#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) {

data = value;

left = right = nullptr;

}

};

Node\* insert(Node\* root, int value) {

if (root == nullptr)

return new Node(value);

if (value < root->data)

root->left = insert(root->left, value);

else

root->right = insert(root->right, value);

return root;

}

int longestPath(Node\* root) {

if (root == nullptr)

return 0;

int leftHeight = longestPath(root->left);

int rightHeight = longestPath(root->right);

return max(leftHeight, rightHeight) + 1;

}

int findMin(Node\* root) {

if (root == nullptr) {

cout << "Tree is empty.\n";

return -1;

}

while (root->left != nullptr)

root = root->left;

return root->data;

}

void mirror(Node\* root) {

if (root == nullptr)

return;

swap(root->left, root->right);

mirror(root->left);

mirror(root->right);

}

bool search(Node\* root, int key) {

if (root == nullptr)

return false;

if (root->data == key)

return true;

else if (key < root->data)

return search(root->left, key);

else

return search(root->right, key);

}

void inorder(Node\* root) {

if (root == nullptr) return;

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

int main() {

Node\* root = nullptr;

int n, val;

cout << "Enter the number of initial elements to insert in the BST: ";

cin >> n;

cout << "Enter " << n << " elements:\n";

for (int i = 0; i < n; ++i) {

cin >> val;

root = insert(root, val);

}

cout << "Inorder Traversal of BST: ";

inorder(root);

cout << endl;

cout << "\nEnter a value to insert into the BST: ";

cin >> val;

root = insert(root, val);

cout << "BST after insertion: ";

inorder(root);

cout << endl;

cout << "\nNumber of nodes in the longest path from root: " << longestPath(root) << endl;

cout << "Minimum data value in the BST: " << findMin(root) << endl;

mirror(root);

cout << "BST after mirroring (Inorder Traversal): ";

inorder(root);

cout << endl;

cout << "\nEnter a value to search in the BST: ";

cin >> val;

cout << "Search result: " << (search(root, val) ? "Found" : "Not Found") << endl;

return 0;

}