Experiment 6

Student Name: Shubham Kumar Sharma **UID:** 22BET10353
Branch: IT **Section/Group:** 22BET_IOT-702/A
Semester: 6th **Date of Performance:** 07/02/25
Subject Name: Advance Programming-II **Subject Code:** 22ITP-367

Problem: 1: Maximum Depth of Binary Tree

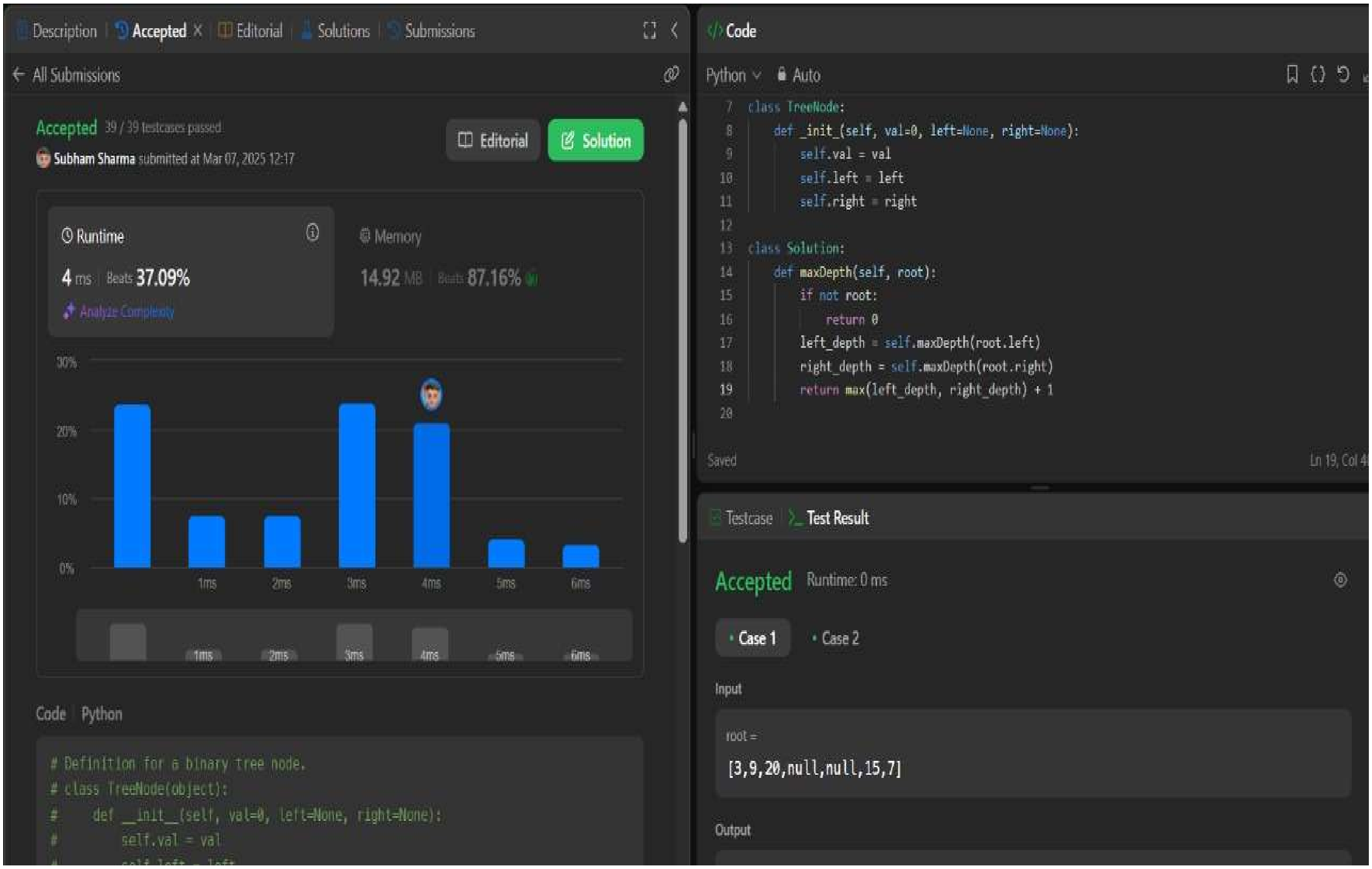
Problem Statement: Given the root of a binary tree, return *its maximum depth*. A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Objective: The goal of this code is to **calculate the maximum depth** of a **binary tree**, which is defined as the **number of nodes along the longest path** from the **root node** down to the **farthest leaf node**.

