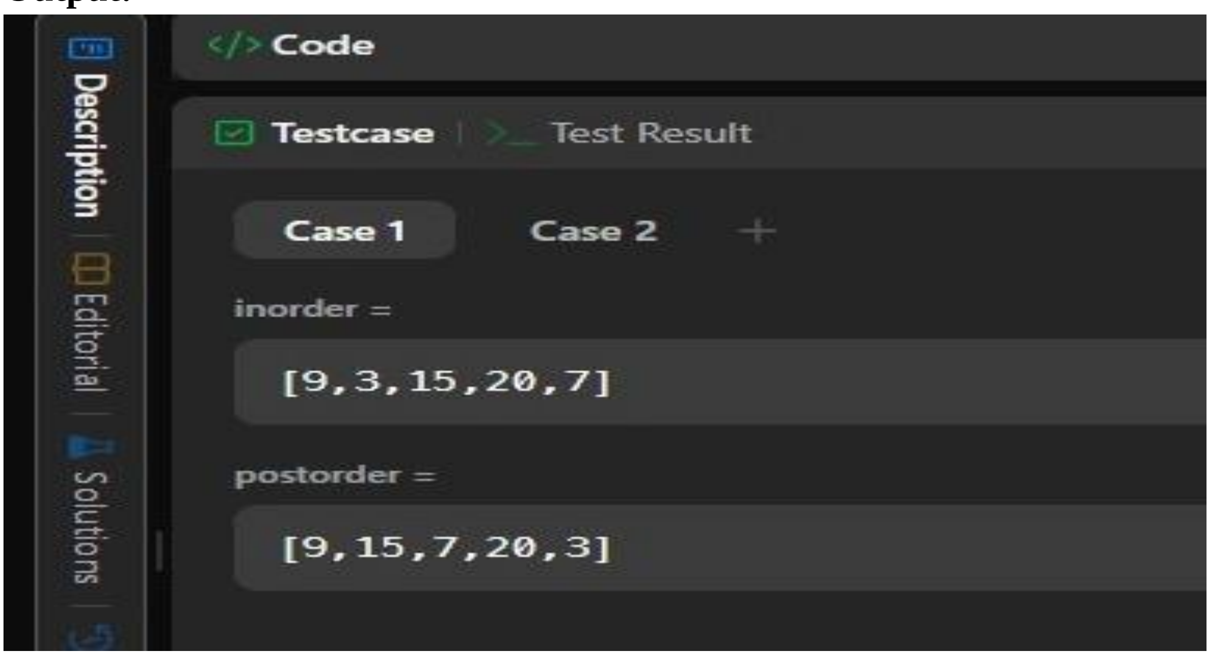
```
// Definition for a binary tree node
```

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class Solution {

    private Map<Integer, Integer> inorderMap; // To store index of elements in inorder
array
    private int postIndex; // Pointer for postorder array

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

        // Store element-to-index mapping for quick look-up
        for (int i = 0; i < inorder.length; i++) {
            inorderMap.put(inorder[i], i);
        }

        // Build the tree recursively
        return buildSubTree(postorder, 0, inorder.length - 1);
    }

    private TreeNode buildSubTree(int[] postorder, int left, int right) {
        if (left > right) {
            return null;
        }
```

```
    // Get current root value from postorder traversal
    int rootVal = postorder[postIndex--];
    TreeNode root = new TreeNode(rootVal);

    // Find root index in inorder array
    int rootIndex = inorderMap.get(rootVal);

    // Build right subtree first (because postorder visits left->right->root)
    root.right = buildSubTree(postorder, rootIndex + 1, right);
    root.left = buildSubTree(postorder, left, rootIndex - 1);

    return root;
}

// Helper method to print tree in level-order for visualization
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> result = new ArrayList<>();
    if (root == null) return result;

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

    while (!queue.isEmpty()) {
        int levelSize = queue.size();
        List<Integer> level = new ArrayList<>();

        for (int i = 0; i < levelSize; i++) {
            TreeNode node = queue.poll();
            if (node != null) {
                level.add(node.val);
                if (node.left != null) queue.offer(node.left);
                if (node.right != null) queue.offer(node.right);
            }
        }
        result.add(level);
    }
    return result;
}
```

```
    public static void main(String[] args) {
        Solution solution = new Solution();

        // Example 1
        int[] inorder1 = {9, 3, 15, 20, 7};
        int[] postorder1 = {9, 15, 7, 20, 3};
        TreeNode root1 = solution.buildTree(inorder1, postorder1);
        System.out.println("Level-order traversal: " + solution.levelOrder(root1));

