#include <iostream>

#include <queue>

using namespace std;

// Node structure for the binary tree

struct Node {

    int data;

    Node\* left;

    Node\* right;

};

// Function to create a new node

Node\* createNode(int value) {

    Node\* newNode = new Node();

    newNode->data = value;

    newNode->left = newNode->right = nullptr;

    return newNode;

}

// Function to count the number of nodes in the tree

int countNodes(Node\* root) {

    if (root == nullptr)

        return 0;

    return 1 + countNodes(root->left) + countNodes(root->right);

}

// Function to count the number of leaf nodes in the tree

int countLeafNodes(Node\* root) {

    if (root == nullptr)

        return 0;

    if (root->left == nullptr && root->right == nullptr)

        return 1;

    return countLeafNodes(root->left) + countLeafNodes(root->right);

}

// Function to print leaf nodes of the tree

void printLeafNodes(Node\* root) {

    if (root == nullptr)

        return;

    if (root->left == nullptr && root->right == nullptr)

        cout << root->data << " ";

    printLeafNodes(root->left);

    printLeafNodes(root->right);

}

// Function to find the height of the tree

int heightOfTree(Node\* root) {

    if (root == nullptr)

        return -1; // Height of empty tree is -1

    int leftHeight = heightOfTree(root->left);

    int rightHeight = heightOfTree(root->right);

    return 1 + max(leftHeight, rightHeight);

}

// Function to create the mirror image of the tree

Node\* mirrorImage(Node\* root) {

    if (root == nullptr)

        return nullptr;

    Node\* temp = root->left;

    root->left = mirrorImage(root->right);

    root->right = mirrorImage(temp);

    return root;

}

// Function to print the tree in level order (BFS)

void printLevelOrder(Node\* root) {

    if (root == nullptr)

        return;

    queue<Node\*> q;

    q.push(root);

    while (!q.empty()) {

        Node\* current = q.front();

        q.pop();

        cout << current->data << " ";

        if (current->left != nullptr)

            q.push(current->left);

        if (current->right != nullptr)

            q.push(current->right);

    }

    cout << endl;

}

// Function to build a binary tree from user input

Node\* buildTree() {

    int n;

    cout << "Enter the number of nodes in the tree: ";

    cin >> n;

    Node\* root = nullptr;

    cout << "Enter the values of the nodes: ";

    for (int i = 0; i < n; ++i) {

        int value;

        cin >> value;

        if (root == nullptr)

            root = createNode(value);

        else {

            Node\* current = root;

            Node\* parent = nullptr;

            while (current != nullptr) {

                parent = current;

                if (value < current->data)

                    current = current->left;

                else

                    current = current->right;

            }

            if (value < parent->data)

                parent->left = createNode(value);

            else

                parent->right = createNode(value);

        }

    }

    return root;

}

int main() {

    Node\* root = buildTree();

    cout << "Number of nodes in the tree: " << countNodes(root) << endl;

    cout << "Number of leaf nodes in the tree: " << countLeafNodes(root) << endl;

    cout << "Leaf nodes of the tree: ";

    printLeafNodes(root);

    cout << endl;

    cout << "Height of the tree: " << heightOfTree(root) << endl;

    cout << "Mirror image of the tree (level order):" << endl;

    Node\* mirrorRoot = mirrorImage(root);

    printLevelOrder(mirrorRoot);

    cout << "Number of nodes in the mirror tree: " << countNodes(mirrorRoot) << endl;

    cout << "Height of the mirror tree: " << heightOfTree(mirrorRoot) << endl;

    return 0;

}