1. Code:

```
class TreeNode:
    def _init_(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right


class Solution:
    def maxDepth(self, root):
        if not root:
            return 0
        left_depth = self.maxDepth(root.left)
        right_depth = self.maxDepth(root.right)
        return max(left_depth, right_depth) + 1
```

3. Result:



Problem 2: Validate Binary Search Tree

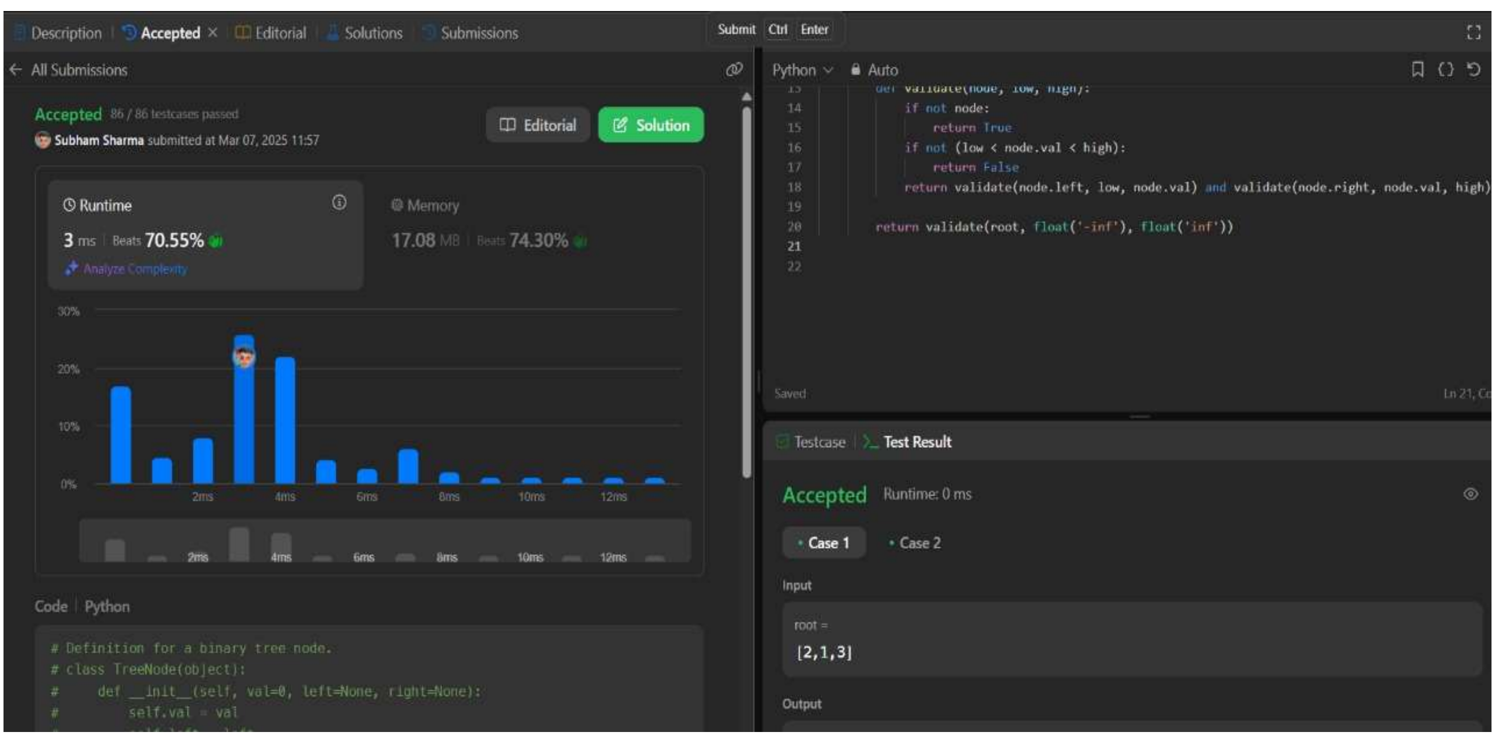
Problem Statement: Given the root of a binary tree, determine if it is a valid binary search tree (BST). A valid BST is defined as follows: The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. Both the left and right subtrees must also be binary search trees.

1. **Objective:** The goal of this code is to **determine whether a given binary tree is a valid Binary Search Tree (BST)**.

2. Code:

```
class Solution(object):
    def isValidBST(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: bool
        """
        def validate(node, low, high):
            if not node:
                return True
            if not (low < node.val < high):
                return False
            return validate(node.left, low,
node.val) and validate(node.right, node.val,
high)

        return validate(root, float('-inf'),
float('inf'))
```

