Experiment 3

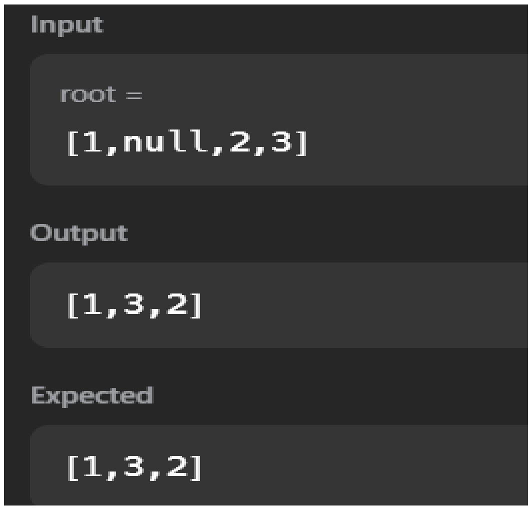
Student Name: Lokesh Gupta **UID: 22BCS16079**
Branch: CSE **Section/Group: FL_IOT-602**
Semester: 06TH **Date of Performance: 14/02/25**
Subject Name: Advance Programming Lab II **Subject Code: 22CSP-351**

1. Binary Tree Inorder Traversal(94)

Code:

```
class Solution {
public:
    void inorder(TreeNode* root, vector<int>& result) {
        if (!root) return;
        inorder(root->left, result);
        result.push_back(root->val);
        inorder(root->right, result); }
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        inorder(root, result);
        return result;
    }
};
```

Output:



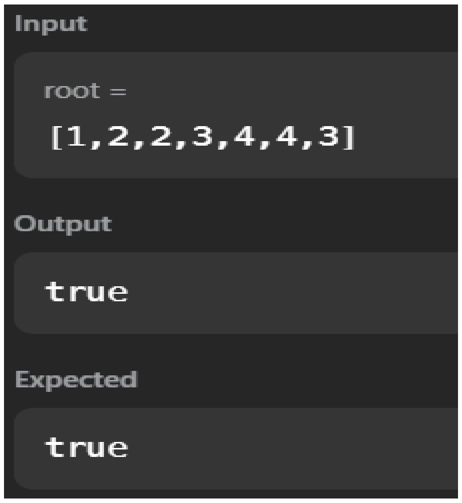
```
Input
root =
[1,null,2,3]
Output
[1,3,2]
Expected
[1,3,2]
```

2. **Symmetric Tree(101)**

   Code:

```
class Solution {
public:
   bool isMirror(TreeNode* left, TreeNode* right) {
      if (!left && !right) return true;  // Both are null, symmetric
      if (!left || !right) return false; // One is null, not symmetric
      if (left->val != right->val) return false; // Values do not match

      // Check symmetry recursively
      return isMirror(left->left, right->right) && isMirror(left->right, right->left);
   }
   bool isSymmetric(TreeNode* root) {
      if (!root) return true;  // Empty tree is symmetric
      return isMirror(root->left, root->right);
   }
};
```

   Output:



```
Input

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

Output

true

Expected

true
```

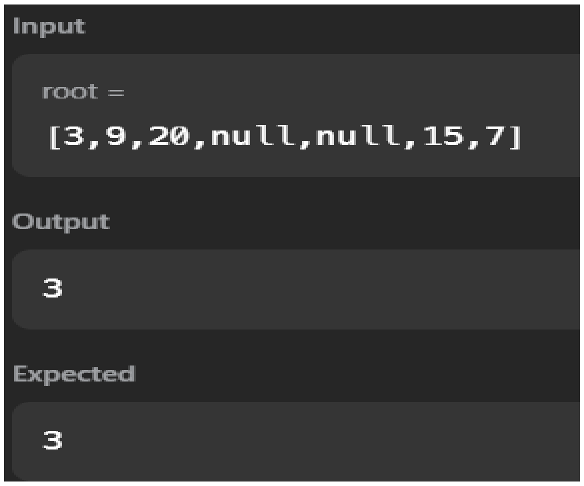
3. **Maximum Depth of Tree(104)**

   Code:

```
class Solution {
public:
   int maxDepth(TreeNode* root) {
      if (root == nullptr) return 0; // Base case: If tree is empty, depth is 0
         int left = maxDepth(root->left);  // Recursively find left depth
```

```
        int right = maxDepth(root->right); // Recursively find right depth
        return max(left, right) + 1; // Return the max depth plus 1 for the current node
    }
};
```

