Experiment 1.2

Student Name: Nikhil Sharma **UID:**22BCS15209
Branch:CSE **Section/Group:**640/B
Semester: 6 **Date of Performance:** 17/03/25
Subject Name:Advance Programming -2 **Subject Code:** 22CSH-351

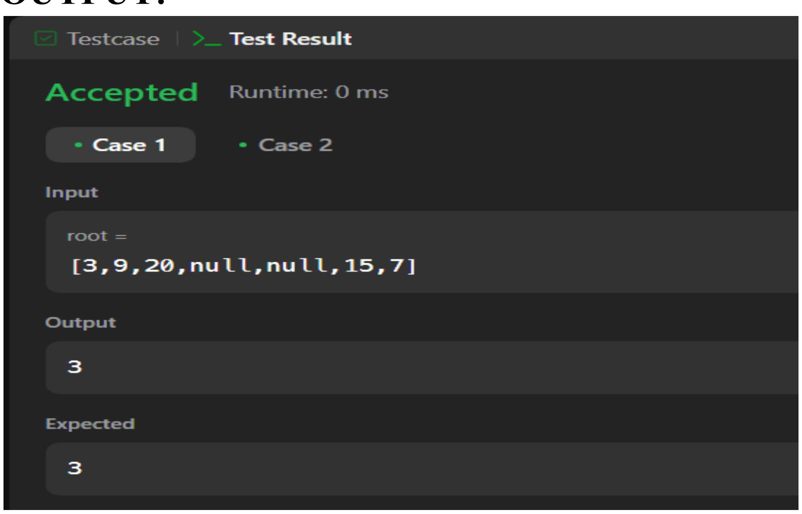
Aim 1: Given the root of a binary tree, return its maximum depth.

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.

Code:

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(!root) return 0;
        int left=maxDepth(root->left);
        int right=maxDepth(root->right);
        return 1 + max(left,right);
    }
};
```

OUTPUT:



Aim 2: 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.

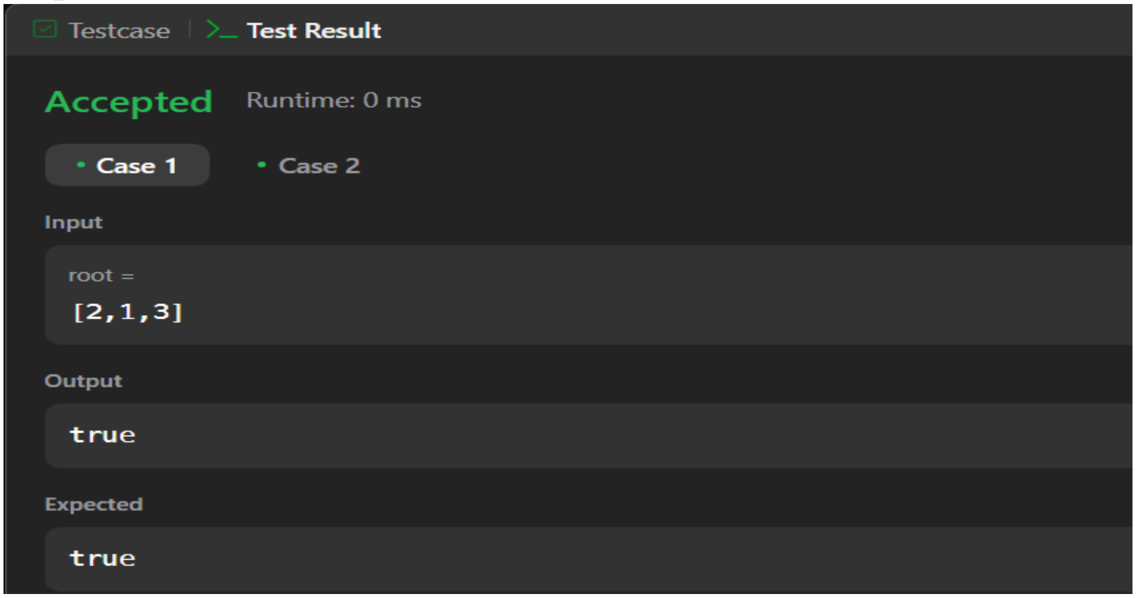
Code:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    void inorder(TreeNode* root,vector<long long>&ans){
    if(root == nullptr){
        return;
    }
    inorder(root->left,ans);
    ans.push_back(root->val);
    inorder(root->right,ans);

    }

    bool isValidBST(TreeNode* root) {
        vector<long long>ans;
    inorder(root,ans);
    if(ans.size()==1){
        return true;
    }
    for(int i =1;i<ans.size();i++){
        if(ans[i]-ans[i-1] <= 0){
            return false;
        }
    }
    return true;
    }
};
```