Result



Problem 3: Symmetric Tree

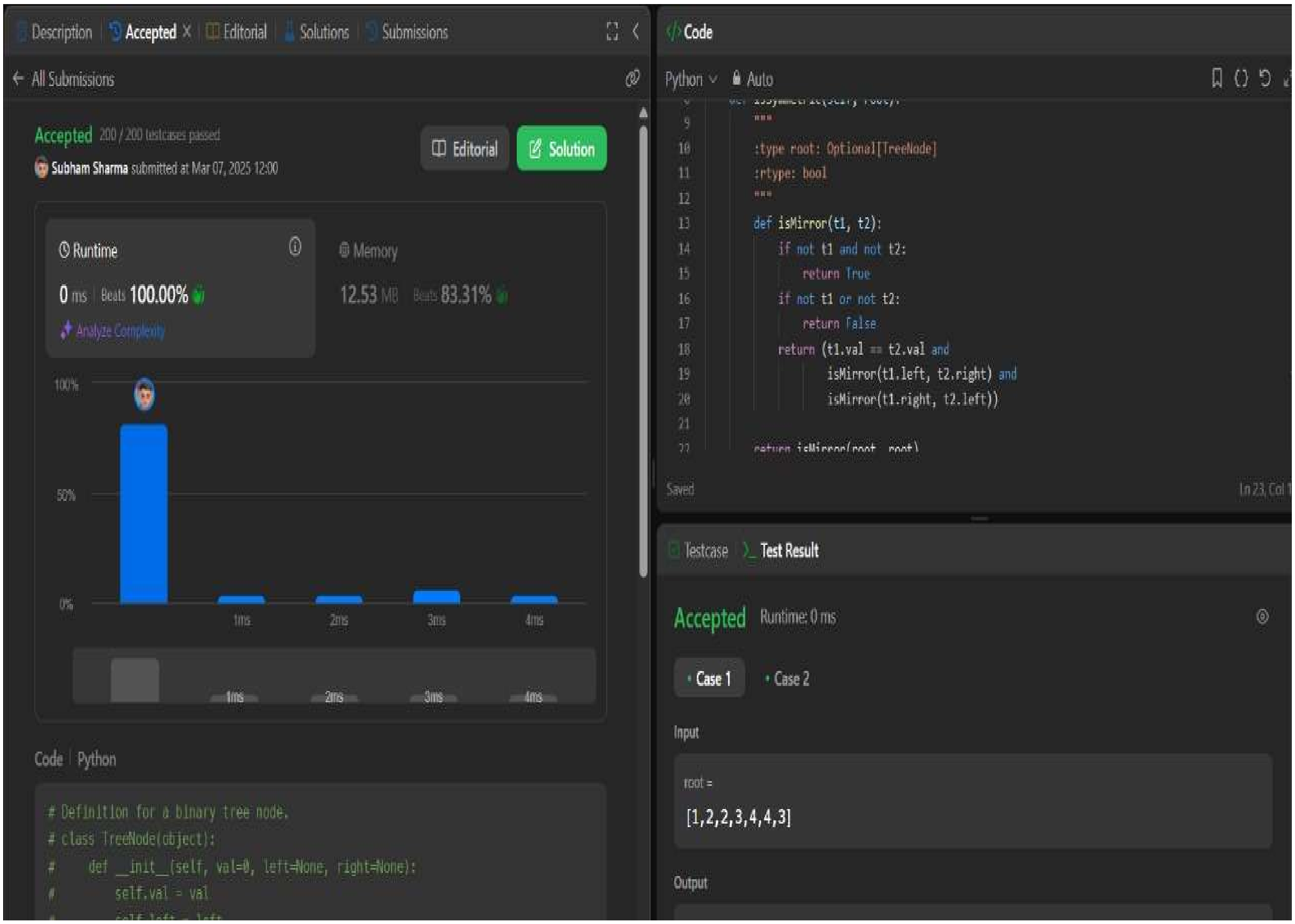
Problem Statement: Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

1. **Objective:** The goal of this code is to **check whether a given binary tree is symmetric (i.e., a mirror of itself around its center)**.

2. **Code:**

```
class Solution(object):
    def isSymmetric(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: bool
        """
        def isMirror(t1, t2):
            if not t1 and not t2:
                return True
            if not t1 or not t2:
                return False
            return (t1.val == t2.val and
                    isMirror(t1.left, t2.right) and
                    isMirror(t1.right, t2.left))

        return isMirror(root, root)
```

3. Result:


Description
Accepted
Editorial
Solutions
Submissions
Code
All Submissions
Python
Auto
Accepted 200 / 200 testcases passed
Subham Sharma submitted at Mar 07, 2025 12:00
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
12.53 MB | Beats 83.31%
100%
50%
0%
1ms
2ms
3ms
4ms
:type root: Optional[TreeNode]
:rtype: bool
def isMirror(t1, t2):
if not t1 and not t2:
return True
if not t1 or not t2:
return False
return (t1.val == t2.val and
isMirror(t1.left, t2.right) and
isMirror(t1.right, t2.left))
Saved
Testcase
Test Result
Accepted Runtime: 0 ms
Case 1
Case 2
Input
root =
[1,2,2,3,4,4,3]
Output
Code | Python
Definition for a binary tree node.
class TreeNode(object):
def __init__(self, val=0, left=None, right=None):
self.val = val

Problem 4: Binary Tree Level Order Traversal

Problem Statement:

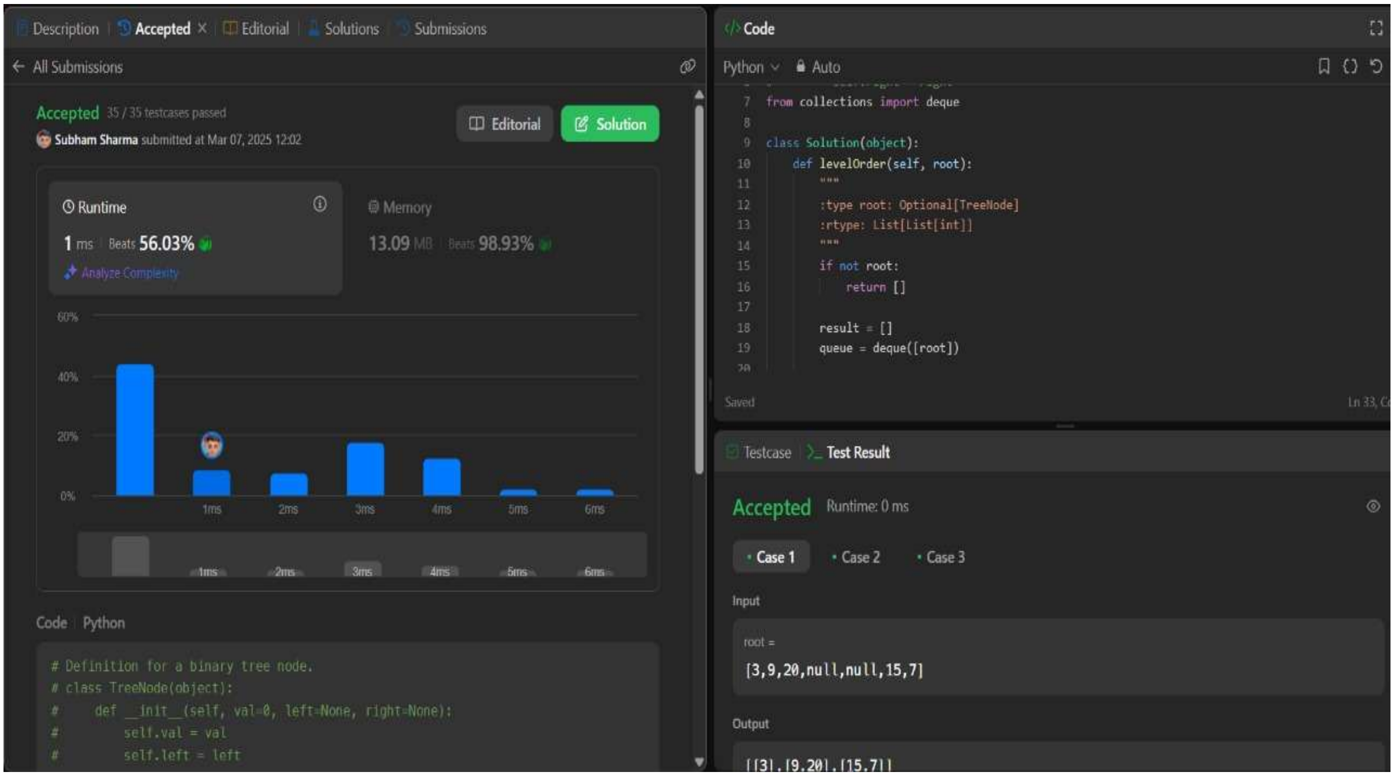
Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Objective: The goal of this code is to **perform a level-order traversal of a binary tree and return the node values in a list of lists, where each sublist represents a level of the tree.**

2. Code:

```
from collections import deque

class Solution(object):
    def levelOrder(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: List[List[int]]
        """
        if not root:
            return []

        result = []
        queue = deque([root])

        while queue:
            level = []
            for _ in range(len(queue)):  # Process nodes at the current level
                node = queue.popleft()
                level.append(node.val)
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)
            result.append(level)

        return result
```

3.Result:



