**DSA Lab Task (outputs with code**

#include <iostream>

using namespace std;

struct BSTNode {

int data;

BSTNode\* left;

BSTNode\* right;

BSTNode(int val) : data(val), left(nullptr), right(nullptr) {}

};

class BST {

public:

BSTNode\* root;

BST() : root(nullptr) {}

BSTNode\* insert(BSTNode\* node, int val) {

if (!node) return new BSTNode(val);

if (val < node->data)

node->left = insert(node->left, val);

else if (val > node->data)

node->right = insert(node->right, val);

return node;

}

void inorder(BSTNode\* node) {

if (!node) return;

inorder(node->left);

cout << node->data << " ";

inorder(node->right);

}

};

struct AVLNode {

int data;

AVLNode\* left;

AVLNode\* right;

int height;

AVLNode(int val) : data(val), left(nullptr), right(nullptr), height(1) {}

};

class AVL {

public:

AVLNode\* root;

AVL() : root(nullptr) {}

int height(AVLNode\* node) {

return node ? node->height : 0;

}

int getBalance(AVLNode\* node) {

return node ? height(node->left) - height(node->right) : 0;

}

AVLNode\* rightRotate(AVLNode\* y) {

AVLNode\* x = y->left;

AVLNode\* T2 = x->right;

x->right = y;

y->left = T2;

y->height = 1 + max(height(y->left), height(y->right));

x->height = 1 + max(height(x->left), height(x->right));

return x;

}

AVLNode\* leftRotate(AVLNode\* x) {

AVLNode\* y = x->right;

AVLNode\* T2 = y->left;

y->left = x;

x->right = T2;

x->height = 1 + max(height(x->left), height(x->right));

y->height = 1 + max(height(y->left), height(y->right));

return y;

}

AVLNode\* insert(AVLNode\* node, int val) {

if (!node) return new AVLNode(val);

if (val < node->data)

node->left = insert(node->left, val);

else if (val > node->data)

node->right = insert(node->right, val);

else

return node;

node->height = 1 + max(height(node->left), height(node->right));

int balance = getBalance(node);

if (balance > 1 && val < node->left->data)

return rightRotate(node);

if (balance < -1 && val > node->right->data)

return leftRotate(node);

if (balance > 1 && val > node->left->data) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && val < node->right->data) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

void inorder(AVLNode\* node) {

if (!node) return;

inorder(node->left);

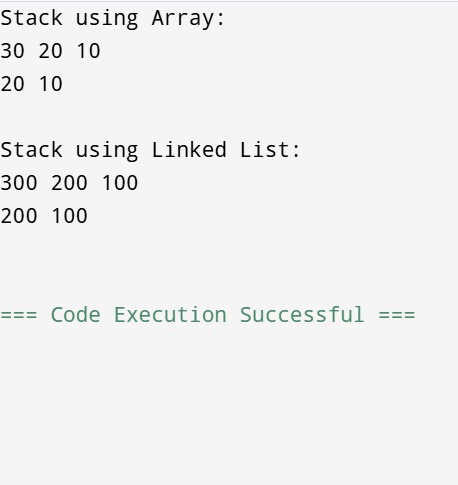
cout << node->data << " ";

inorder(node->right);

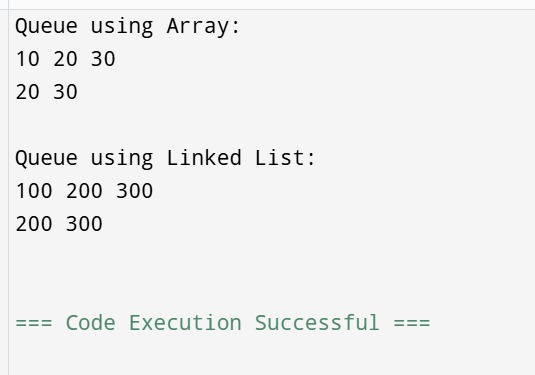
    }

};

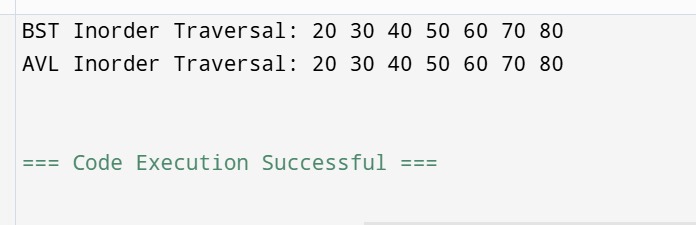
Output:



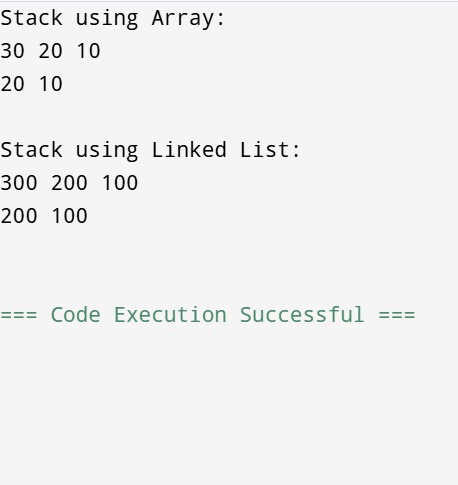
Output:



OutPut:



Output:



Output:

