#include <iostream>

#include <algorithm>

#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 calculate the height of a tree

int height(Node\* root) {

if (root == nullptr)

return 0;

return max(height(root->left), height(root->right)) + 1;

}

// Function to calculate the balance factor of a node

int balanceFactor(Node\* root) {

if (root == nullptr)

return 0;

return height(root->left) - height(root->right);

}

// Function to insert a value into the binary tree

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

if (root == nullptr)

return createNode(value);

if (value < root->data)

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

else if (value > root->data)

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

// Calculate balance factor after each insertion

cout << "Balance Factor after inserting " << value << ": " << balanceFactor(root) << endl;

return root;

}

// Function to perform left rotation

Node\* leftRotation(Node\* root) {

Node\* newRoot = root->right;

root->right = newRoot->left;

newRoot->left = root;

return newRoot;

}

// Function to perform right rotation

Node\* rightRotation(Node\* root) {

Node\* newRoot = root->left;

root->left = newRoot->right;

newRoot->right = root;

return newRoot;

}

// Function to identify rotation cases for AVL balancing

Node\* avlBalancing(Node\* root) {

int bf = balanceFactor(root);

if (abs(bf) <= 1)

return root; // Tree is balanced

if (bf > 1) {

if (balanceFactor(root->left) < 0) {

// LR Rotation Case

root->left = leftRotation(root->left);

}

// LL Rotation Case

root = rightRotation(root);

} else {

if (balanceFactor(root->right) > 0) {

// RL Rotation Case

root->right = rightRotation(root->right);

}

// RR Rotation Case

root = leftRotation(root);

}

return root;

}

// Function to perform level order traversal (BFS) of the tree

void levelOrderTraversal(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);

}

}

int main() {

Node\* root = nullptr;

int n, value;

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

cin >> n;

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

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

cin >> value;

root = insertNode(root, value);

// Rebalance the tree after each insertion

root = avlBalancing(root);

}

cout << "Tree (Level Order Traversal): ";

levelOrderTraversal(root);

cout << endl;

return 0;

}