Problem 5: Convert Sorted Array to Binary Search Tree

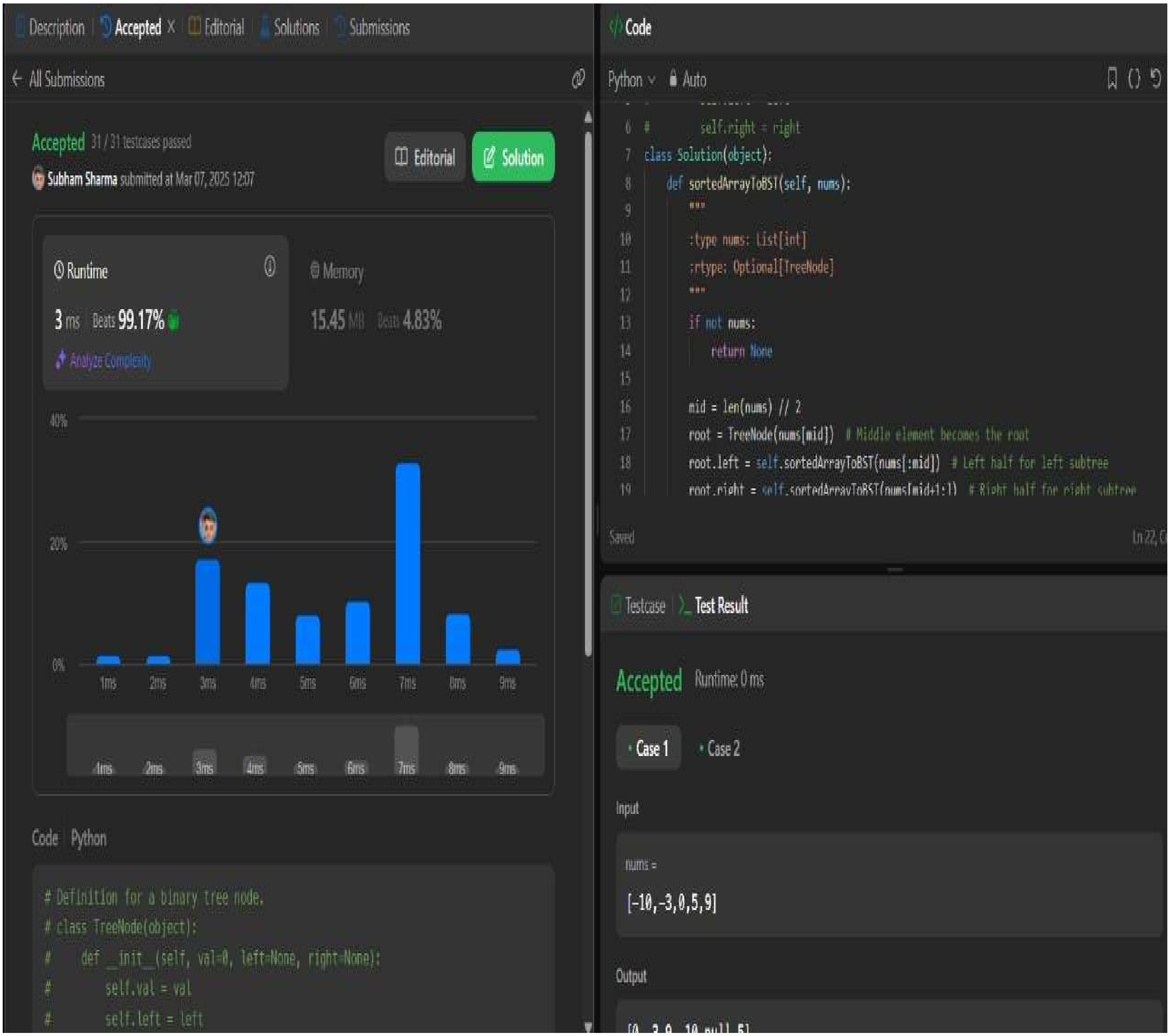
Problem Statement: Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree..

1. **Objective:** The goal of this code is to **convert a sorted integer array into a height-balanced Binary Search Tree (BST)**.

2. **Code:**

```
class Solution(object):
    def sortedArrayToBST(self, nums):
        """
        :type nums: List[int]
        :rtype: Optional[TreeNode]
        """
        if not nums:
            return None

        mid = len(nums) // 2
        root = TreeNode(nums[mid])  # Middle element becomes the root
        root.left = self.sortedArrayToBST(nums[:mid])  # Left half for left subtree
        root.right = self.sortedArrayToBST(nums[mid+1:])  # Right half for right subtree

        return root
```

3. Result:


Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 31 / 31 testcases passed
Subham Sharma submitted at Mar 07, 2025 12:07
Editorial
Solution
Runtime
3 ms Beats 99.17%
Analyze Complexity
Memory
15.45 MB Beats 4.83%
Code Python
Definition for a binary tree node.
class TreeNode(object):
def __init__(self, val=0, left=None, right=None):
self.val = val
self.left = left
Code
Python
Auto
# self.right = right
class Solution(object):
def sortedArrayToBST(self, nums):
"""
:type nums: List[int]
:rtype: Optional[TreeNode]
"""
if not nums:
return None
mid = len(nums) // 2
root = TreeNode(nums[mid]) # Middle element becomes the root
root.left = self.sortedArrayToBST(nums[:mid]) # Left half for left subtree
Saved
Testcase
Test Result
Accepted Runtime: 0 ms
Case 1
Case 2
Input
nums =
[-10,-3,0,5,9]
Output

Problem 6: Binary Tree Inorder Traversal

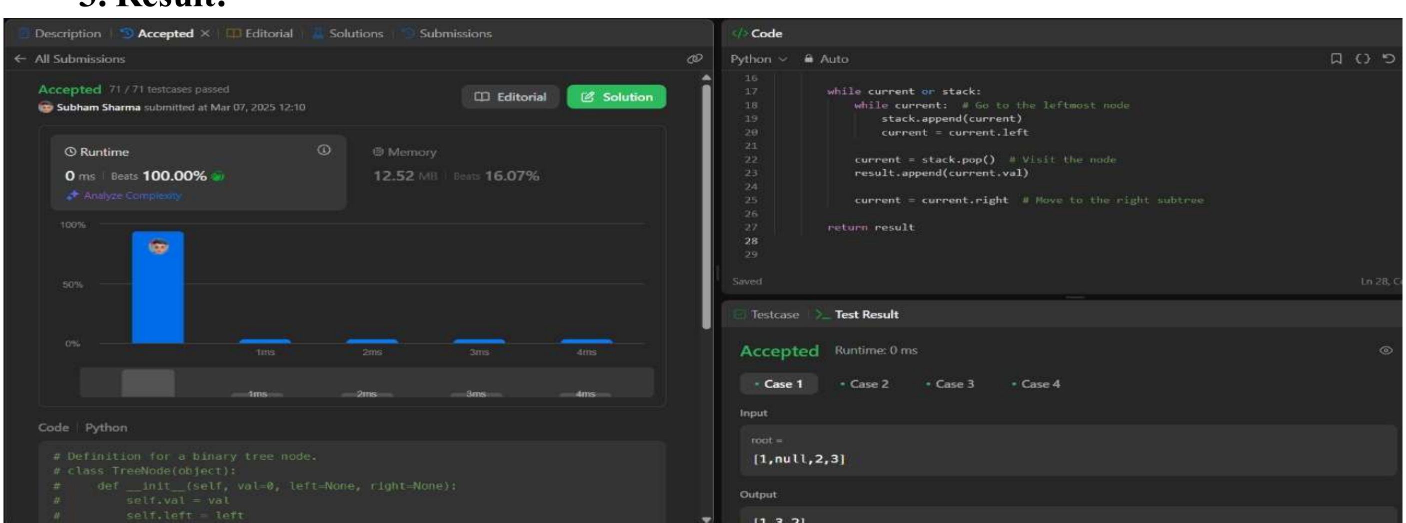
Problem Statement: Given the root of a binary tree, return the inorder traversal of its nodes' values.

