Day -6

Name: Jobanjeet Singh

Uid : 22BCS15377

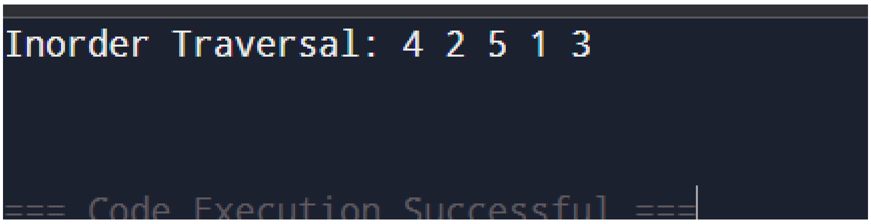
Section : 620 -B

Q1. Binary Order Traversal

```
#include <iostream>

using namespace std;


struct TreeNode {

    int val;

    TreeNode* left;

    TreeNode* right;

    TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};


void inorderTraversal(TreeNode* root) {

    if (root == NULL) return;

    inorderTraversal(root->left);

    cout << root->val << " ";

    inorderTraversal(root->right);

}
```

```
int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);

    cout << "Inorder Traversal: ";
    inorderTraversal(root); // Output: 4 2 5 1 3
    cout << endl;

    return 0;
}
```

Output:



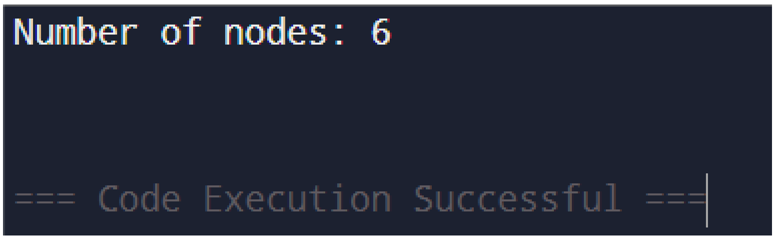
```
Inorder Traversal: 4 2 5 1 3


=== Code Execution Successful ===
```

Q2. Count Complete Tree Node

```
#include <iostream>
#include <cmath>
using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int getHeight(TreeNode* node) {
    int height = 0;
    while (node) {
        height++;
        node = node->left;
    }
    return height;
}

int countNodes(TreeNode* root) {
    if (!root) return 0;
    int leftHeight = getHeight(root->left);
    int rightHeight = getHeight(root->right);
    if (leftHeight == rightHeight) {
        return (1 << leftHeight) + countNodes(root->right);
    } else {
        return (1 << rightHeight) + countNodes(root->left);
    }
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);

    cout << "Number of nodes: " << countNodes(root) << endl; // Output: 6

    return 0;
}
```

Output:



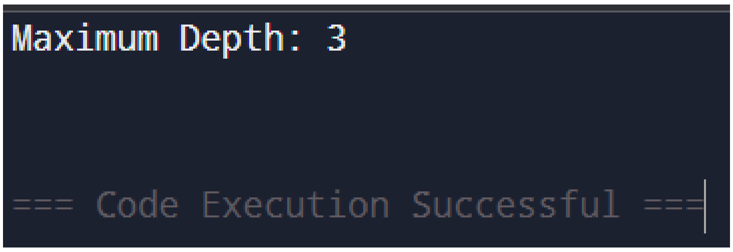
```
Number of nodes: 6


=== Code Execution Successful ===
```

Q3. Binary Tree – Find Maximum Depth

```
#include <iostream>
#include <algorithm>
using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int maxDepth(TreeNode* root) {
    if (!root) return 0;
    int leftDepth = maxDepth(root->left);
    int rightDepth = maxDepth(root->right);
    return 1 + max(leftDepth, rightDepth);
}

int main() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(9);
    root->right = new TreeNode(20);
    root->right->left = new TreeNode(15);
    root->right->right = new TreeNode(7);

    cout << "Maximum Depth: " << maxDepth(root) << endl; // Output: 3

    return 0;
}
```

Output:

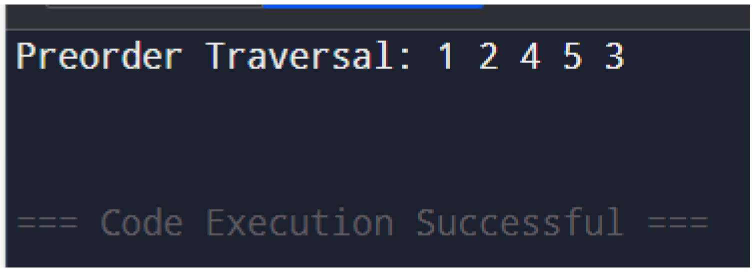


```
Maximum Depth: 3


=== Code Execution Successful ===
```