        // Example 2
        int[] inorder2 = {-1};
        int[] postorder2 = {-1};
        TreeNode root2 = solution.buildTree(inorder2, postorder2);
        System.out.println("Level-order traversal: " + solution.levelOrder(root2));
    }
}
```

3. Output:



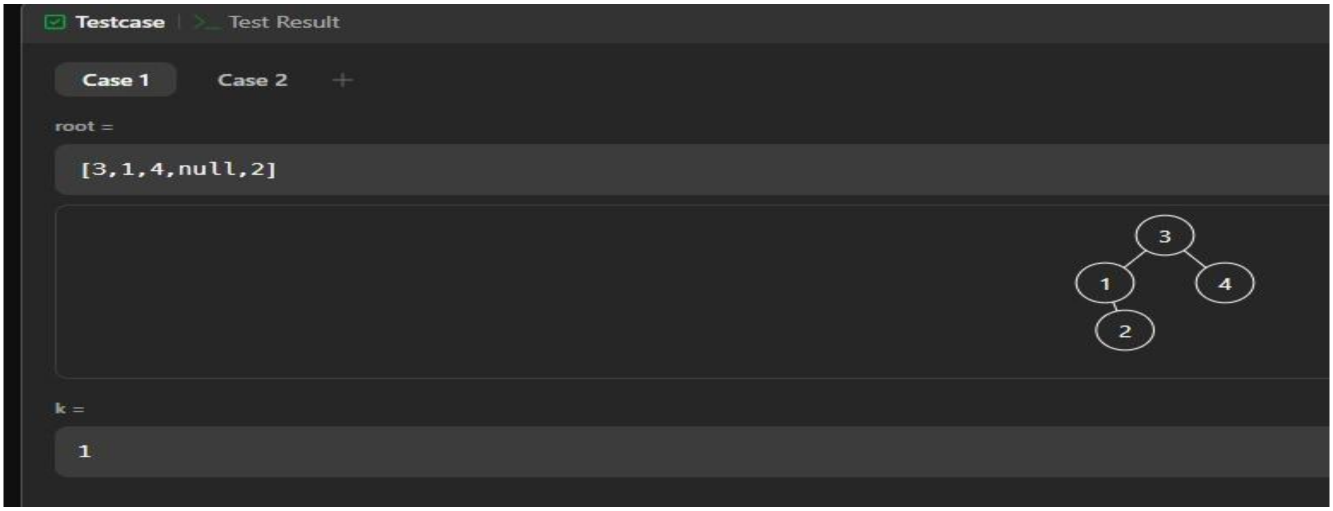
Problem-8

1. **Aim:**
   Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) of all the values of the nodes in the tree.
2. **Code:**

```
// Definition for a binary tree node
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}
public class Solution {
    // Method 1: Inorder Traversal (Iterative)
    public int kthSmallest(TreeNode root, int k) {
        Stack<TreeNode> stack = new Stack<>();
        TreeNode current = root;

        while (current != null || !stack.isEmpty()) {
            // Traverse left subtree
            while (current != null) {
                stack.push(current);
                current = current.left;
            }
            // Process node
            current = stack.pop();
            k--;
            if (k == 0) {
                return current.val; // Found kth smallest element
```

```
            }
            // Move to right subtree
            current = current.right;
        }
        return -1; // Should never reach here if input is valid
    }
    // Sample tree builder for testing
    public TreeNode buildSampleTree1() {
        TreeNode root = new TreeNode(3);
        root.left = new TreeNode(1);
        root.left.right = new TreeNode(2);
        root.right = new TreeNode(4);
        return root;
    }
    public TreeNode buildSampleTree2() {
        TreeNode root = new TreeNode(5);
        root.left = new TreeNode(3);
        root.right = new TreeNode(6);
        root.left.left = new TreeNode(2);
        root.left.right = new TreeNode(4);
        root.left.left.left = new TreeNode(1);
        return root;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();

        // Example 1: root = [3,1,4,null,2], k = 1
        TreeNode root1 = solution.buildSampleTree1();
        System.out.println("Kth smallest element: " + solution.kthSmallest(root1, 1)); //
Output: 1