Output:



Input

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

Output
```
3
```

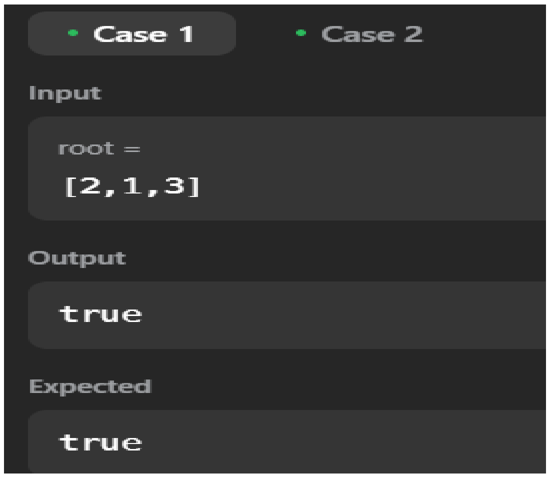
Expected
```
3
```

4. **Validate Binary Search Tree(98)**

Code:

```
class Solution {
public:
    bool validate(TreeNode* root, long minVal, long maxVal) {
        if (!root) return true; // Base case: empty tree is valid
        if (root->val <= minVal || root->val >= maxVal) return false; // Violates BST rule
        return validate(root->left, minVal, root->val) &&
               validate(root->right, root->val, maxVal);
    }
    bool isValidBST(TreeNode* root) {
        return validate(root, LONG_MIN, LONG_MAX);
    }
};
```

Output:



Case 1 Case 2

Input

```
root =
[2,1,3]
```

Output

```
true
```

Expected

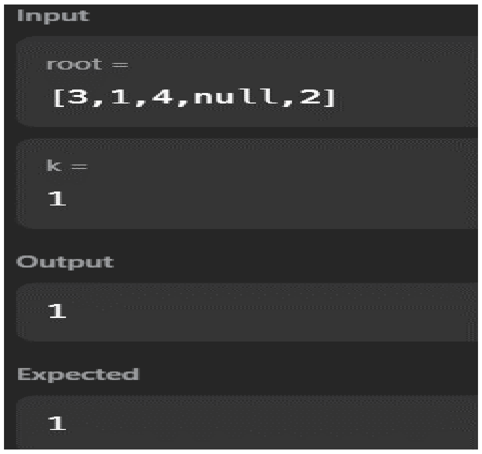
```
true
```

5. **Kth Smallest Element in a BST(230)**

Code:

```
class Solution {
public:
    void inorder(TreeNode* root, vector<int>& result) {
        if (!root) return;
        inorder(root->left, result);
        result.push_back(root->val);
        inorder(root->right, result);  }
    int kthSmallest(TreeNode* root, int k) {
        vector<int> result;
        inorder(root, result);
        return result[k - 1]; // k-th smallest element (1-based index)
    }
};
```

Output:



Input

```
root =
[3,1,4,null,2]
```

```
k =
1
```

Output

```
1
```

Expected

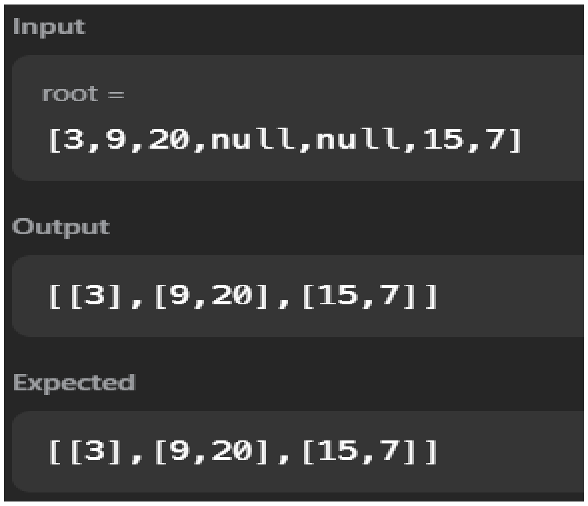
```
1
```

6. Binary Tree Level Order Traversal(102)

Code:

