DATA STRUCTURES LABORATORY MANUAL – ICE 2144

III SEMESTER B. TECH

EXPT. 7 TREES

1) Write a C++ program to search a given element in a Binary Tree.

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

class Node
{
    public:
        int data;
        Node *left, *right;
        Node(int data)
            {
                    this→data = data;
                    left = right = NULL;
            }
};

// Function to traverse the tree in preorder and check if the given node exists in it

bool ifNodeExists(Node* node, int key)
{
            if (node == NULL)
                    return false;

            if (node→data == key)
                    return true;

//then recur on left subtree
            bool res1 = ifNodeExists(node→left, key);
// node found, no need to look further
            if(res1)
                    return true;
//node is not found in left,so recur on right subtree
            bool res2 = ifNodeExists(node→right, key);
                    return res2;
}

int main()
{
            Node* root = new Node(0);
            root→left = new Node(1);
            root→left→left = new Node(3);
            root→left→left->left = new Node(7);
            root→left→right = new Node(4);
```

```
        root→left→right->left = new Node(8);
        root→left→right->right = new Node(9);
        root→right = new Node(2);
        root→right→left = new Node(5);
        root→right→right = new Node(6);

        int key = 4;

        if (ifNodeExists(root, key))
                cout << "YES";
        else
                cout << "NO";

        return 0;
}
```

2) Write a C++ program to delete an element from a binary search tree.

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

class Node
{
   public:
      int key;
      Node *left, *right;
};

// A utility function to create a new BST node
Node* newNode(int item)
{
        Node* temp = new Node;
        temp→key = item;
        temp→left = temp→right = NULL;
        return temp;
}

// A utility function to do inorder traversal of BST
void inorder(Node* root)
{
        if (root != NULL)
        {
                inorder(root→left);
                cout<<root→key;
                inorder(root→right);
        }
```

```
}

// A utility function to insert a new node with given key in BST
Node* insert(Node* node, int key)
{
	//If the tree is empty, return a new node
	if (node == NULL)
		return newNode(key);

	// Otherwise, recur down the tree
	if (key < node→key)
		node→left = insert(node→left, key);
	else
		node→right = insert(node→right, key);

	//return the (unchanged) node pointer
	return node;
}

/* Given a binary search tree and a key, this function deletes the key and returns the
new root */
Node* deleteNode(Node* root, int k)
{
	if (root == NULL)
		return root;

	// Recursive calls for ancestors of node to be deleted
	if (root→key > k)
	{
		root→left = deleteNode(root→left, k);
		return root;
	}
	else if (root→key < k)
	{
		root→right = deleteNode(root→right, k);
		return root;
	}

	// We reach here when root is the node to be deleted.

	// If one of the children is empty
	if (root→left == NULL)
	{
		Node* temp = root→right;
		delete root;
		return temp;
```

```
        }
        else if (root→right == NULL)
        {
                Node* temp = root→left;
                delete root;
                return temp;
        }
        // If both children exist
        else
        {
                Node* succParent = root;

                // Find successor
                Node* succ = root→right;
                while (succ→left != NULL)
                {
                        succParent = succ;
                        succ = succ→left;
                }

                /*Delete successor. Since successor is always left child of its parent
                we can safely make successor's right right child as left of its parent.
                If there is no succ, then assign succ→right to succParent→right*/

                if (succParent != root)
                        succParent→left = succ→right;
                else
                        succParent→right = succ→right;

                // Copy Successor Data to root
                root→key = succ→key;

                // Delete Successor and return root
                delete succ;
                return root;
        }
}

int main()
{
        /* Let us create following BST
                        50
                /           \
                30          70
                / \ / \
        20 40 60 80 */
```

```
    Node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    cout<<"Original BST: ";
    inorder(root);
    cout<<"\n\nDelete a Leaf Node: 20\n";
    root = deleteNode(root, 20);
    cout<<"Modified BST tree after deleting Leaf Node:\n";
    inorder(root);
    cout<<"\n\nDelete Node with single child: 70\n";
    root = deleteNode(root, 70);
    cout<<"Modified BST tree after deleting single child Node:\n";
    inorder(root);
    cout<<"\n\nDelete Node with both child: 50\n";
    root = deleteNode(root, 50);
    cout<<"Modified BST tree after deleting both child Node:\n";
    inorder(root);
    return 0;
}
```

3) Given a binary tree, write a program to find the height of the tree.

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

class node
{
    public:
        int data;
        node* left;
        node* right;
};

/* Compute the "maxDepth" of a tree -- the number of nodes along the longest path
from the root node down to the farthest leaf node. */

int maxDepth(node* node)
{
    if (node == NULL)
        return 0;
    else
```

```
        {
                // compute the depth of each subtree

                int lDepth = maxDepth(node→left);
                int rDepth = maxDepth(node→right);

                //use the larger one
                if (lDepth > rDepth)
                        return (lDepth + 1);
                else
                        return (rDepth + 1);
        }
}

/* Helper function that allocates a new node with the given data and NULL left and
right pointers. */

node* newNode(int data)
{
        node* Node = new node();
        Node→data = data;
        Node→left = NULL;
        Node→right = NULL;
        return (Node);
}

int main()
{
        node* root = newNode(1);
        root→left = newNode(2);
        root→right = newNode(3);
        root→left->left = newNode(4);
        root→left->right = newNode(5);
        cout << "Height of tree is " << maxDepth(root);
        return 0;
}
```

Exercise:

1) Write a program to calculate the size of a tree. (Size of a tree is the number of elements in a tree). Use recursion.
2) Write a program having a function to find the node with minimum value in a Binary Search Tree. (Hint: Find the in-order traversal of a tree.)
3) Write a program with a function to determine if two trees are identical or not. (Hint: Two trees are identical when they have the same data and the arrangement of data is also the same)