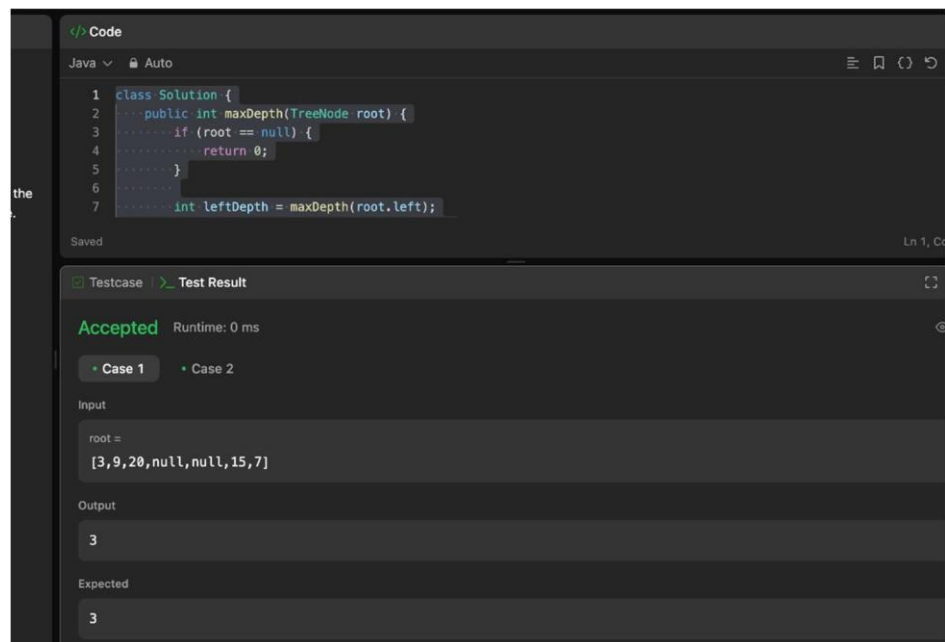


Problem 1

Aim:

Maximum Depth of Binary Tree Code:

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



```
</> Code  
Java ▾ Auto  
1 class Solution {  
2     public int maxDepth(TreeNode root) {  
3         if (root == null) {  
4             return 0;  
5         }  
6         int leftDepth = maxDepth(root.left);  
7  
Saved Ln 1, Co  
  
Testcase Test Result  
Accepted Runtime: 0 ms  
• Case 1 • Case 2  
Input  
root =  
[3,9,20,null,null,15,7]  
Output  
3  
Expected  
3
```

