**WORKSHEET 5**

**Student Name: ANMOLPREET SINGH UID:** 22BCS15759

**Branch:** BE-CSE **Section/Group:** 22BCS\_NTPP-602-A

**Semester:** 6th **Date of Performance:** 17/02/2025

**Subject Name:** AP LAB - II **Subject Code:** 22CSP-351

1. **Aim:** 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.

1. **Source Code:**

class Solution:

def maxDepth(self, root: Optional[TreeNode]) -> int: if not root:

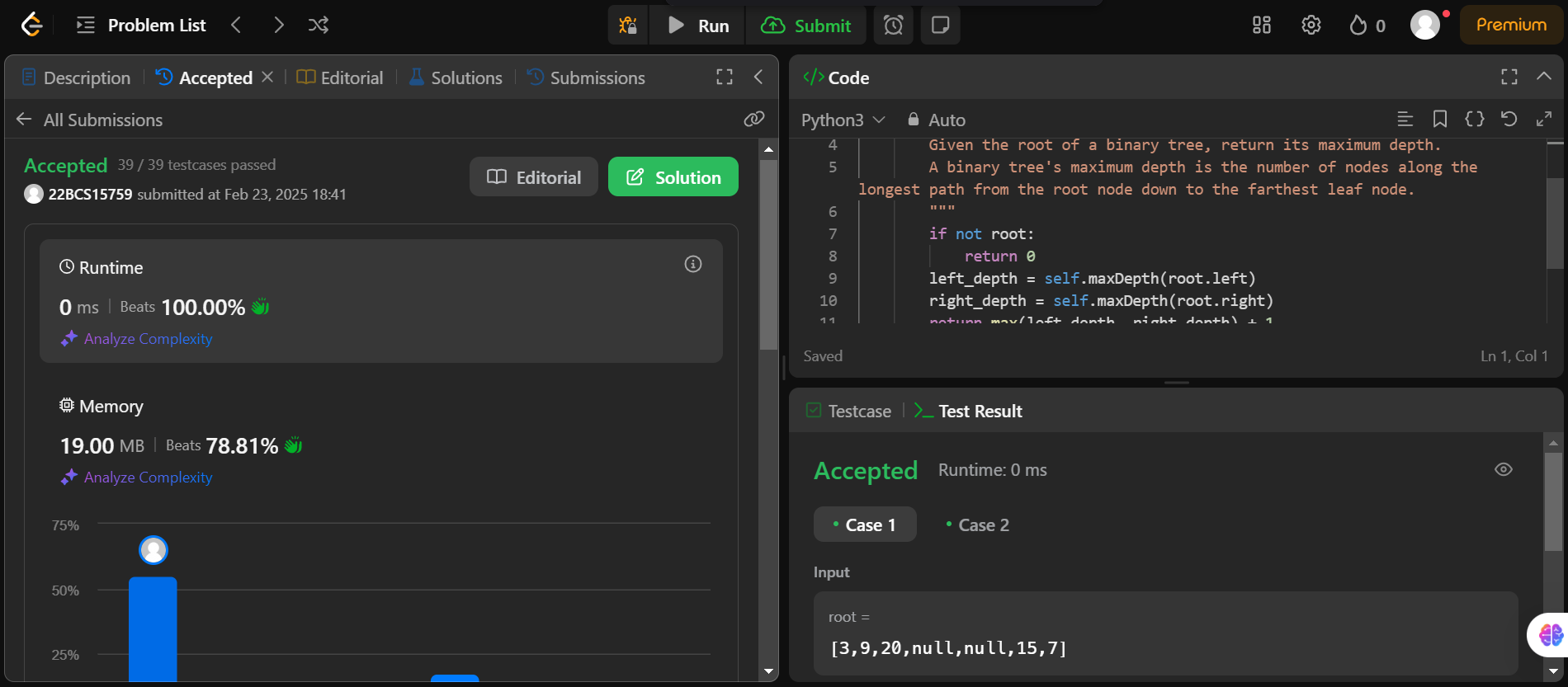
return 0

return 1 + max(self.maxDepth(root.left), self.maxDepth(root.right))

class TreeNode:

def init (self, val=0, left=None, right=None): self.val = val

self.left = left self.right = right

1. **Screenshots of outputs:** 

**2.**

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

**Source Code:**

class Solution:

def isSymmetric(self, root: Optional[TreeNode]) -> bool:

def isMirror(t1: Optional[TreeNode], t2: Optional[TreeNode]) -> bool: if not t1 and not t2:

return True

if not t1 or not t2: return False

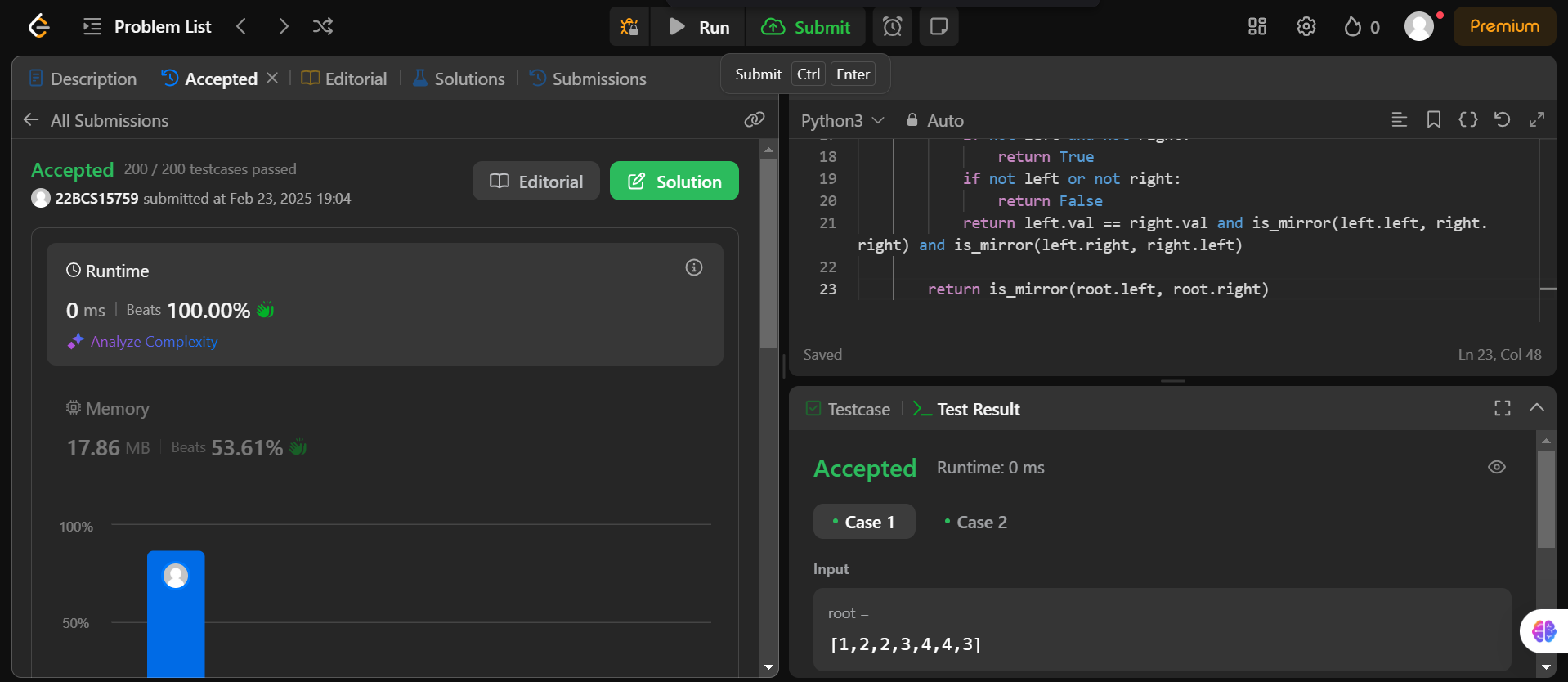
return (t1.val == t2.val and isMirror(t1.left, t2.right) and isMirror(t1.right, t2.left))

return isMirror(root, root) class TreeNode:

def init (self, val=0, left=None, right=None): self.val = val

self.left = left self.right = right

**Screenshots of outputs:**

****

**3.**

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

**Source Code:**

class TreeNode:

def init (self, val=0, left=None, right=None): self.val = val

self.left = left self.right = right

class Solution:

def buildTree(self, preorder: List[int], inorder: List[int]) -> Optional[TreeNode]: inorder\_map = {value: idx for idx, value in enumerate(inorder)}