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if (!root) return result;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int size = q.size();
            vector<int> level;
            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();
                level.push_back(node->val);
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.push_back(level);
        }
        return result;
    }
};
```

Output:



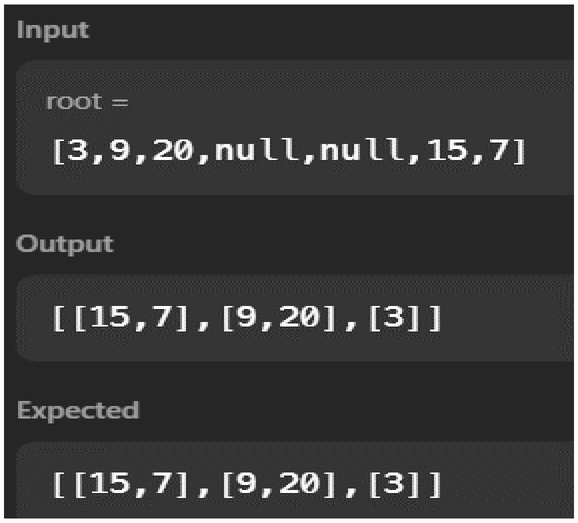
```
Input
root =
[3,9,20,null,null,15,7]
Output
[[3],[9,20],[15,7]]
Expected
[[3],[9,20],[15,7]]
```

7. Binary Tree Level Order Traversal (107)

Code:

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> result;
        if (!root) return result;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int size = q.size();
            vector<int> level;
            for (int i = 0; i < size; i++) {
                TreeNode* node = q.front();
                q.pop();
                level.push_back(node->val);
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            result.push_back(level); }
        reverse(result.begin(), result.end()); // Reverse to get bottom-up order
        return result;
    }
};
```

Output:



Input

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

Output

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

Expected

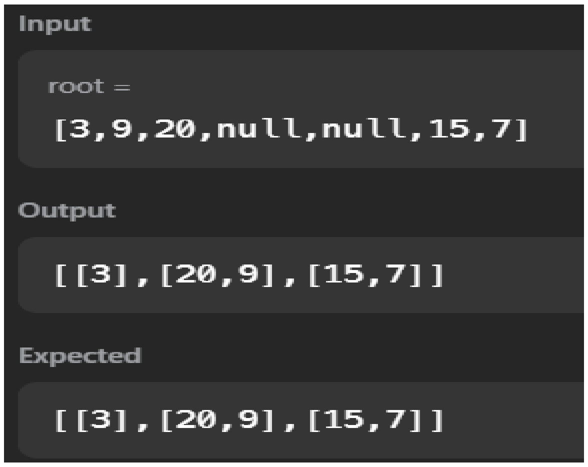
```
[[15,7],[9,20],[3]]
```

8. Binary Tree Zigzag Level Order Traversal(103)

Code:

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if (!root) return res;  // Return empty result if tree is empty
        queue<TreeNode*> q;
        q.push(root);
        bool leftToRight = true;  // Direction flag
        while (!q.empty()) {
            int n = q.size();
            vector<int> level(n); // Store nodes at current level
            for (int i = 0; i < n; i++) {
                TreeNode* node = q.front();
                q.pop();
                int index = leftToRight ? i : (n - 1 - i);
                level[index] = node->val;
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
            res.push_back(level);
            leftToRight = !leftToRight;  // Toggle direction
        }
        return res; }
};
```

Output:



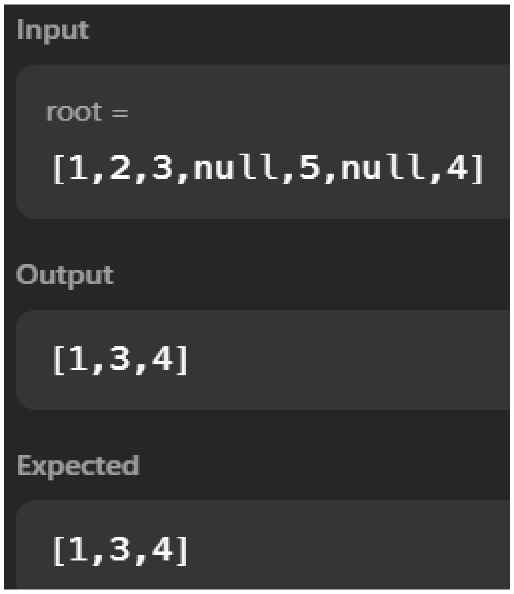
```
Input
root =
[3,9,20,null,null,15,7]
Output
[[3],[20,9],[15,7]]
Expected
[[3],[20,9],[15,7]]
```

9. **Binary Tree Right Side View(199)**

Code:

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> res;
        if (!root) return res;
        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            int n = q.size();
            for (int i = 0; i < n; i++) {
                TreeNode* node = q.front();
                q.pop();
                if (i == n - 1) res.push_back(node->val);
                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }
        }
        return res;
    }
};
```

Output:



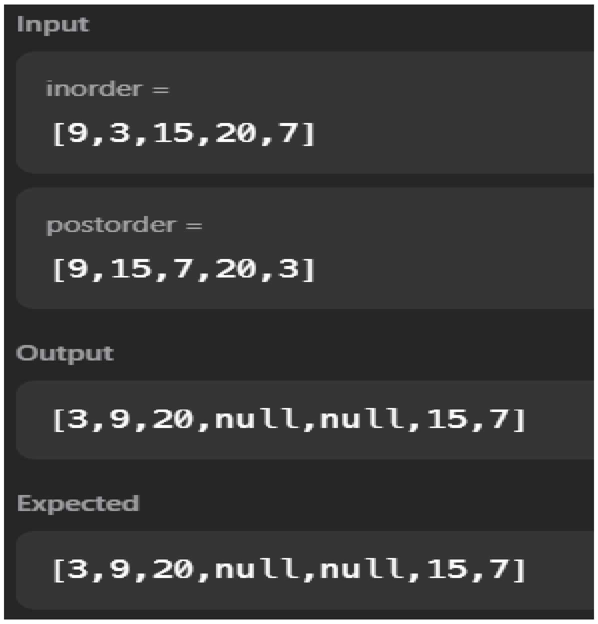
```
Input

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

Output

[1,3,4]

Expected

[1,3,4]
```

10. Construct Binary Tree from Inorder and Postorder Traversal (106)

Code:

```
class Solution {
public:
    unordered_map<int, int> inorderMap;
    int postIndex;
    TreeNode* helper(vector<int>& inorder, vector<int>& postorder, int left, int right) {
        if (left > right) return nullptr;
        int rootVal = postorder[postIndex--];
        TreeNode* root = new TreeNode(rootVal);
        int inorderIndex = inorderMap[rootVal];
        root->right = helper(inorder, postorder, inorderIndex + 1, right);
        root->left = helper(inorder, postorder, left, inorderIndex - 1);
        return root;
    }
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        postIndex = postorder.size() - 1;
        for (int i = 0; i < inorder.size(); i++)
            inorderMap[inorder[i]] = i;
        return helper(inorder, postorder, 0, inorder.size() - 1);
    }
};
```

Output:



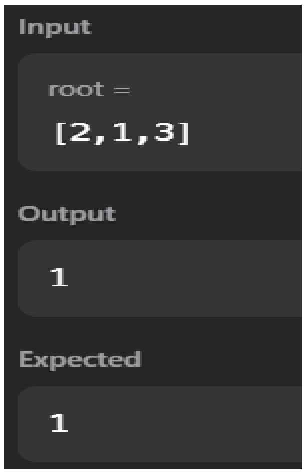
```
Input
inorder =
[9,3,15,20,7]
postorder =
[9,15,7,20,3]
Output
[3,9,20,null,null,15,7]
Expected
[3,9,20,null,null,15,7]
```

