



## Experiment 6.1

**Student Name:** Vishal Bhatia

**Branch:** BE-CSE

**Semester:** 6<sup>th</sup>

**Subject Name:** Advanced Programming-2

**UID:** 22BCS12608

**Section/Group:** IoT\_641(A)

**Date of Performance:** 4/3/25

**Subject Code:** 22CSP-351

**1. Aim:** Check if a binary tree is a mirror of itself. Use recursive DFS or iterative BFS (queue) to compare corresponding nodes. The solution runs in  $O(n)$  time and  $O(h)$  space. Edge cases include an empty tree (symmetric) and a single-node tree.

**2. Objective:** Implement an algorithm to check if tree is symmetrical.

**3. Implementation/Code:**

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

    bool isMirror(TreeNode* t1, TreeNode* t2) {
        if (!t1 && !t2) return true;
        if (!t1 || !t2) return false;
        return (t1->val == t2->val) &&
            isMirror(t1->left, t2->right) &&
            isMirror(t1->right, t2->left);
    }
};
```

**4. Output**

Case 1:

Input:

Root=[[1,2,2,3,4,4,3]

Output:

True

Case 2:

Input:

Root=[1,2,2,null,3,null,3]

Output:

false

## Experiment 6.2

1. **Aim:** - Binary Tree Inorder Traversal Return the inorder (left-root-right) traversal of a binary tree using recursion or an iterative stack-based approach. The solution runs in  $O(n)$  time and  $O(h)$  space. Edge cases include an empty tree ( $[]$ ) and a left-skewed or right-skewed tree
2. **Objective:** Implement algorithms to efficiently perform Binary tree inorder traversal.

### 3. Code:

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        inorder(root, res);
        return res;
    }

private:
    void inorder(TreeNode* node, vector<int>& res) {
        if (!node) {
            return;
        }
        inorder(node->left, res);
        res.push_back(node->val);
        inorder(node->right, res);
    }
};
```

### 4. Output

Case 1:

Input:

Root= [1,null,2,3]

[1,2,3,4,5,null,8,null,null,6,7,9]

Output:

[1,3,2]

Case 2:

Input:

Root=

Output:

[4,2,6,5,7,1,3,9,8]

## Experiment 6.3

1. **Aim:** Find the kth smallest element in a BST using an in-order traversal (iterative or recursive). The approach runs in  $O(k)$  time and  $O(h)$  space. Edge cases include  $k = 1$  (smallest element) and  $k = n$  (largest element).
2. **Objective:** Implement an algorithm to find the kth smallest element in a BST.

### 3. Code:

```
class Solution {
private:
    void inorder(TreeNode* root, vector<int>&a){
        if(root==nullptr){
            return;
        }

        // traverse left subtree
        inorder(root->left, a);
        // store value of root on which we are standing
        a.push_back(root->val);
        // process the right subtree
        inorder(root->right,a);
    }

public:
    int kthSmallest(TreeNode* root, int k) {
        // creating a vector of int
        vector<int>a;
        //calling inorder traversal
        inorder(root, a);
        //returning the ans..
        return a[k-1];
    }
};
```

### 4. Output

Case 1:

Input:

Root= [3,1,4,null,2] k=1

Output:

1

Case 2:

Input:

root= [5,3,6,2,4,null,null,1] k=3

Output:

3



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## 5. Learning Outcomes:

- Understand the steps involved in BFS and DFS.
- Learnt the concept of tree symmetry.
- Learn how to analyze and compare the time complexity.
- Implement appropriate algorithm based on the problem's constraints.