Output:



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

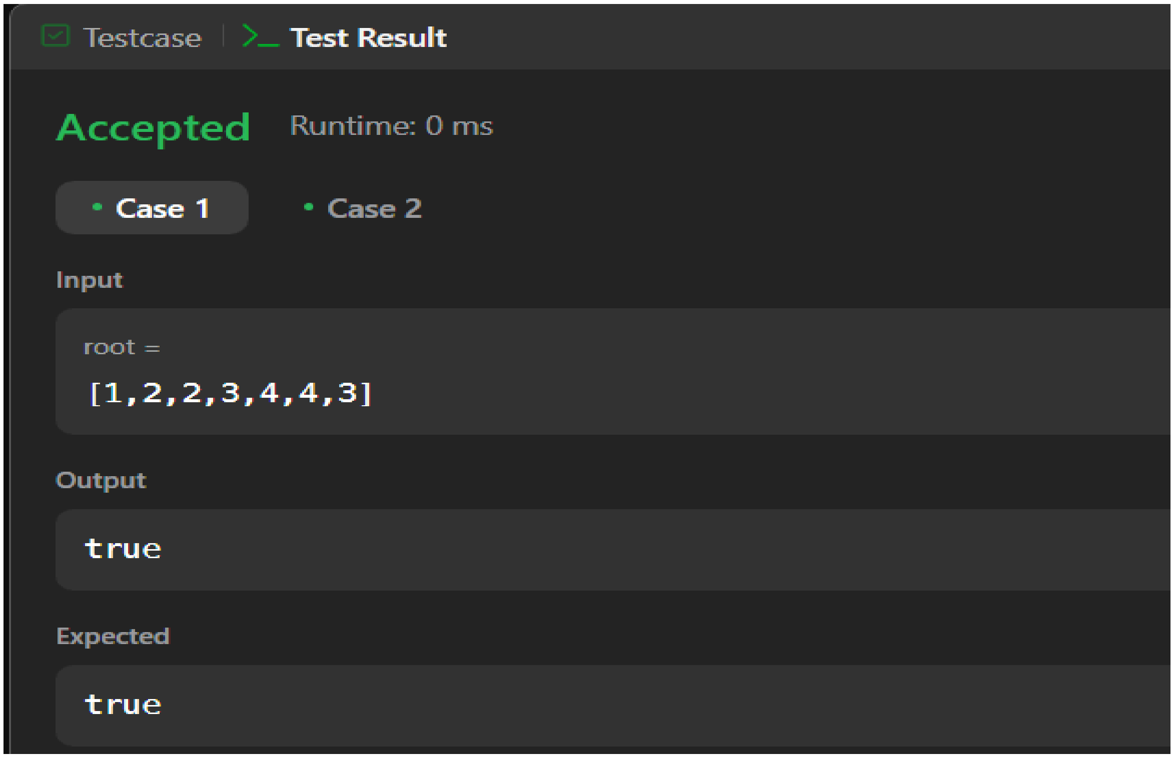
code -
```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        return isMirror(root->left, root->right);
    }

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);
    }
};
```

output :



Learning Outcomes:

1. Understanding Tree Structure and Operations
2. Implementing Binary Trees and Variants
3. Analyzing Tree-Based Algorithms
4. Applying Trees to Real-World Problems