
# Data Structures and Algorithms
# Lab ASSIGNMENT NO 7

# SUBMITTED BY:

Hasaan Ahmad SP22-BSE-017

**SUBMITTED TO: Sir Syed Ahmad Qasim**

# Code:

```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *left;
    Node *right;
    Node(int data)
    {
        this->data = data;
        this->left = NULL;
        this->right = NULL;
    }
};
// Iterative insertion
void insert(Node *root, int data)
{
    Node *newNode = new Node(data);
    if (root == NULL)
    {
        root = newNode;
        return;
    }
    Node *temp = root;
    while (temp != NULL)
    {
        if (temp->data > data)
        {
            if (temp->left == NULL)
            {
                temp->left = newNode;
                return;
            }
            temp = temp->left;
        }
        else
        {
            if (temp->right == NULL)
            {
                temp->right = newNode;
                return;
            }
```

```
            temp = temp->right;
        }
    }
}
// Traversals
void inOrder(Node *root)
{
    if (root == NULL)
        return;
    inOrder(root->left);
    cout << root->data << " ";
    inOrder(root->right);
}
void preOrder(Node *root)
{
    if (root == NULL)
        return;
    cout << root->data << " ";
    preOrder(root->left);
    preOrder(root->right);
}
void postOrder(Node *root)
{
    if (root == NULL)
        return;
    postOrder(root->left);
    postOrder(root->right);
    cout << root->data << " ";
}
Node *insertRec(Node *root, int data)
{
    if (root == NULL)
    {
        root = new Node(data);
        return root;
    }
    if (root->data > data)
    {
        root->left = insertRec(root->left, data);
    }
    else
    {
        root->right = insertRec(root->right, data);
    }
    return root;
```

```
}

Node *search(Node *root, int key)
{
    if (root == NULL || root->data == key)
    {
        return root;
    }
    if (root->data > key)
    {
        return search(root->left, key);
    }
    return search(root->right, key);
}
// Finding Maximum and Minimum Value
Node *findMin(Node *root)
{
    if (root == NULL)
    {
        return NULL;
    }
    while (root->left != NULL)
    {
        root = root->left;
    }
    return root;
}
Node *findMax(Node *root)
{
    if (root == NULL)
    {
        return NULL;
    }
    while (root->right != NULL)
    {
        root = root->right;
    }
    return root;
}
// Write down the method to count to number of nodes and find the sum of all
nodes.
int countNodes(Node *root)
{
    if (root == NULL)
    {
```

```
        return 0;
    }
    return countNodes(root->left) + countNodes(root->right) + 1;
}
int sumOfNodes(Node *root)
{
    if (root == NULL)
    {
        return 0;
    }
    return sumOfNodes(root->left) + sumOfNodes(root->right) + root->data;
}
// Graded Lab 1
// Traversals with Right branch priority
void inOrderRight(Node *root)
{
    if (root == NULL)
        return;
    inOrderRight(root->right);
    cout << root->data << " ";
    inOrderRight(root->left);
}
void preOrderRight(Node *root)
{
    if (root == NULL)
        return;
    cout << root->data << " ";
    preOrderRight(root->right);
    preOrderRight(root->left);
}
void postOrderRight(Node *root)
{
    if (root == NULL)
        return;
    postOrderRight(root->right);
    postOrderRight(root->left);
    cout << root->data << " ";
}
// Graded Lab 2
// Write down code to print and count Leaf nodes of a BST
int printLeafNodes(Node *root)
{
    if (root == NULL)
    {
        return 0;
```

```
    }
    if (root->left == NULL && root->right == NULL)
    {
        cout << root->data << " ";
        return 1;
    }
    return printLeafNodes(root->left) + printLeafNodes(root->right);
}
int countLeafNodes(Node *root)
{
    if (root == NULL)
    {
        return 0;
    }
    if (root->left == NULL && root->right == NULL)
    {
        return 1;
    }
    return countLeafNodes(root->left) + countLeafNodes(root->right);
}
// Graded Lab 3
// Introduce method to delete a Node from BST, keep in mind that there are 3
possiblities for deletion, Node
// without any child, Node with One child and Node with both the children
Node *remove(Node *root, int key)
{
    if (root == NULL)
    {
        return NULL;
    }
    if (key < root->data)
    {
        root->left = remove(root->left, key);
        return root;
    }
    else if (key > root->data)
    {
        root->right = remove(root->right, key);
        return root;
    }
    else
    {
        // Case 1: No Child
        if (root->left == NULL && root->right == NULL)
        {
```

```
            delete root;
            return NULL;
        }
        // Case 2: One Child
        if (root->left != NULL && root->right == NULL)
        {
            Node *temp = root->left;
            delete root;
            return temp;
        }
        if (root->left == NULL && root->right != NULL)
        {
            Node *temp = root->right;
            delete root;
            return temp;
        }
        // Case 3: Two Child
        Node *replace = root->right;
        while (replace->left != NULL)
        {
            replace = replace->left;
        }
        root->data = replace->data;
        root->right = remove(root->right, replace->data);
        return root;
    }
}


main()
{
    // Testing the code
    Node *root = NULL;
    root = insertRec(root, 5);
    insertRec(root, 3);
    insertRec(root, 7);
    insertRec(root, 1);
    insertRec(root, 4);
    insertRec(root, 6);
    insertRec(root, 8);
    insertRec(root, 9);
    insertRec(root, 10);
    insertRec(root, 11);
    insertRec(root, 12);
    insertRec(root, 13);
```

```
    insertRec(root, 14);
    insertRec(root, 15);
    inOrder(root);
    cout << endl;
    preOrder(root);
    cout << endl;
    postOrder(root);
    cout << endl;
    cout << "Number of Nodes: " << countNodes(root) << endl;
    cout << "Sum of Nodes: " << sumOfNodes(root) << endl;
    cout << "Number of Leaf Nodes: " << countLeafNodes(root) << endl;
    cout << "Leaf Nodes: ";
    printLeafNodes(root);
    cout << endl;
    cout << "Inorder with Right Branch Priority: ";
    inOrderRight(root);
    cout << endl;
    cout << "Preorder with Right Branch Priority: ";
    preOrderRight(root);
    cout << endl;
    cout << "Postorder with Right Branch Priority: ";
    postOrderRight(root);
    cout << endl;
    cout << "Minimum Value: " << findMin(root)->data << endl;
    cout << "Maximum Value: " << findMax(root)->data << endl;
    cout << "Searching for 7: " << search(root, 7)->data << endl;
    cout << "Searching for 17: " << search(root, 17) << endl;
    cout << "Deleting 7: ";
    root = remove(root, 7);
    inOrder(root);
    cout << endl;


    return 0;
}
```

# Output:

```
1 3 4 5 6 7 8 9 10 11 12 13 14 15
5 3 1 4 7 6 8 9 10 11 12 13 14 15
1 4 3 6 15 14 13 12 11 10 9 8 7 5
Number of Nodes: 14
Sum of Nodes: 118
Number of Leaf Nodes: 4
Leaf Nodes: 1 4 6 15
Inorder with Right Branch Priority: 15 14 13 12 11 10 9 8 7 6 5 4 3 1
Preorder with Right Branch Priority: 5 7 8 9 10 11 12 13 14 15 6 3 4 1
Postorder with Right Branch Priority: 15 14 13 12 11 10 9 8 6 7 4 1 3 5
Minimum Value: 1
Maximum Value: 15
Searching for 7: 7
Searching for 17: 0
Deleting 7: 1 3 4 5 6 8 9 10 11 12 13 14 15
PS D:\Ishtudy Material\3rd Sem\DSA\LAB\LAB 08>
```