

## Experiment 6

**Name:** Ujjwal kumar

**UID:** 22BCS14185

**Branch:** CSE

**Section/Group:** 640`B

**Semester:** 6

**Date of Performance:**10-03-25

**Subject Name:** AP LAB-II

**Subject Code:** 22CSP-351

### 1. Aim:

- To find and implement the maximum depth of Binary Tree.
- To develop an algorithm for Binary Tree Inorder traversal.

### 2. Objective:

- To implement and analyze maximum depth of Binary Tree.
- To develop an algorithm for Binary Tree Inorder traversal.

### 3. Implementation/Code:

```
A. class Solution {  
public int maxDepth(TreeNode root) {  
    if (root == null) return 0; // Base case: If tree is empty  
  
    int leftDepth = maxDepth(root.left); // Recursively find left subtree depth  
    int rightDepth = maxDepth(root.right); // Recursively find right subtree depth  
  
    return Math.max(leftDepth, rightDepth) + 1; // Return the maximum depth  
}  
}
```



The screenshot shows a code editor with a light blue background. At the top, there's a tab labeled 'Code' with a green icon. Below the tab, there's a toolbar with icons for Java, Auto, and other editor functions. The code is written in Java and is color-coded: keywords are in blue, comments are in green, and strings are in red. The code is as follows:

```
1 class Solution {  
2     public int maxDepth(TreeNode root) {  
3         if (root == null) return 0; // Base case: If tree is empty  
4  
5         int leftDepth = maxDepth(root.left); // Recursively find left subtree depth  
6         int rightDepth = maxDepth(root.right); // Recursively find right subtree depth  
7  
8         return Math.max(leftDepth, rightDepth) + 1; // Return the maximum depth  
9     }  
10 }  
11
```

**B.** `import java.util.*;`

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        inorderHelper(root, result);
        return result;
    }

    private void inorderHelper(TreeNode node, List<Integer> result) {
        if (node == null) return;

        inorderHelper(node.left, result); // Visit left subtree
        result.add(node.val);             // Visit root
        inorderHelper(node.right, result); // Visit right subtree
    }
}
```

 Code

Java   Auto

```
1  import java.util.*;
2
3  class Solution {
4  ✓  public List<Integer> inorderTraversal(TreeNode root) {
5      List<Integer> result = new ArrayList<>();
6      inorderHelper(root, result);
7      return result;
8  }
9
10     private void inorderHelper(TreeNode node, List<Integer> result) {
11         if (node == null) return;
12
13         inorderHelper(node.left, result); // Visit left subtree
14         result.add(node.val);             // Visit root
15         inorderHelper(node.right, result); // Visit right subtree
16     }
17 }
18
```

Saved

## 4. Output:

### A.

☒ Testcase | [Test Result](#)

**Accepted** Runtime: 0 ms

• Case 1

• Case 2

Input

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

Output

3

Expected

3

### B.

☒ Testcase | [Test Result](#)

**Accepted** Runtime: 0 ms

• Case 1

• Case 2

• Case 3

• Case 4

Input

root =  
[1,null,2,3]

Output

[1,3,2]

Expected

[1,3,2]

## 5. Learning Outcome:

- Understand string manipulation techniques in C++.
- Implement efficient algorithms for detecting cyclic rotations.
- Apply mathematical approaches to solve missing number problems.
- Utilize standard library functions like accumulate and find.
- Enhance problem-solving skills through algorithm design and analysis.