# Experiment 6

**StudentName: Shubham** **UID:**22BCS15853
**Branch:** CSE **Section/Group:** KRG 2B
**Semester:** 6th **Date of Performance:**28/02/25
**Subject Name:** Advanced Programming Lab-2 **Subject Code:** 22CSP-351

## Problem -1

1. **Aim:** Maximum depth of binary tree
2. **Objective:**To determine the maximum depth (or height) of a binary tree.
3. **Implementation/Code:**

```
class Solution {
public:
   int maxDepth(TreeNode* root) {

      if(root==NULL)
      return NULL;
      int maxleft=maxDepth(root->left);
      int maxright=maxDepth(root->right);
      return max(maxleft,maxright)+1;

   }
};
```

4. **Output:**

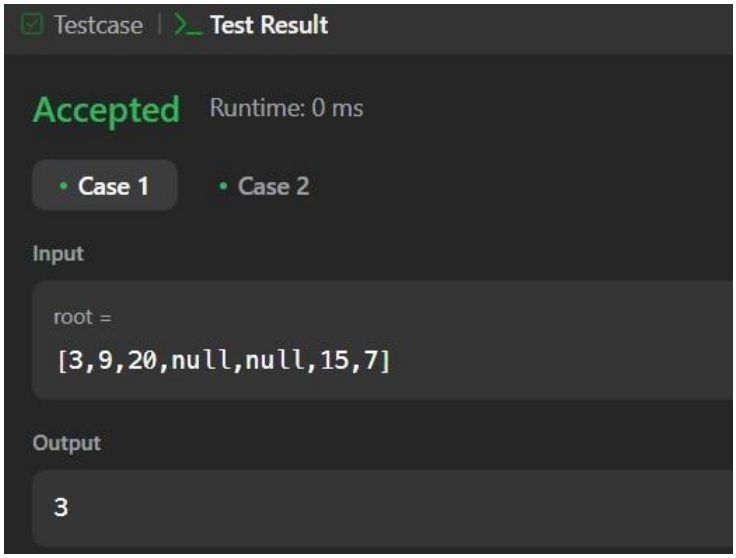


**Figure 1**

5. **Learning Outcome:**

- **Understanding Recursion in Binary Trees** – Learn how recursive functions traverse and process tree nodes efficiently.
- **Computing Maximum Depth** – Understand how to determine the longest path from the root to a leaf node.
- **Optimized Tree Traversal (O(n) Complexity)** – Recognize how the function visits each node once, ensuring efficient computation.

## Problem-2

1. **Aim:** Symmetric tree
2. **Objectives:**To determine whether a given binary tree is symmetric around its center (i.e., a mirror image of itself).

3. **Implementation/Code:**

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return ismirror(root->left,root->right);
    }
    private:
    bool ismirror(TreeNode*n1,TreeNode*n2){
        if(n1==nullptr && n2==nullptr){
            return true;
        }
        if(n1==nullptr || n2==nullptr){
            return false;
        }
        return n1->val==n2->val && ismirror(n1->left,n2->right)&& ismirror(n1-
>right,n2->left);
    }
};
```

4. Output:

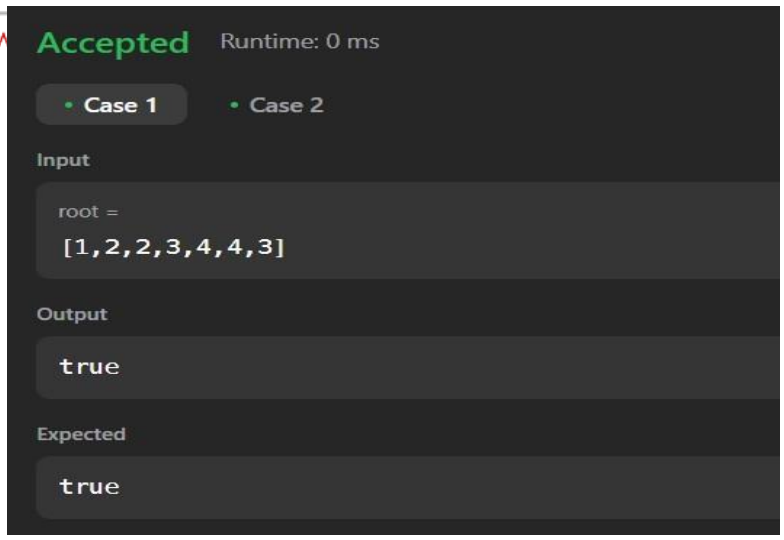


**Figure 2**

## 5. Learning Outcomes:

1. **Understanding Recursion in Tree Comparison** – Learn how recursion is used to compare nodes in a mirrored fashion.
2. **Identifying Symmetric Trees** – Understand how to check if two subtrees are structurally identical and have the same values.
3. **Handling Edge Cases** – Learn to correctly handle cases where subtrees are nullptr (empty), ensuring the function returns the correct result.