Experiment 6

Student Name: Akshant Kumar **UID:22BCS13418**

Branch: CSE **Section/Group: NTPP-IOT627/A**

Semester: 6th **Date of Performance:20-3-25**

Subject Name: AP Lab-2 **Subject Code:** 22CSP-351

Problem -1

1. **Aim:** Binary tree level order transversal

2. **Objective:** LeetCode Problem: Binary Tree Level Order Traversal Problem Statement: Given the root of a binary tree, return its level order traversal (i.e., from left to right, level by level). Initialize an empty result list result and a queue queue containing the root node.While the queue is not empty:Get the current level's node count (level_size). Create an empty list level_nodes to store nodes at this level. Process all nodes in the current level: Remove the front node from the queue. Add its value to level_nodes. Add its left and right children (if they exist) to the queue. Append level_nodes to result. Return result.

3. **Implementation/Code:**

```
from collections import deque
def levelOrder(root):
    if not root:
        return [
    result = []
    queue = deque([root])
    while queue:
        level_size = len(queue)
        level_nodes = []
        for _ in range(level_size):
            node = queue.popleft()
            level_nodes.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        result.append(level_nodes)
    return result
```

Output:

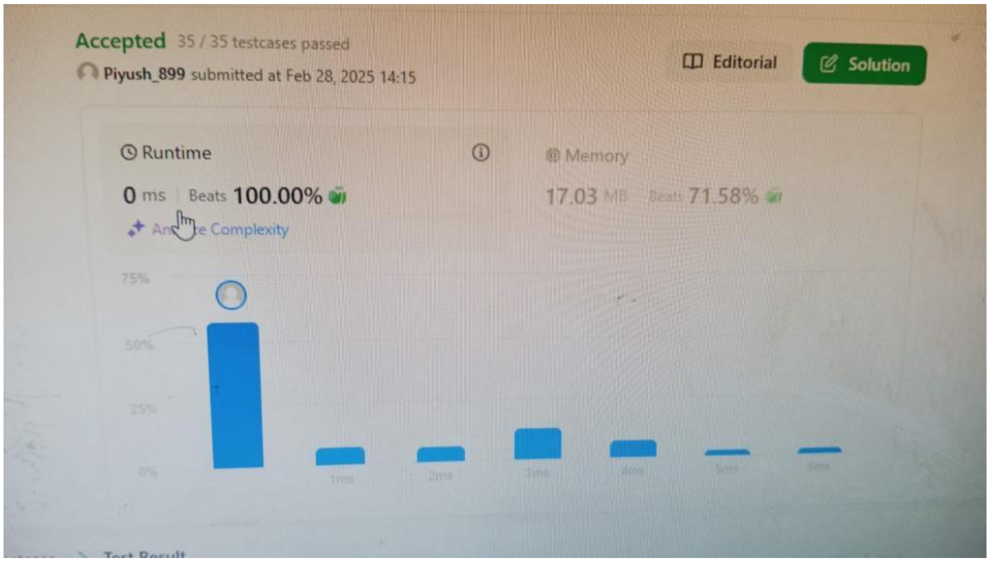

Figure 1

4. Learning Outcome:

- **Learn how BFS works using a queue to traverse a tree level by level.**
- **Queue Data Structure Usage:Understand how to use a queue (deque in Python) to maintain nodes at each level.**
- **Tree Traversal Techniques:Learn about different ways to traverse a tree, including level order, depth-first (preorder, inorder, postorder), and zigzag order.**
- **Handling Edge Cases in Trees:**
- **Learn how to handle edge cases like an empty tree, single-node tree, or trees with only left/right children.**

<u>Problem-2</u>

1. Aim: Sort Maximum depth of a binary tree

2. Implementation/Code:

```
  int maxDepth(TreeNode* root)
  {
     if(root==NULL)
        return 0;
     return max(1+maxDepth(root->left), 1+maxDepth(root->right));

  }
};
```

