Experiment 3 A

Student Name: Karanvir Singh	**UID: 22BCS16269**
Branch: CSE	**Section/Group: Ntpp 602-A**
Semester: 6TH	**Date of Performance:03/02/25**
Subject Name: AP Lab-2	**Subject Code: 22CSH-352**

1. TITLE:

Maximum Depth of Binary Tree

2. 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.

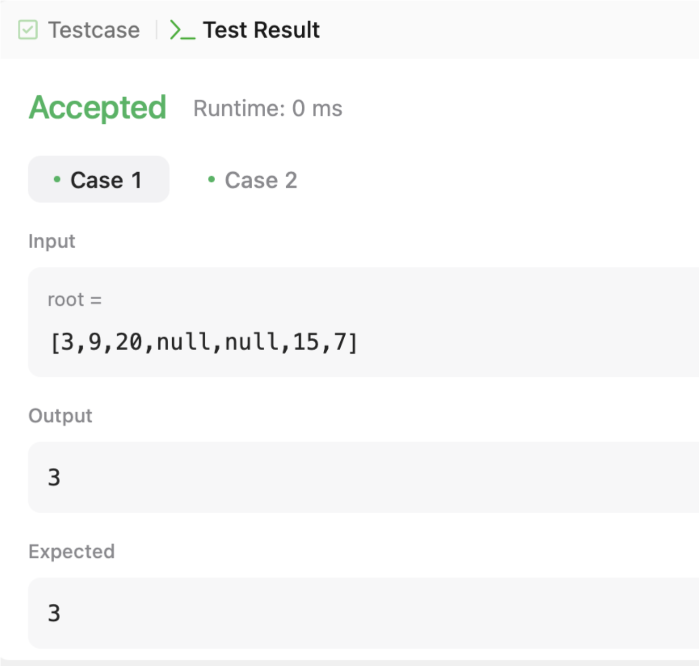
3. Algorithm

- Start DFS with the root node at depth 0.
- If the node is null, return the current depth.
- Recursively explore left and right children, increasing depth by 1.
- Return the maximum depth from left or right subtree.

Implemetation/Code

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

Output



Accepted Runtime: 0 ms

Input

```
root =
[3,9,20,null,null,15,7]
```

Output

```
3
```

Expected

```
3
```

Time Complexity : O(n)

Space Complexity : O(h)

Learning Outcomes:-

- Learn how to traverse trees using depth-first search.
- Learn how to determine the height or depth of binary trees.

Experiment 3 B

Student Name: Karanvir Singh **UID: 22BCS1269**

Branch: CSE **Section/Group: Ntpp 602-A**

Semester: 6TH **Date of Performance:03/02/25**

Subject Name: AP Lab-2 **Subject Code: 22CSH-352**

1. TITLE:

Binary Tree Level Order Traversal

2. AIM:

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).

3. Algorithm

- Initialize an empty result list ans and a queue q with the root node.
- While the queue is not empty, repeat the following steps:
- Initialize an empty list currLevel for the current level.
- Process each node in the queue: dequeue the node, add its value to currLevel, and enqueue its left and right children if they exist.
- After processing all nodes at the current level, append currLevel to ans.
- Return ans as the result containing all levels of the tree.

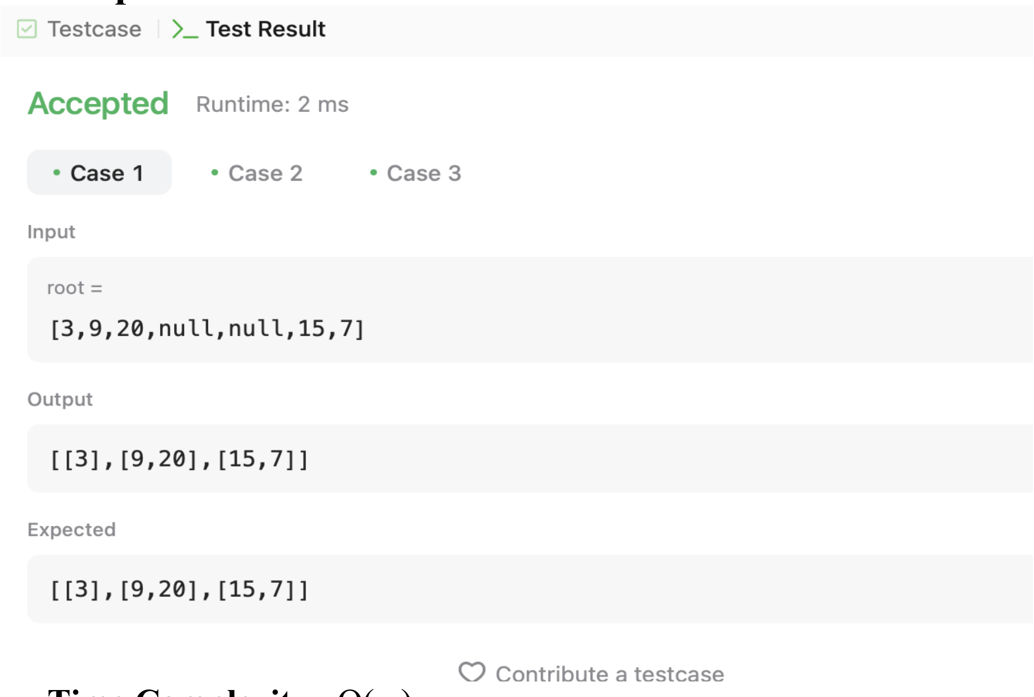
Implemetation/Code:

```
class Solution {
public:
vector<vector<int>> levelOrder(TreeNode* root) {
if (root == nullptr)
return {};
vector<vector<int>> ans;
queue<TreeNode*> q{{root}};

while (!q.empty()) {
vector<int> currLevel;
for (int sz = q.size(); sz > 0; --sz) {
TreeNode* node = q.front();
q.pop();
currLevel.push_back(node->val);
if (node->left)
q.push(node->left);
if (node->right)
q.push(node->right);
}
ans.push_back(currLevel);
}

return ans;
}
};
```

Output



Accepted Runtime: 2 ms

Input

root =

```
[3,9,20,null,null,15,7]
```

Output

```
[[3],[9,20],[15,7]]
```

Expected

```
[[3],[9,20],[15,7]]
```

Time Complexity : O(n)

Space Complexity : O(h)

Learning Outcomes:-

- Learn how to perform and apply in-order traversal in binary trees to solve problems.
- Python generators to manage state and produce results on demand during tree traversal.