Q4. Binary Order Pre Traversal

```
#include <iostream>

#include <vector>

#include <stack>

using namespace std;


struct TreeNode {

    int val;

    TreeNode* left;

    TreeNode* right;

    TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};


vector<int> preorderTraversal(TreeNode* root) {

    vector<int> result;

    if (root == NULL) return result;

    stack<TreeNode*> stk;

    stk.push(root);
    while (!stk.empty()) {
        TreeNode* node = stk.top();
        stk.pop();
        result.push_back(node->val);
        if (node->right) stk.push(node->right);
        if (node->left) stk.push(node->left);
    }
    return result;
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);

    vector<int> result = preorderTraversal(root);

    cout << "Preorder Traversal: ";
    for (int val : result) {
        cout << val << " ";
    }
    cout << endl;
    return 0;
}
```

Output:

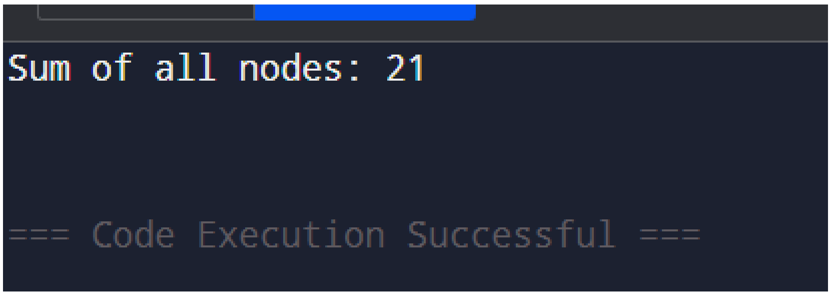


```
Preorder Traversal: 1 2 4 5 3


=== Code Execution Successful ===
```

Q5. Binary Tree – Sum of all Nodes

```
#include <iostream>
#include <queue>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int sumOfNodes(TreeNode* root) {
    if (root == NULL) return 0;
    int sum = 0;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* current = q.front();
        q.pop();
        sum += current->val;
        if (current->left) q.push(current->left);
        if (current->right) q.push(current->right);
    }
    return sum;
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->right = new TreeNode(6);

    cout << "Sum of all nodes: " << sumOfNodes(root) << endl; // Output: 21

    return 0;
}
```

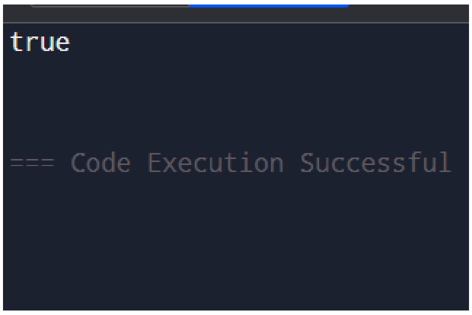
Output:



```
Sum of all nodes: 21


=== Code Execution Successful ===
```

Q6. Same Tree

```
#include <iostream>

#include <queue>

using namespace std;


struct TreeNode {

    int val;

    TreeNode* left;

    TreeNode* right;

    TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};


bool isSameTree(TreeNode* p, TreeNode* q) {

    queue<TreeNode*> qp, qq;

    qp.push(p);

    qq.push(q);

    while (!qp.empty() && !qq.empty()) {

        TreeNode* nodeP = qp.front(); qp.pop();

        TreeNode* nodeQ = qq.front(); qq.pop();
        if (!nodeP && !nodeQ) continue;
        if (!nodeP || !nodeQ || nodeP->val != nodeQ->val) return false;
        qp.push(nodeP->left);
        qp.push(nodeP->right);
        qq.push(nodeQ->left);
        qq.push(nodeQ->right);
    }
    return qp.empty() && qq.empty();
}

int main() {
    TreeNode* p = new TreeNode(1);
    p->left = new TreeNode(2);
    p->right = new TreeNode(3);
    TreeNode* q = new TreeNode(1);
    q->left = new TreeNode(2);
    q->right = new TreeNode(3);
    cout << (isSameTree(p, q) ? "true" : "false") << endl;
    return 0;
}
```