4. Output:

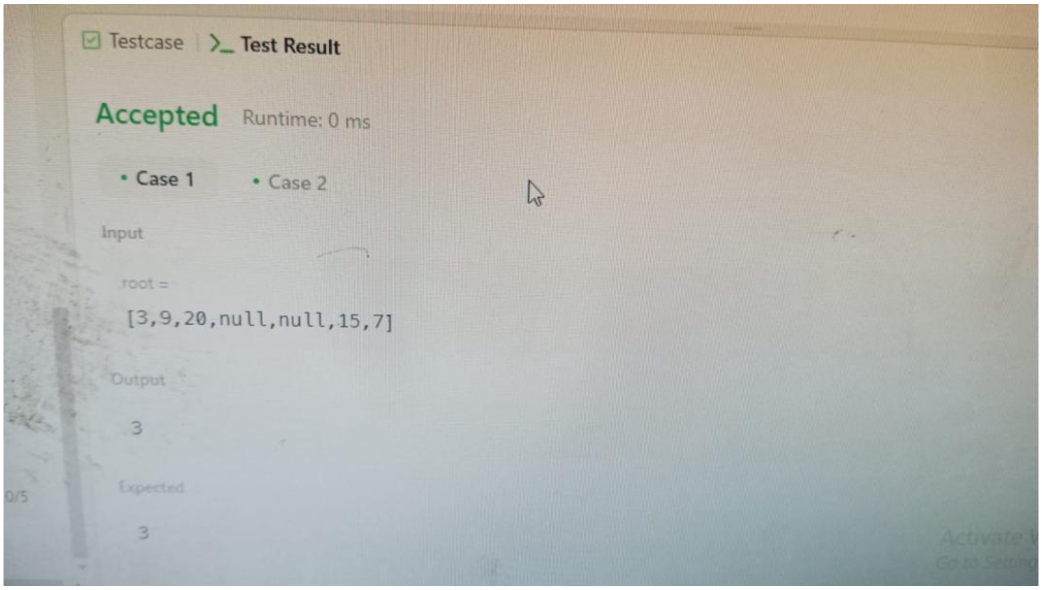

Figure 2

- **<u>Learning Outcomes:</u>**

- Understand the concept of maximum depth in a binary tree.
- Implement recursive and iterative approaches to find the maximum depth.
- Analyze the time and space complexity of different approaches.
- Apply depth-first search (DFS) and breadth-first search (BFS) to tree traversal.
- Develop problem-solving skills related to binary tree structures.

Problem – 3

1. **Aim:** Symmetric tree

Implementation/Code:

```
class Solution {
public:
    bool areMirrImg(TreeNode* root1, TreeNode* root2){
        if(!root1 && !root2){
            return true;
        }
        if(!root1 || !root2){
            return false;
        }
        return (root1->val == root2->val) && (areMirrImg(root1->left,root2->right)) && (areMirrImg(root1->right,root2->left));
    }
    bool isSymmetric(TreeNode* root) {
        if(!root){
            return true;
        }
        return areMirrImg(root->left,root->right);
    }
};
```

Output:

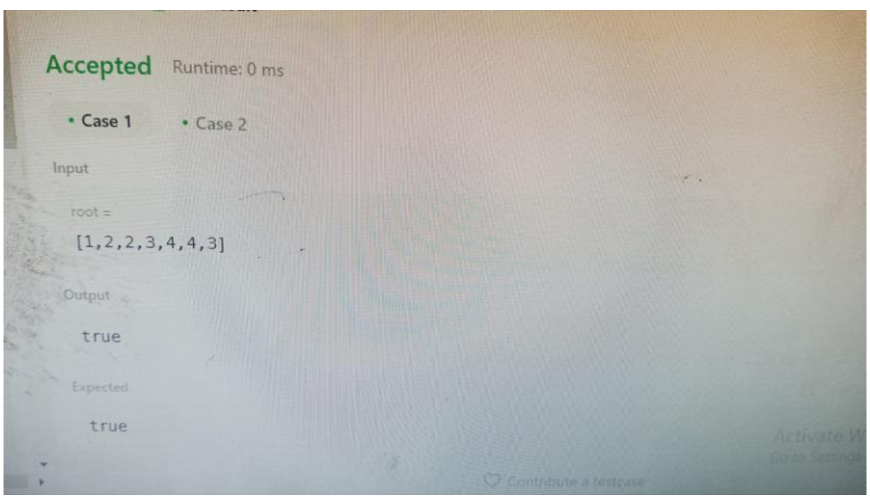

Figure 3

2. Learning Outcomes

- Understand the concept of symmetric trees in binary tree structures.
- Implement a recursive and iterative approach to check tree symmetry.
- Analyze time and space complexity of symmetry-checking algorithms.
- Apply tree traversal techniques (BFS/DFS) for symmetry verification.
- Strengthen problem-solving skills related to binary trees.