Experiment 6

Name: Tuntun kumar **UID: 22BCS16442**

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:

a. To find and implement the maximum depth of Binary Tree.

b. 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
        }
    }
```

Code

Java Auto

```
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

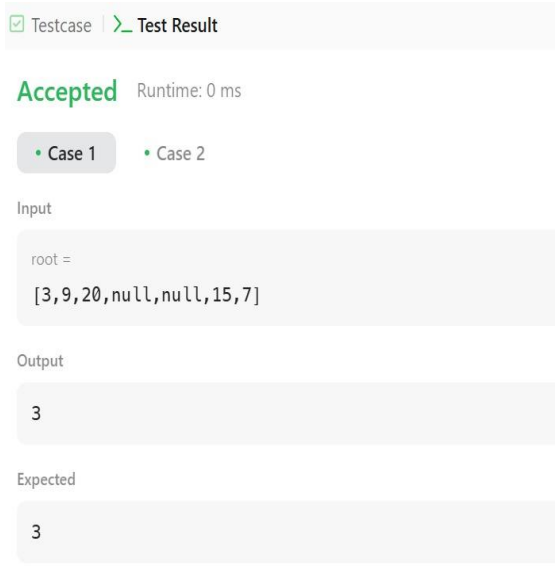
Java Auto

```java
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
    }
}
```

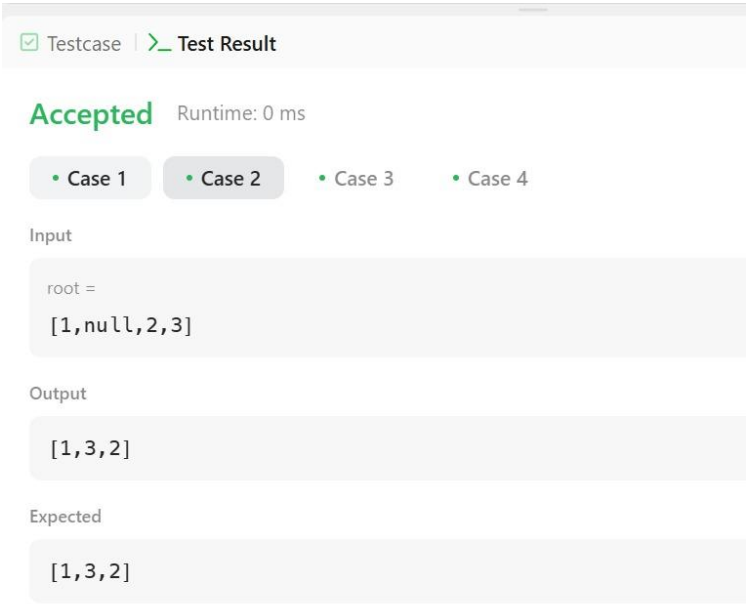
Saved

4. Output:

A.



B.



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.