1. **Objective:** The goal of this code is to **return the inorder traversal of a binary tree's node values**.

1. **Code:**

```
class Solution(object):
    def inorderTraversal(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: List[int]
        """
        result = []
        stack = []
        current = root

        while current or stack:
            while current:  # Go to the leftmost node
                stack.append(current)
                current = current.left

            current = stack.pop()  # Visit the node
            result.append(current.val)

            current = current.right  # Move to the right subtree

        return result
```

3. Result:



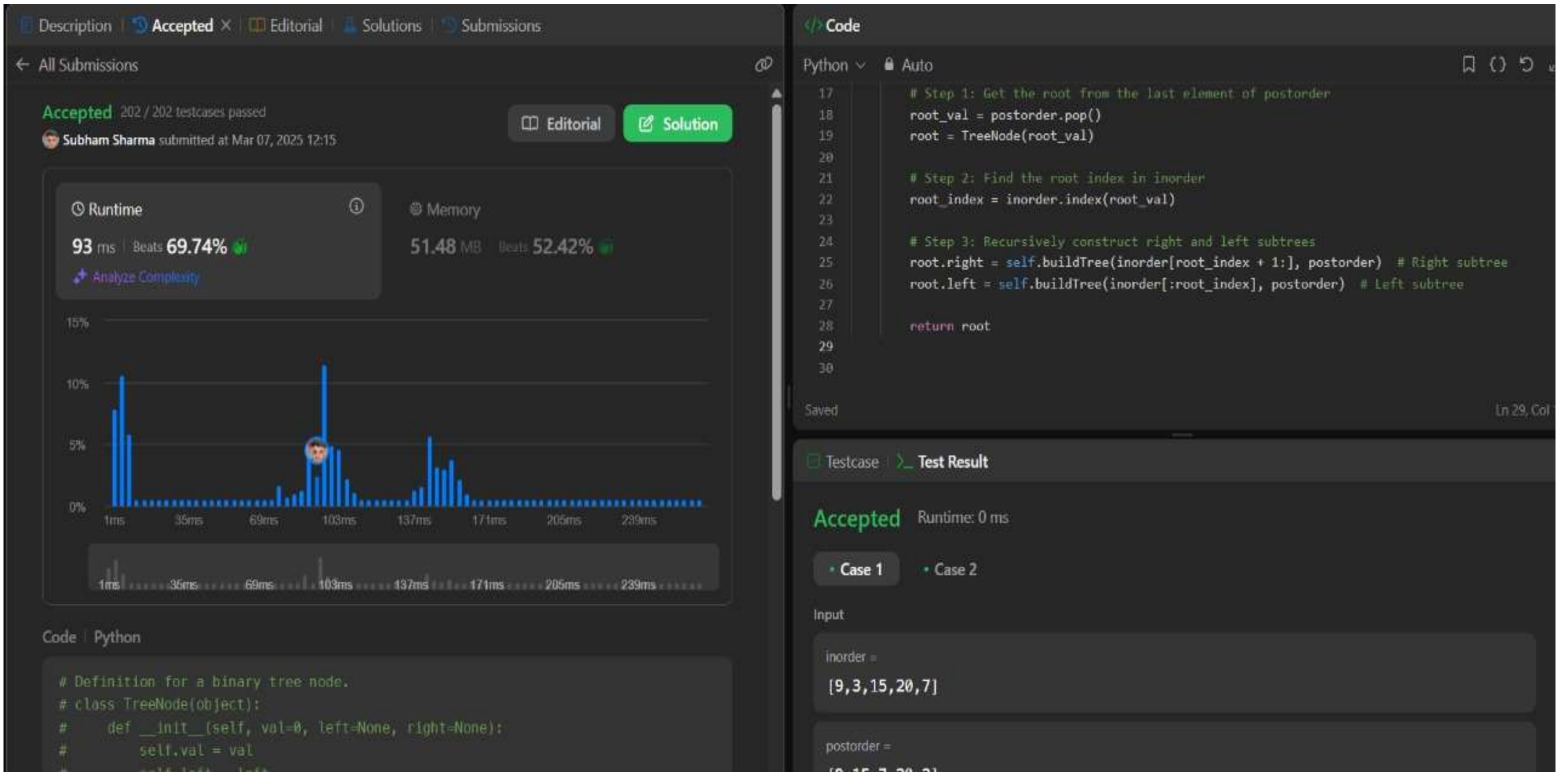
Problem 7: Construct Binary Tree from Inorder and Postorder Traversal

Problem Statement: 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.