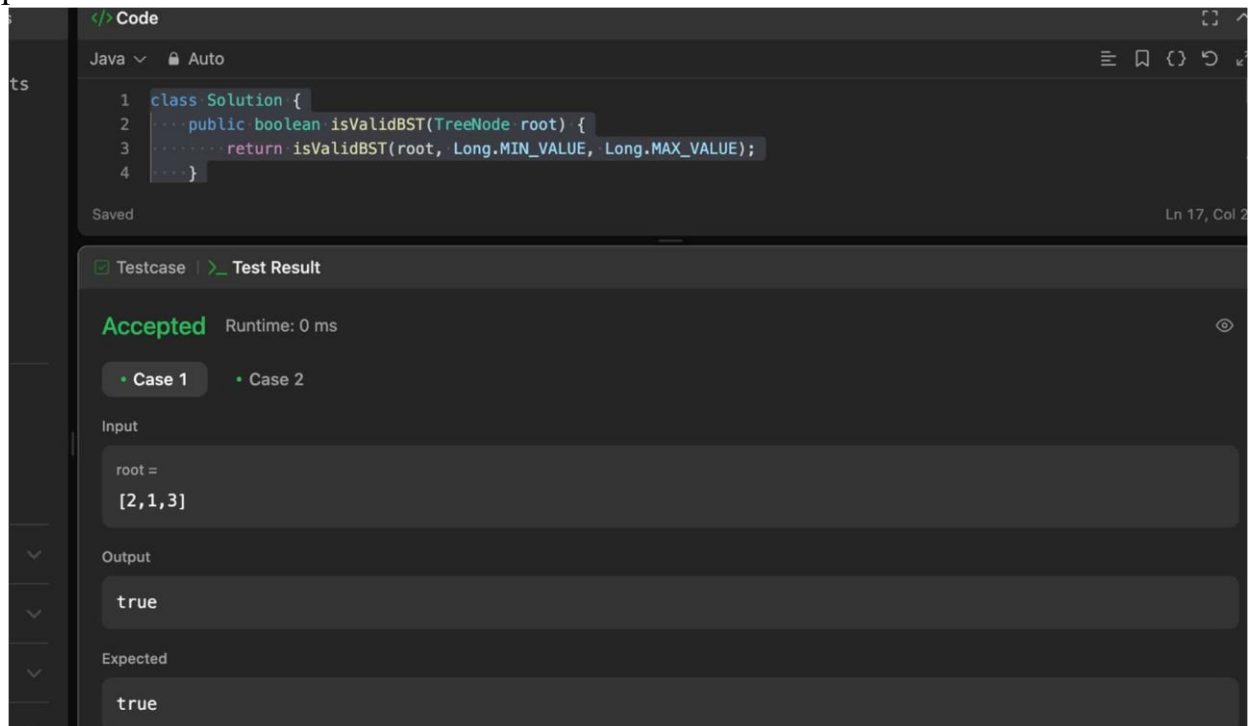
Problem 2

Aim:

Validate Binary Search Tree Code:

```
class Solution {  
    public boolean isValidBST(TreeNode root) {  
        return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);  
    }  
  
    private boolean isValidBST(TreeNode node, long min, long max) {  
        if (node == null) {            return true;  
        }  
  
        if (node.val <= min || node.val >= max) {  
            return false;  
        }  
  
        return isValidBST(node.left, min, node.val) && isValidBST(node.right, node.val, max);  
    }  
}
```

Output:



Problem 3

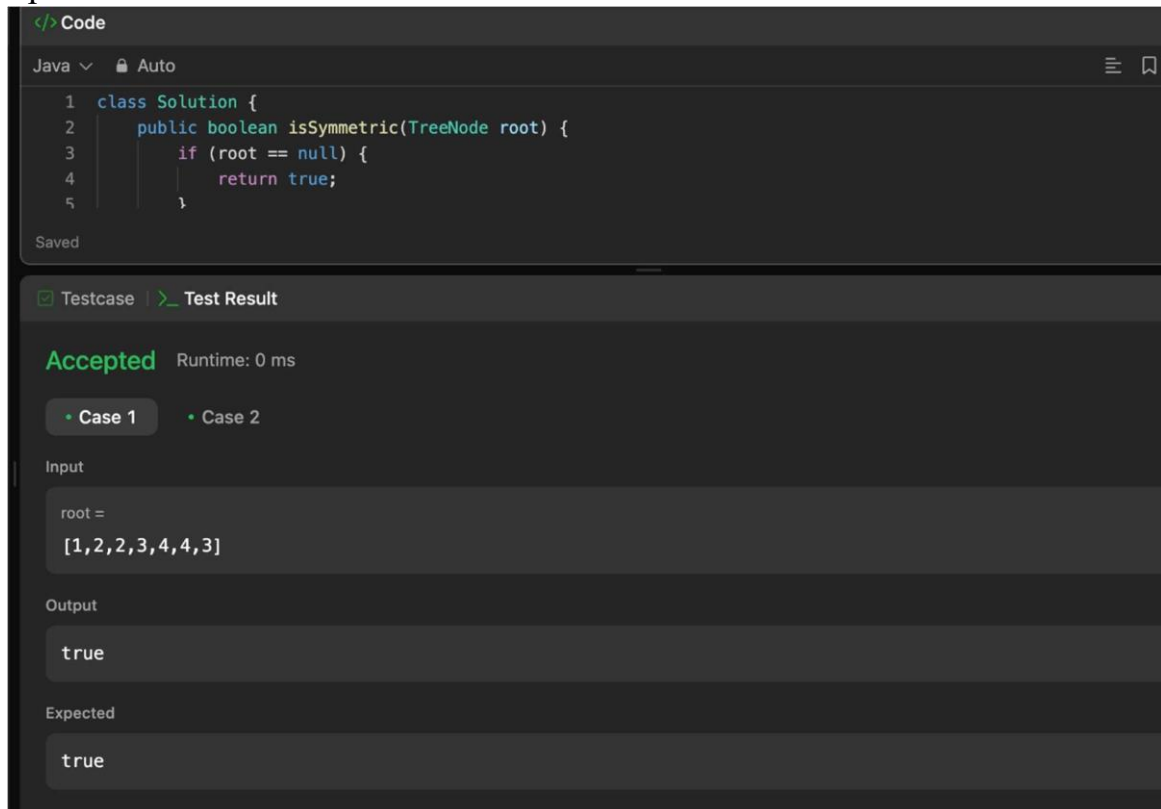
Aim: Symmetric Tree

```
Code : 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;
        }

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

Output:



```
</> Code
Java ▾ 🔒 Auto
1 class Solution {
2     public boolean isSymmetric(TreeNode root) {
3         if (root == null) {
4             return true;
5         }
6     }
7 }
Saved
```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =
[1,2,2,3,4,4,3]

Output

true

Expected

true

Aim: Binary Tree Level Order Traversal Code:

```
import java.util.*;

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.offer(root);

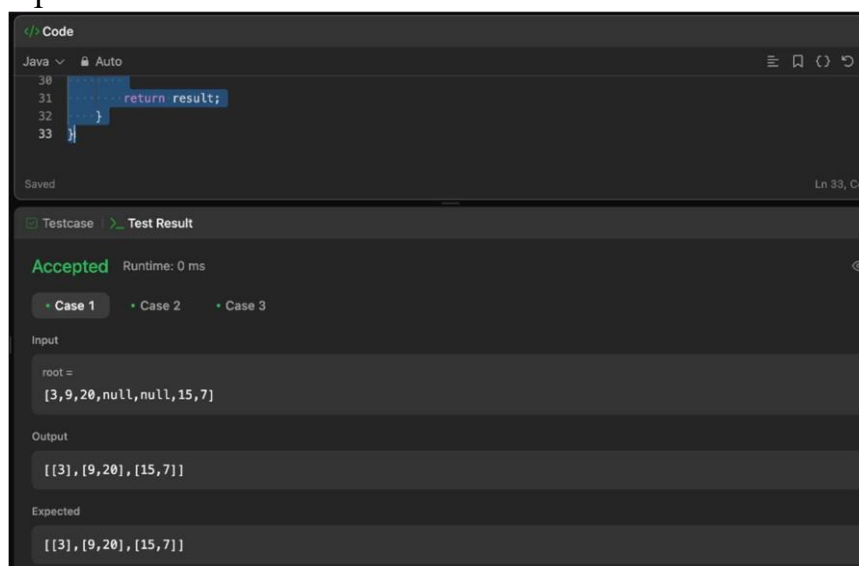
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            List<Integer> currentLevel = new ArrayList<>();

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

                if (currentNode.left != null) {
                    queue.offer(currentNode.left);
                }
                if (currentNode.right != null) {
                    queue.offer(currentNode.right);
                }
            }
            result.add(currentLevel);
        }

        return result;
    }
}
```

Output:



The screenshot displays a code editor with the Java code for binary tree level order traversal. The code is saved and shows the final return statement. Below the code editor, the test results are shown, indicating that the code is accepted. The test case details include the input root node, the output list of lists, and the expected result, all of which match.