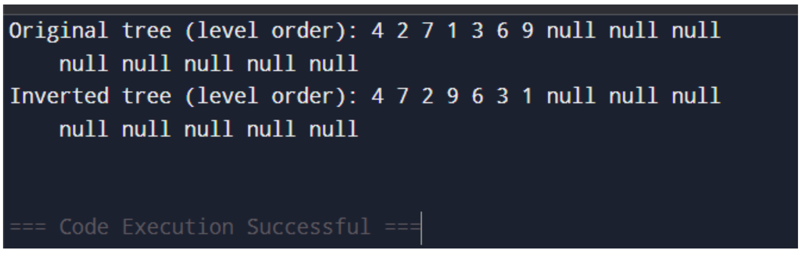


```
true


=== Code Execution Successful
```

Q7. Invert Binary Tree

```
#include <iostream>
#include <queue>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

TreeNode* invertTree(TreeNode* root) {
    if (root == NULL) return NULL;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* node = q.front();
        q.pop();
        TreeNode* temp = node->left;
        node->left = node->right;
        node->right = temp;
        if (node->left) q.push(node->left);
        if (node->right) q.push(node->right);
    }
    return root;
}

void printLevelOrder(TreeNode* root) {
    if (root == NULL) return;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* node = q.front();
        q.pop();
        if (node) {
            cout << node->val << " ";
            q.push(node->left);
            q.push(node->right);
        } else {
            cout << "null ";
        }
    }
}

int main() {
    TreeNode* root = new TreeNode(4);
    root->left = new TreeNode(2);
    root->right = new TreeNode(7);
    root->left->left = new TreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(9);

    cout << "Original tree (level order): ";
    printLevelOrder(root);
    cout << endl;

    root = invertTree(root);
    cout << "Inverted tree (level order): ";
    printLevelOrder(root);
    cout << endl;

    return 0;
}
```

Output:



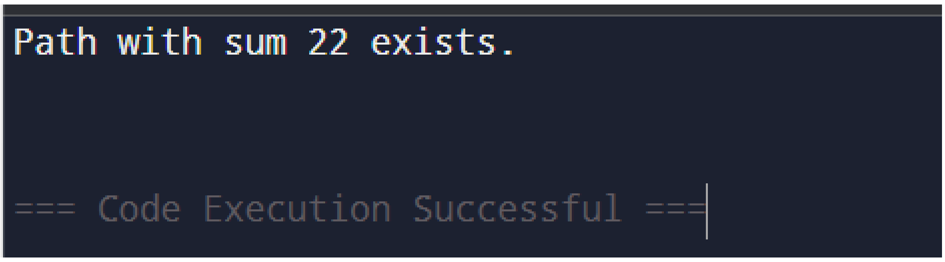
```
Original tree (level order): 4 2 7 1 3 6 9 null null null
    null null null null null
Inverted tree (level order): 4 7 2 9 6 3 1 null null null
    null null null null null


=== Code Execution Successful ===
```

Q8. Path Sum

```
#include <iostream>
```

```
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

bool hasPathSum(TreeNode* root, int sum) {
    if (root == NULL) return false;
    if (root->left == NULL && root->right == NULL) {
        return sum == root->val;
    }
    int remainingSum = sum - root->val;
    return hasPathSum(root->left, remainingSum) || hasPathSum(root->right, remainingSum);
}

int main() {
    TreeNode* root = new TreeNode(5);
    root->left = new TreeNode(4);
    root->right = new TreeNode(8);
    root->left->left = new TreeNode(11);
    root->right->left = new TreeNode(13);
    root->right->right = new TreeNode(4);
    root->left->left->left = new TreeNode(7);
    root->left->left->right = new TreeNode(2);
    root->right->right->right = new TreeNode(1);

    int targetSum = 22;
    if (hasPathSum(root, targetSum)) {
        cout << "Path with sum " << targetSum << " exists." << endl;
    } else {
        cout << "No path with sum " << targetSum << " exists." << endl;
    }

    return 0;
}
```

Output:



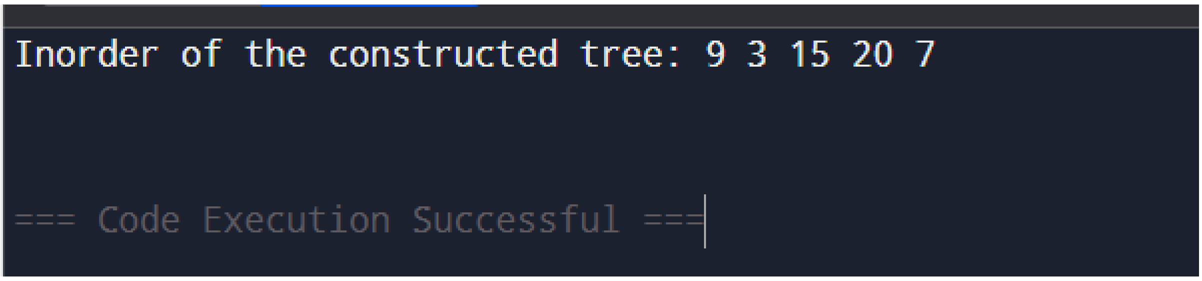
```
Path with sum 22 exists.


=== Code Execution Successful ===
```