1. **Objective:** The goal of this code is to **construct a binary tree** given its **inorder and postorder traversal arrays**.

1. **Code**

```
class Solution(object):
    def buildTree(self, inorder, postorder):
        """
        :type inorder: List[int]
        :type postorder: List[int]
        :rtype: Optional[TreeNode]
        """
        if not inorder or not postorder:
            return None

        # Step 1: Get the root from the last element of postorder
        root_val = postorder.pop()
        root = TreeNode(root_val)

        # Step 2: Find the root index in inorder
        root_index = inorder.index(root_val)

        # Step 3: Recursively construct right and left subtrees
        root.right = self.buildTree(inorder[root_index + 1:], postorder)  # Right subtree
        root.left = self.buildTree(inorder[:root_index], postorder)  # Left subtree

        return root
```



Problem 8: Kth Smallest element in a BST

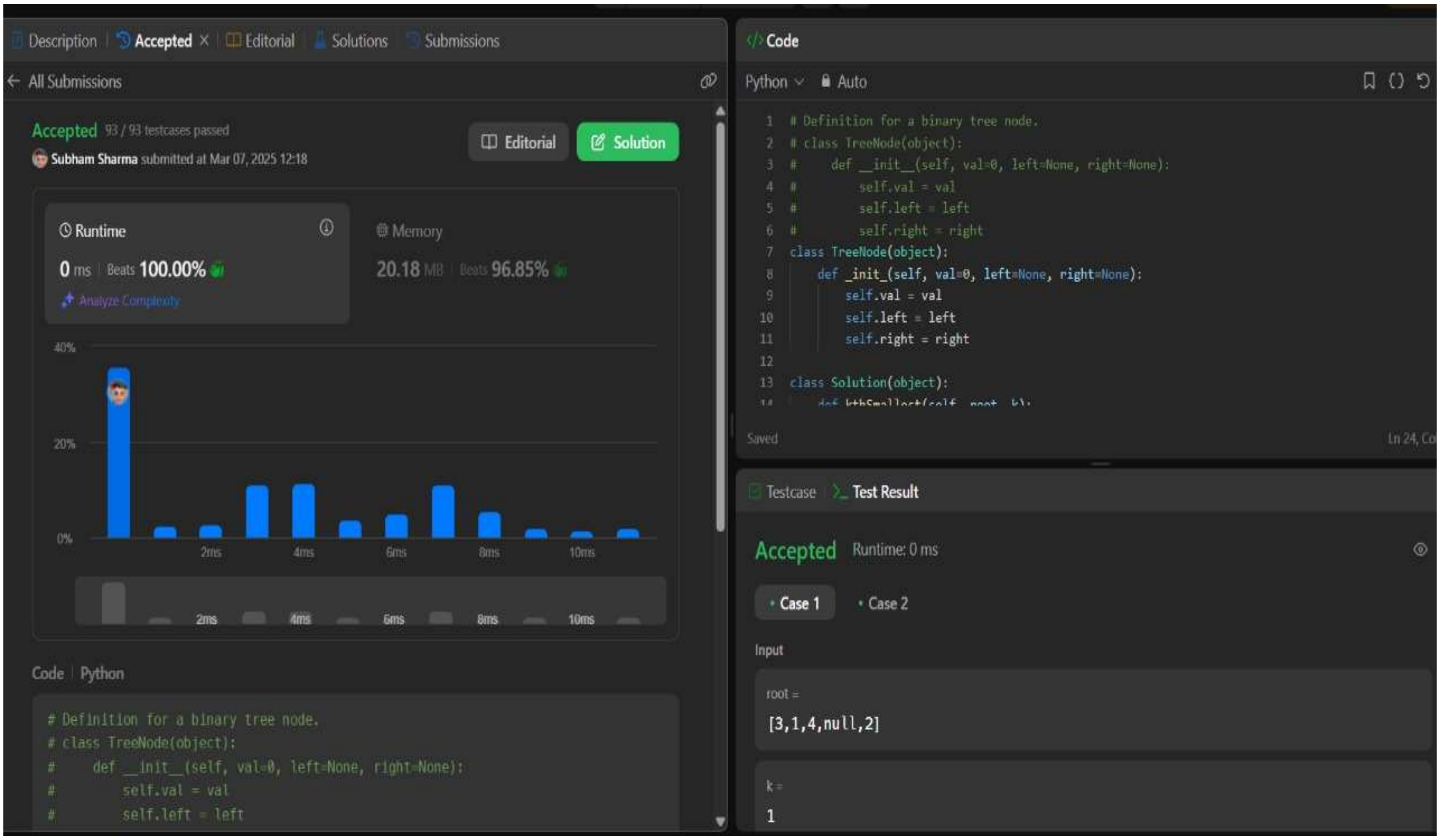
Problem Statement: Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) of all the values of the nodes in the tree.

