National University of Computer and Emerging Sciences



# Laboratory Manual

*for*

# Data Structures Lab

|  |  |
| --- | --- |
| Course Instructor | Samman Ashraf |
| Lab Instructor(s) | Muhammad Ahmed  Hamza Ayub |
| Section | 4N |
| Semester | Spring 2023 |

**Department of Computer Science**

FAST-NU, Lahore, Pakistan

## Objectives:

In this lab, students will practice:

1. Binary Search Trees
2. Recursive insert operation, recursive delete operation, and inorder traversal, and some other recursive operations on BST

# Binary Search Tree(BST)

In this tutorial, you will learn how Binary Search Tree works. Also, you will find working examples of Binary Search Tree in C, C++, Java and Python.

Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.

* It is called a binary tree because each tree node has a maximum of two children.
* It is called a search tree because it can be used to search for the presence of a number in O(log(n)) time.

The properties that separate a binary search tree from a regular [binary tree](https://www.programiz.com/data-structures/trees) is

1. All nodes of left subtree are less than the root node
2. All nodes of right subtree are more than the root node
3. Both subtrees of each node are also BSTs i.e. they have the above two properties

## Question 1

Implement the following Tree Node:

struct Node

{

int data; Node \*left; Node \*right;

};

Now implement a binary search tree class “BST” which contains root of type **Node** as data member.

class BST

{

Node\* root;

};

You have to implement the following member functions for your binary search tree:

1. A default Constructor which sets the root to nullptr.
2. A recursive “insert” function which is passed as parameter **int data**. It should insert the data while considering the insertion rules. If the data already exists in the BST, simply return false and true otherwise.

bool insert(int d)

1. A copy constructor which uses recursion to deep copy another Binary Search Tree object.
2. A function “inorderPrint” which prints the keys using in-order traversal.

void inorderPrint () const

1. A function “search” which is passed as parameter a key. The function then uses recursion to return pointer to the corresponding node. If the key does not exist, the function returns nullptr. Node\* search(int key)
2. A function “length” which uses recursion to return the count of total nodes in BST.

int length() const

1. A “deleteKey” function which is passed as parameter a key. The function then uses recursion to delete the node that contains that key.

void deleteKey(int key)

1. Destructor

Your MAIN FUNCTION:

void main()

{

BST<int> tree;

tree.insertRec(500);

tree.insertRec(1000);

tree.insertRec(1);

tree.insertRec(600);

tree.insertRec(700);

tree.insertRec(10);

tree.insertRec(30);

tree.insertRec(9000);

tree.insertRec(50000);

tree.insertRec(20);

cout << "Printing datas using recursive inorder traversal: ";

tree.inorderPrintdatasRec();

cout << "\nCOPY CONSTRUCTOR\n";

BST<int> tree1(tree);

tree1.inorderPrintdatasRec();

cout << endl << endl << "Tree Length: " << tree.length() << endl << endl;

cout << "DELETE KEY\n";

tree1.deleteKey(9000);

tree1.inorderPrintdatasRec();

cout << "\nSEARCH:";

if (tree.search(1))

{

cout << "\nFound" << endl;

}

else

cout << "NOT FOUND\n";

if (tree.search(123))

{

cout << "Found" << endl;

}

else

cout << "NOT FOUND\n";

cout << "Length of Tree: " << tree.length() << "\n";

cout << endl;

system("pause");

}