```
Code
Java Auto
30
31     return result;
32 }
33

Saved Ln 33, Col 1

Testcase Test Result
Accepted Runtime: 0 ms

Case 1 Case 2 Case 3
Input
root =
[3,9,20,null,null,15,7]
Output
[[3], [9,20], [15,7]]
Expected
[[3], [9,20], [15,7]]
```

Problem 5

Aim:

Convert Sorted Array to Binary Search Tree Code:

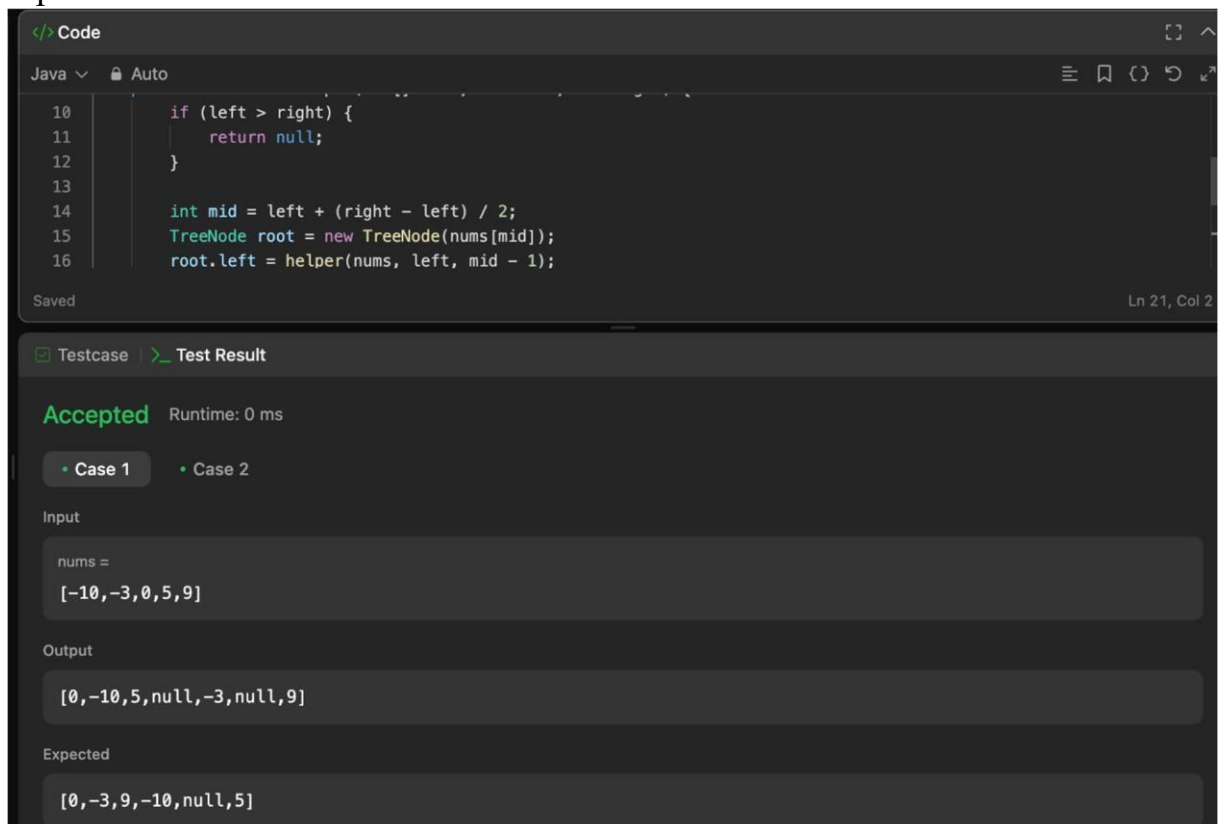
```
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        if (nums == null || nums.length == 0) {
            return null;
        }
        return helper(nums, 0, nums.length - 1);
    }

    private TreeNode helper(int[] nums, int left, int right) {
        if (left > right) {
            return null;
        }

        int mid = left + (right - left) / 2;
        TreeNode root = new TreeNode(nums[mid]);
        root.left = helper(nums, left, mid - 1);
        root.right = helper(nums, mid + 1, right);

        return root;
    }
}
```

Output:



The screenshot shows a code editor with the following Java code:

```
10     if (left > right) {
11         return null;
12     }
13
14     int mid = left + (right - left) / 2;
15     TreeNode root = new TreeNode(nums[mid]);
16     root.left = helper(nums, left, mid - 1);
```

Below the code editor, the test results are displayed:

- Accepted** Runtime: 0 ms
- Case 1** (selected) • Case 2
- Input:** nums = [-10,-3,0,5,9]
- Output:** [0,-10,5,null,-3,null,9]
- Expected:** [0,-3,9,-10,null,5]

Problem 6

Aim:

Binary Tree Inorder Traversal Code:

```
import java.util.*;