1. **Objective:** The goal of this code is to **find the kth smallest value (1-indexed) in a Binary Search Tree (BST).**

2. **Code**

```
class TreeNode(object):
    def _init_(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution(object):
    def kthSmallest(self, root, k):
        stack = []
        while True:
            while root:
                stack.append(root)
                root = root.left
            root = stack.pop()
            k -= 1
            if k == 0:
                return root.val
            root = root.right
```

Result



Problem 9: Populating Next Right Pointers in Each Node

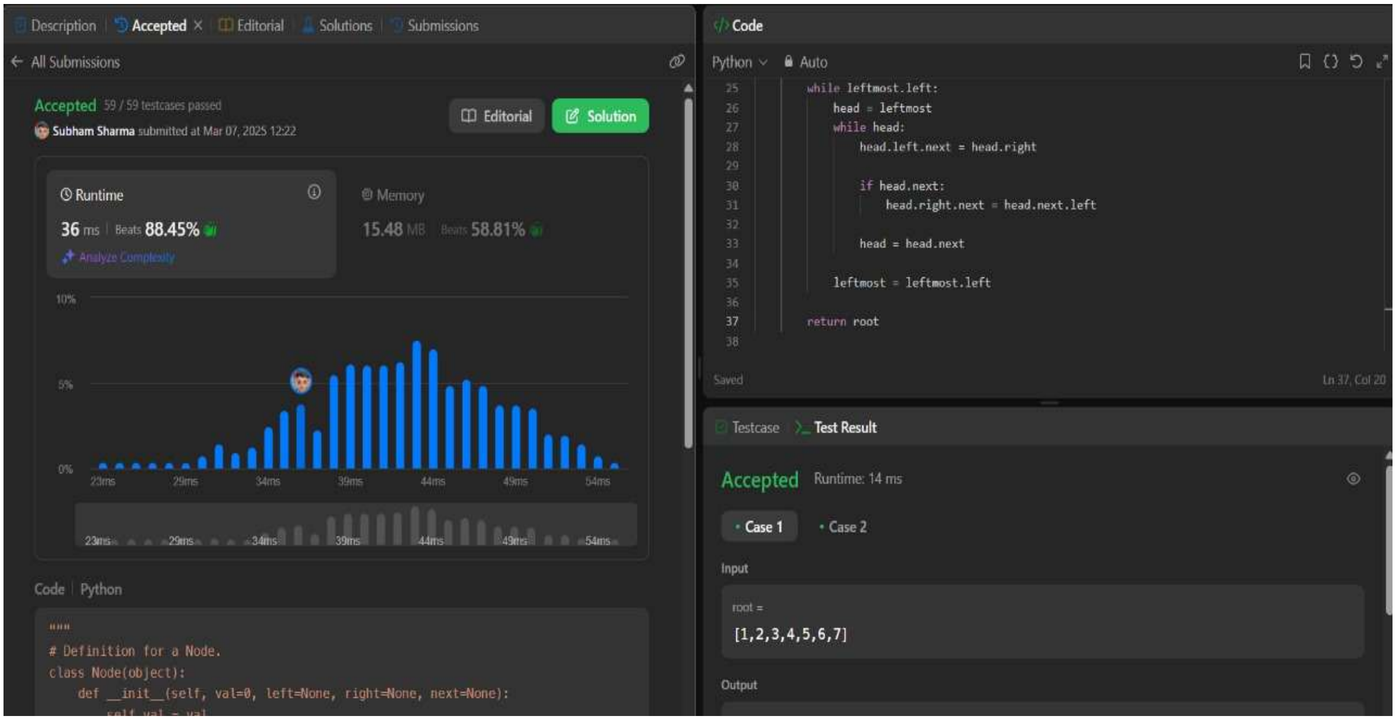
Problem Statement: You are given a perfect binary tree where all leaves are on the same level, and every parent has two children. The binary tree has the following definition: struct Node { int val; Node *left; Node *right; Node *next; } Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL. Initially, all next pointers are set to NULL.

1. **Objective:** You are given a perfect binary tree where all leaves are on the same level, and every parent has two children. The binary tree has the following definition: struct Node { int val; Node *left; Node *right; Node *next; } 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. Initially, all next pointers are set to NULL.

Code

```
class Node(object):
    def _init_(self, val=0, left=None, right=None, next=None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next

class Solution(object):
    def connect(self, root):
        if not root:
            return None

        leftmost = root

        while leftmost.left:
            head = leftmost
            while head:
                head.left.next = head.right

                if head.next:
                    head.right.next = head.next.left

                head = head.next

            leftmost = leftmost.left

        return root
```

RESULT



Learning Outcomes:

- Implementing recursive depth-first search (DFS).
- Understanding the BST property (left subtree < root < right subtree).
- Understanding tree mirroring and level-based traversal.
- Using queues for level-by-level traversal.
- Understanding balanced BSTs and why they are efficient.
- Traversing a perfect binary tree level by level. Using constant space traversal (without extra memory).