        // Example 2: root = [5,3,6,2,4,null,null,1], k = 3
        TreeNode root2 = solution.buildSampleTree2();
        System.out.println("Kth smallest element: " + solution.kthSmallest(root2, 3)); //
Output: 3
    }
}}
```

3. Output:



Problem-9

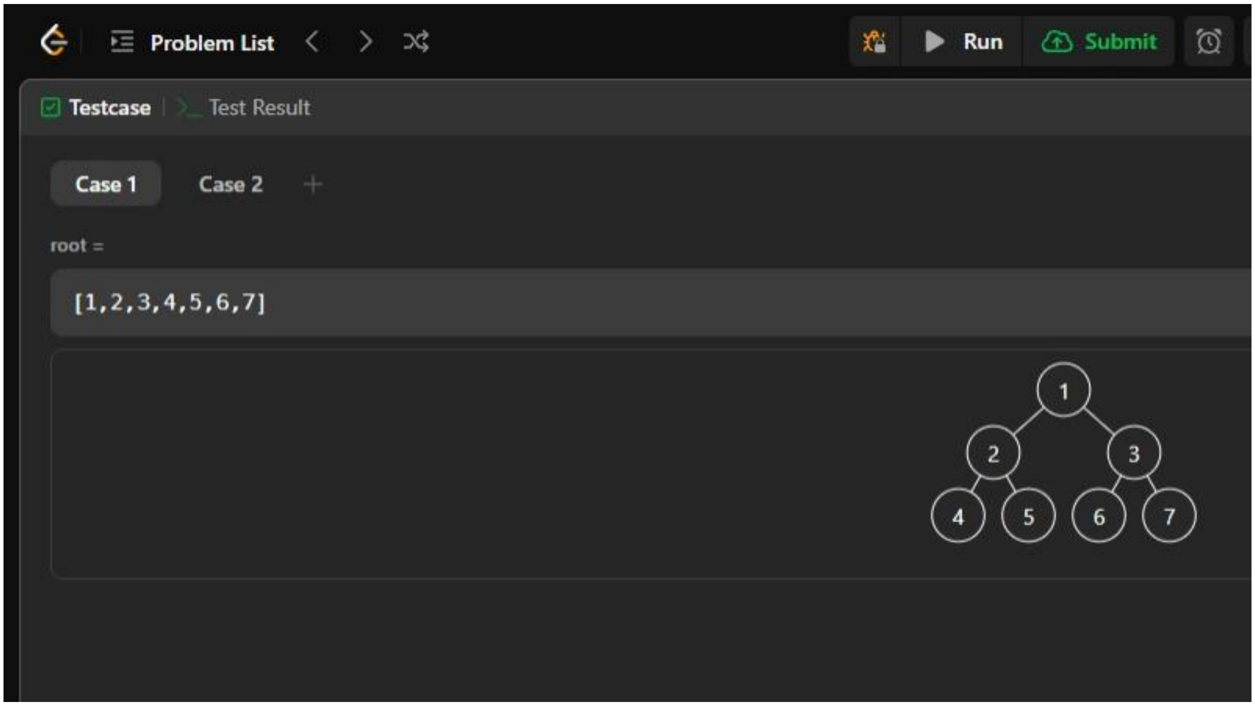
1. **Aim:** Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.
2. **Code:**

```
// Definition for a Node
class Node {
    public int val;
    public Node left;
    public Node right;
    public Node next;

    public Node(int val) {
        this.val = val;
        left = null;
        right = null;
        next = null;
    }
}

public class Solution {
```

```
// Using constant space (O(1)) iterative approach
public Node connect(Node root) {
    if (root == null) {
        return null;
    }

    Node leftmost = root;

    // Traverse level by level
    while (leftmost.left != null) {
        Node current = leftmost;

        while (current != null) {
            // Connect the left and right children of the current node
            current.left.next = current.right;

            // Connect the right child to the next left child if exists
            if (current.next != null) {
                current.right.next = current.next.left;
            }

            // Move to the next node at the current level
            current = current.next;
        }

        // Move to the next level (leftmost node of the next level)
        leftmost = leftmost.left;
    }

    return root;
}

// Helper method to print the tree's next pointers (for testing)
public void printTree(Node root) {
    Node levelStart = root;
    while (levelStart != null) {
        Node current = levelStart;
        while (current != null) {
```

```
            System.out.print(current.val + " -> ");
            current = current.next;
          }
          System.out.println("NULL");
          levelStart = levelStart.left; // move to next level
        }
    }

    public static void main(String[] args) {
        Solution solution = new Solution();

        // Construct the example tree: [1,2,3,4,5,6,7]
        Node root = new Node(1);
        root.left = new Node(2);
        root.right = new Node(3);
        root.left.left = new Node(4);
        root.left.right = new Node(5);
        root.right.left = new Node(6);
        root.right.right = new Node(7);

        // Connect nodes at each level
        solution.connect(root);

        // Print the next pointers for each level
        solution.printTree(root);
    }
}
```

3. Output:



Problem-10

1. **Aim:** Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).
2. **Code:**

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left, right;

    TreeNode(int val) {
        this.val = val;
    }
}

public class Solution {
```

```
public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
    List<List<Integer>> result = new ArrayList<>();
    if (root == null) return result;

    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    boolean leftToRight = true;

    while (!queue.isEmpty()) {
        int levelSize = queue.size();
        LinkedList<Integer> level = new LinkedList<>();

        for (int i = 0; i < levelSize; i++) {
            TreeNode node = queue.poll();

            if (leftToRight) {
                level.addLast(node.val);
            } else {
                level.addFirst(node.val);
            }

            if (node.left != null) queue.offer(node.left);
            if (node.right != null) queue.offer(node.right);
        }

        result.add(level);
        leftToRight = !leftToRight;
    }

    return result;
}

public static void main(String[] args) {
    Solution solution = new Solution();
    TreeNode root = new TreeNode(3);
    root.left = new TreeNode(9);
    root.right = new TreeNode(20);
    root.right.left = new TreeNode(15);
```

```
        root.right.right = new TreeNode(7);
        System.out.println("Zigzag Level Order Traversal: " +
solution.zigzagLevelOrder(root));
    }
}
```

3. **Output:**