11.Find Bottom Left Tree Value(513)

Code:

```
class Solution {
public:
    int findBottomLeftValue(TreeNode* root) {
        queue<TreeNode*> q;
        q.push(root);
        int bottomLeft = root->val;
        while (!q.empty()) {
            TreeNode* node = q.front();
            q.pop();
            bottomLeft = node->val;
            if (node->right) q.push(node->right);
            if (node->left) q.push(node->left);
        }
        return bottomLeft;
    }
};
```

Output:



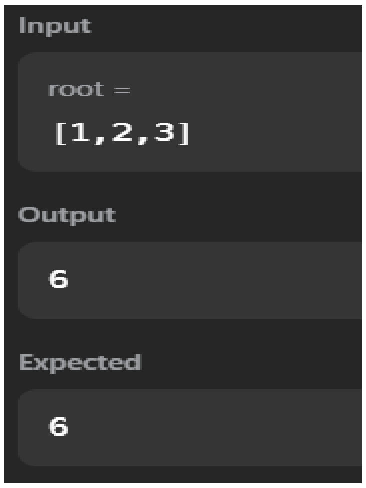
```
Input
root =
[2,1,3]
Output
1
Expected
1
```

12.Binary Tree Maximum Path Sum(124)

Code:

```
class Solution {
public:
    int maxSum = INT_MIN;
    int helper(TreeNode* node) {
        if (!node) return 0;
        int left = max(0, helper(node->left));
        int right = max(0, helper(node->right));
        maxSum = max(maxSum, left + right + node->val);
        return node->val + max(left, right);
    }
    int maxPathSum(TreeNode* root) {
        helper(root);
        return maxSum;
    }
};
```

Output:



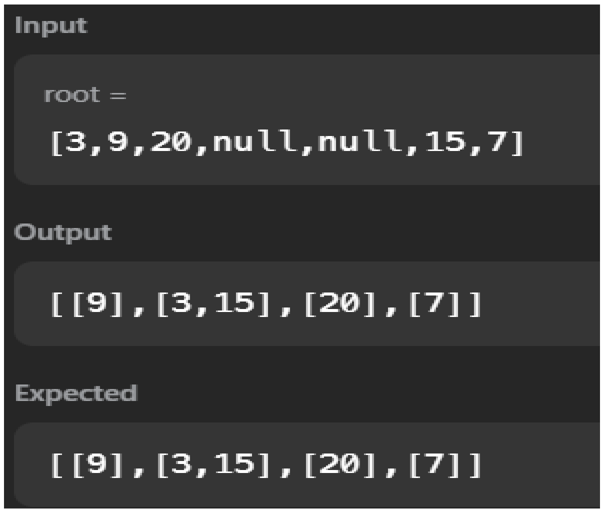
```
Input
root =
[1,2,3]
Output
6
Expected
6
```

13.Vertical Order Traversal of a Binary Tree(987)

Code:

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        map<int, map<int, vector<int>>> nodes;  // {x: {y: [values]}}
        queue<pair<TreeNode*, pair<int, int>>> q;  // {node, {x, y}}
        q.push({root, {0, 0}});
        while (!q.empty()) {
            auto [node, pos] = q.front();
            q.pop();
            int x = pos.first, y = pos.second;
            nodes[x][y].push_back(node->val);
            if (node->left) q.push({node->left, {x - 1, y + 1}});
            if (node->right) q.push({node->right, {x + 1, y + 1}});
        }
        vector<vector<int>> res;
        for (auto& [x, levelMap] : nodes) {
            vector<int> col;
            for (auto& [y, values] : levelMap) {
                sort(values.begin(), values.end());
                col.insert(col.end(), values.begin(), values.end());
            }
            res.push_back(col);
        }
        return res;
    }
};
```

Output:



```
Input

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

Output

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

Expected

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