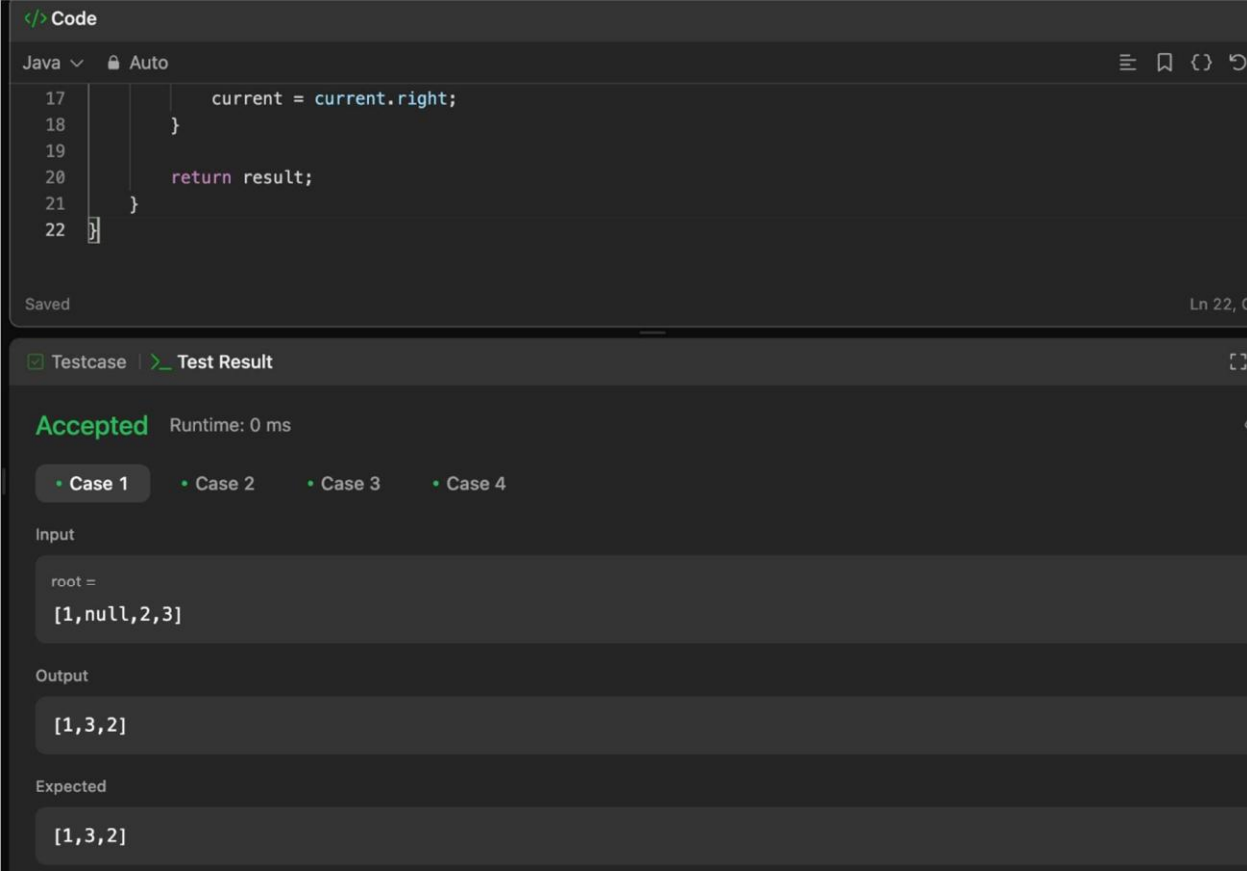
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        Stack<TreeNode> stack = new Stack<>();
        TreeNode current = root;

        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;
    }
}
```

Output:



The screenshot displays a code editor with the following content:

```
</> Code
Java ▾ Auto
17         current = current.right;
18     }
19
20     return result;
21 }
22 }
```

Below the code editor, the test results are shown:

- Testcase** | **Test Result**
- Accepted** Runtime: 0 ms
- **Case 1** • Case 2 • Case 3 • Case 4
- Input**
root =
[1,null,2,3]
- Output**
[1,3,2]
- Expected**
[1,3,2]

Problem 7

Aim:

Construct Binary Tree from Inorder and Postorder Traversal Code:

```

import java.util.*;
class Solution {
    public TreeNode buildTree(int[] inorder,
int[] postorder) {
        if (inorder == null || postorder == null || inorder.length != postorder.length) {
return null;
        }
        Map<Integer, Integer> inorderMap = new HashMap<>();
for (int i = 0; i < inorder.length; i++) {
inorderMap.put(inorder[i], i);
        }
        return buildTreeHelper(inorder, 0, inorder.length - 1, postorder, 0, postorder.length - 1, inorderMap);
    }
    private TreeNode buildTreeHelper(int[] inorder, int inStart, int inEnd, int[] postorder, int postStart, int
postEnd, Map<Integer, Integer> inorderMap) {
        if (inStart > inEnd || postStart > postEnd) {
return
null;
        }
        int rootVal = postorder[postEnd];
TreeNode root = new TreeNode(rootVal);
        int
rootIndex = inorderMap.get(rootVal);
        int
leftSize = rootIndex - inStart;
root.left = buildTreeHelper(inorder, inStart, rootIndex - 1, postorder, postStart, postStart + leftSize - 1,
inorderMap);
        root.right = buildTreeHelper(inorder, rootIndex + 1, inEnd, postorder, postStart + leftSize, postEnd
- 1, inorderMap);
return root;
    }
}

```

The screenshot shows a code editor with the following code:

```

26 root.right = buildTreeHelper(inorder, rootIndex + 1, inEnd, postorder, postStart + leftSize, postEnd - 1,
inorderMap);
27

```

Below the code editor, there is a test case section. It shows the test case is "Accepted" with a runtime of 0 ms. The input is:

```

inorder = [9, 3, 15, 20, 7]
postorder = [9, 15, 7, 20, 3]

```

The output is:

```

[3, 9, 20, null, null, 15, 7]

```

The expected output is:

```

[3, 9, 20, null, null, 15, 7]

```

Output:

Problem 8

Aim:

Kth Smallest element in a BST Code:

```
import java.util.*;
```

```

class Solution {
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        if (inorder == null || postorder == null || inorder.length != postorder.length) {
            return null;
        }
        Map<Integer, Integer> inorderMap = new HashMap<>();
        for (int i = 0; i < inorder.length; i++) {
            inorderMap.put(inorder[i], i);
        }
        return buildTreeHelper(inorder, 0, inorder.length - 1, postorder, 0, postorder.length - 1, inorderMap);
    }

    private TreeNode buildTreeHelper(int[] inorder, int inStart, int inEnd, int[] postorder, int postStart, int postEnd, Map<Integer, Integer> inorderMap) {
        if (inStart > inEnd || postStart > postEnd) {
            return null;
        }

        int rootVal = postorder[postEnd];
        TreeNode root = new TreeNode(rootVal);
        int rootIndex = inorderMap.get(rootVal);
        int leftSize = rootIndex - inStart;

        root.left = buildTreeHelper(inorder, inStart, rootIndex - 1, postorder, postStart, postStart + leftSize - 1, inorderMap);
        root.right = buildTreeHelper(inorder, rootIndex + 1, inEnd, postorder, postStart + leftSize, postEnd - 1, inorderMap);

        return root;
    }

    public int kthSmallest(TreeNode root, int k) {
        Stack<TreeNode> stack = new Stack<>();
        TreeNode current = root;
        int count = 0;

        while (current != null || !stack.isEmpty()) {
            while (current != null) {
                stack.push(current);
                current = current.left;
            }

            current = stack.pop();
            count++;
            if (count == k) {
                return current.val;
            }

            current = current.right;
        }

        return -1; // Should not reach here if k is valid
    }
}

```

Output:

The screenshot shows a code editor with a dark theme. The top section is labeled 'Code' and contains a Java snippet with lines 52 and 53. Line 52 has a blue selection. Below the code editor, there's a 'Testcase' tab and a 'Test Result' tab. The 'Test Result' tab shows a green 'Accepted' status with a runtime of 0 ms. Below this, there are two tabs: 'Case 1' (selected) and 'Case 2'. Under 'Case 1', there are three sections: 'Input', 'Output', and 'Expected'. The 'Input' section shows 'root =' followed by '[3,1,4,null,2]'. The 'Output' section shows '1'. The 'Expected' section shows '1'.

```
</> Code
Java ▾ 🔒 Auto
52 }
53 }
Saved Ln 53, Co

[Testcase] [Test Result]

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input
root =
[3,1,4,null,2]

k =
1

Output
1

Expected
1
```

Problem 9

Aim:

Populating Next Right Pointers in Each Node Code:

```
class Solution {
    public Node connect(Node root) {
        if (root == null) {
            return null;
        }
```

```
        Node leftmost = root;
```

```
        while (leftmost.left != null) {
            Node current = leftmost;
```

```

while (current != null) {
    // Connect left child to right child
    current.left.next = current.right;

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

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

// Move to the next level
leftmost = leftmost.left;
}

return root;
}
}

```

Output:

☒ Testcase
 ☒ Test Result
 runtime: 0 ms

Accepted

Case 1

Case 2

Input

root =
 [1,2,3,4,5,6,7]

Output

[1,#,2,3,#,4,5,6,7,#]

Expected

[1, #, 2, 3, #, 4, 5, 6, 7, #]