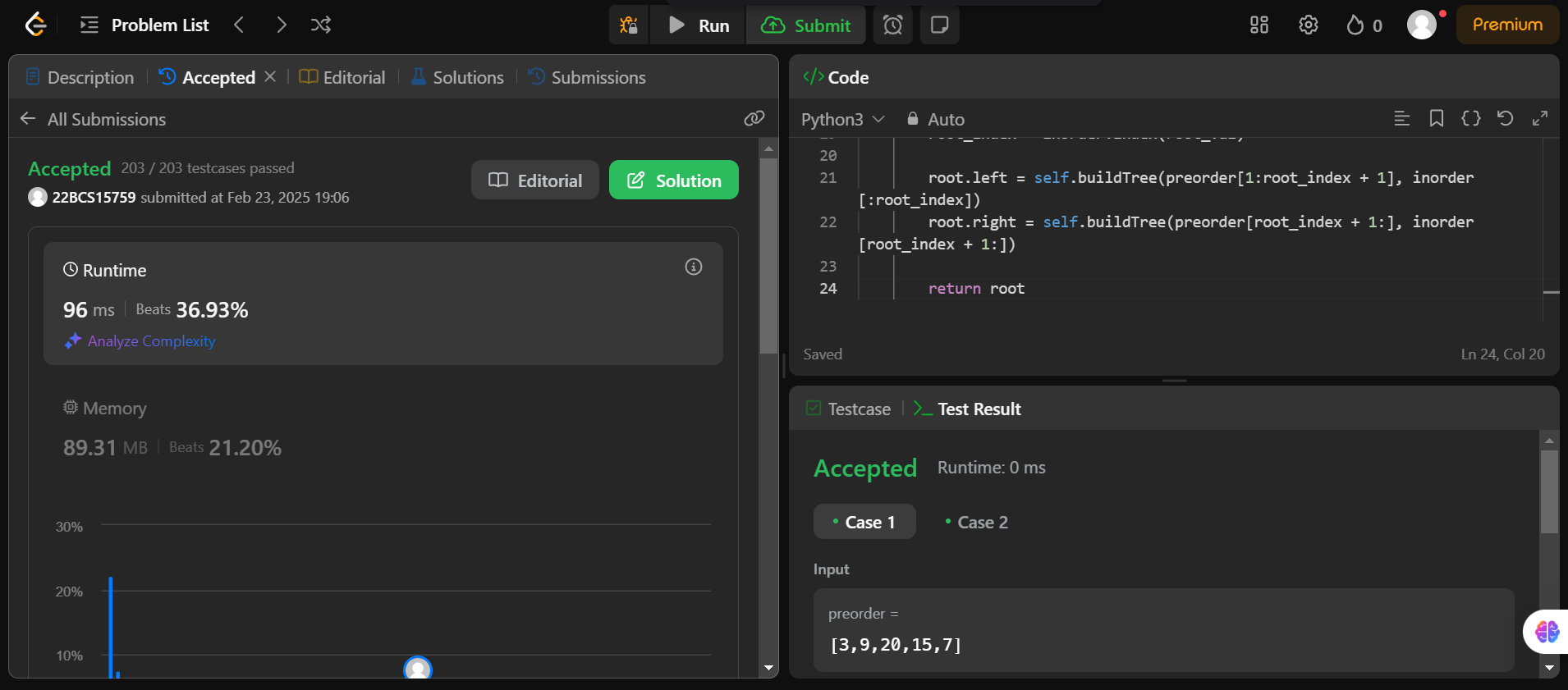
return self.\_buildTreeHelper(preorder, 0, len(preorder) - 1, inorder, 0, len(inorder) - 1, inorder\_map)

def \_buildTreeHelper(self, preorder: List[int], pre\_start: int, pre\_end: int, inorder: List[int], in\_start: int, in\_end: int, inorder\_map: dict) -> Optional[TreeNode]:

if pre\_start > pre\_end or in\_start > in\_end:

return None

1. **Screenshots of outputs:**

****