Q9. Construct Binary Tree from Preorder and Inorder Traversal

```
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

TreeNode* buildTreeHelper(vector<int>& preorder, int preStart, int preEnd, vector<int>& inorder, int inStart, int inEnd, unordered_map<int, int>& inorderMap) {
    if (preStart > preEnd || inStart > inEnd) return NULL;
    int rootVal = preorder[preStart];
    TreeNode* root = new TreeNode(rootVal);
    int inRootIndex = inorderMap[rootVal];
    int leftTreeSize = inRootIndex - inStart;
    root->left = buildTreeHelper(preorder, preStart + 1, preStart + leftTreeSize, inorder, inStart, inRootIndex - 1, inorderMap);
    root->right = buildTreeHelper(preorder, preStart + leftTreeSize + 1, preEnd, inorder, inRootIndex + 1, inEnd, inorderMap);
    return root;
}

TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
    unordered_map<int, int> inorderMap;
    for (int i = 0; i < inorder.size(); i++) {
        inorderMap[inorder[i]] = i;
    }
    return buildTreeHelper(preorder, 0, preorder.size() - 1, inorder, 0, inorder.size() - 1, inorderMap);
}

void printInorder(TreeNode* root) {
    if (root == NULL) return;
    printInorder(root->left);
    cout << root->val << " ";
    printInorder(root->right);
}

int main() {
    vector<int> preorder = {3, 9, 20, 15, 7};
    vector<int> inorder = {9, 3, 15, 20, 7};
    TreeNode* root = buildTree(preorder, inorder);

    cout << "Inorder of the constructed tree: ";
    printInorder(root);
    cout << endl;

    return 0;
}
```

Output:

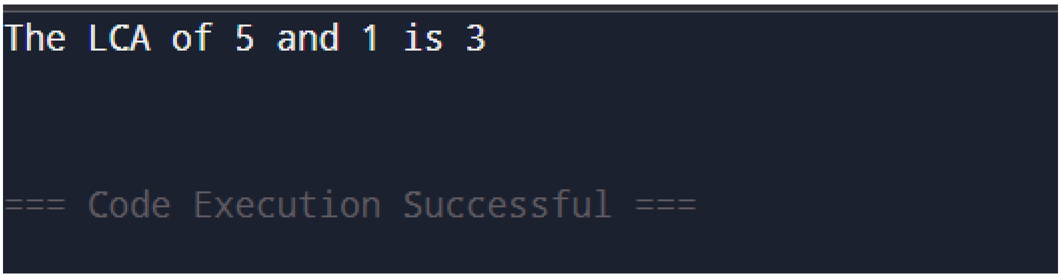


```
Inorder of the constructed tree: 9 3 15 20 7


=== Code Execution Successful ===
```

Q10. Lowest Common Ancestor Binary Tree

```
#include <iostream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p,
TreeNode* q) {
        if (root == NULL || root == p || root == q) {
            return root;
        }
        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);
        if (left != NULL && right != NULL) {
            return root;
        }
        return (left != NULL) ? left : right;
    }
};

TreeNode* createTree() {
    TreeNode* root = new TreeNode(3);
    root->left = new TreeNode(5);
    root->right = new TreeNode(1);
    root->left->left = new TreeNode(6);
    root->left->right = new TreeNode(2);
    root->right->left = new TreeNode(0);
    root->right->right = new TreeNode(8);
    root->left->right->left = new TreeNode(7);
    root->left->right->right = new TreeNode(4);
    return root;
}

int main() {
    Solution solution;
    TreeNode* root = createTree();
    TreeNode* p = root->left;
    TreeNode* q = root->right;
    TreeNode* lca = solution.lowestCommonAncestor(root, p, q);

    if (lca != NULL) {

        cout << "The LCA of " << p->val << " and " << q->val << " is " << lca->val << endl;

    } else {

        cout << "No common ancestor found." << endl;

    }

    return 0;

}
```

Output:



```
The LCA of 5 and 1 is 3


=== Code Execution Successful ===
```