

Name:

Usama Iftikhar Butt

Roll No.

Roll no

Class:

CS-17

Assignment:

Binary Search Trees

Submitted to:

Draw this series in BST Form:

4, 11, 7, 6, 8, 5, 4, 10

4

10

8

4

5

6

7

11

Degree of Nodes:

Degree of **4** is **1** Degree of **11** is **1** Degree of **7** is **2**

Degree of **6** is **1** Degree of **5** is **1** Degree of **4** is **0**

Degree of **8** is **1** Degree of **10** is **0**

Level of Tree:

Level of tree is **6**

Height of Tree:

Height of tree is **5**

Parents Nodes:

Parent nodes are **4**, **11**, **7**, **6**, **8** and **5**

Leaf Nodes:

Leaf Nodes are **4** and **10**

Root Node:

Root Node is **4**

Draw this series in BST Form:

12, 15, 7, 9, 20, 18, 15, 1

12

18

1

15

9

15

20

7

Degree of Nodes:

Degree of **12** is **2** Degree of **7** is **2** Degree of **15** is **1**

Degree of **1** is **0** Degree of **9** is **0** Degree of **20** is **1**

Degree of **18** is **1** Degree of **15** is **0**

Level of Tree:

Level of tree is **5**

Height of Tree:

Height of tree is **4**

Parents Nodes:

Parent nodes are **12**, **15**, **7**, **20** and **18**

Leaf Nodes:

Leaf Nodes are **1, 9** and **15**

Root Node:

Root Node is **12**

Draw this series in BST Form:

22, 16, 12, 15, 11, 17

22

16

11

15

12

17

Degree of Nodes:

Degree of **22** is **1** Degree of **16** is **2** Degree of **12** is **2**

Degree of **17** is **0** Degree of **11** is **0** Degree of **15** is **0**

Level of Tree:

Level of tree is **4**

Height of Tree:

Height of tree is **3**

Parents Nodes:

Parent nodes are **22**, **16** and **12**

Leaf Nodes:

Leaf Nodes are **17, 11** and **15**

Root Node:

Root Node is **22**

**Algorithms**

Insertion in BST

We use a recursive function to insert a data into Binary Search Tree.

Declare **Insert** function having parameters **Data** and **Root** andreturn type pointer to Node

If **Root** = **NULL**

Create new node and insert the **Data**

Else if Data < Data of Root

Left child of Root Node = Call Insert function pass Data and Left child of Root

Else

Right child of Root Node = Call Insert function pass Data and of Right child of Root

Return Root

Search in BST

We use a recursive function to search in BST

Declare a function **Search** having parameters **Root** (Node\*) and return type **void**

If **Root** = **NULL**

Print “data not found”

Else if **Data** < **Data of Root**

Call Search function pass Left child of Root

Else if **Data** > **Data of Root**

Call Search function pass Right child of Root

Else

Print “Data found”

Traversing in BST

There are three types of traversing in BST

* Preorder Traversal
* Inorder Traversal
* Postorder Traversal

We use recursive functions to implement all these Traversals.

Preorder Traversal

Declare an function **Preorder** havingparameters Root (Node\*) and return type void

Print Data in Root Node

Call Preorder function the pass Left child of Root

Call Preorder function the pass Right child of Root

Inorder Traversal

Declare an function **Inorder** havingparameters Root (Node\*) and return type void

Call Preorder function the pass Left child of Root

Print Data in Root Node

Call Preorder function the pass Right child of Root

Postorder Traversal

Declare an function **Inorder** havingparameters Root (Node\*) and return type void

Call Preorder function the pass Left child of Root

Call Preorder function the pass Right child of Root

Print Data in Root Node

Deletion in BST

Declare an function Delete parameters **Data** and **Root** (Node\*) having the return type **\*Node**

If **Root** == **NULL**

Return (Root)

Else if **Data** < **Data** in **Root**

**Left** Child of **Root** = Call Delete Node send **Data** and **Left** Child of **Root**

Else if **Data** > **Data** in **Root**

**Right** Child of **Root** = Call Delete Node send **Data** and **Right** Child of **Root**

Else

If **Left** and **Right** child of **Root** are **Null**

Delete (**Root**)

**Root** = **NULL**

Else if **Left** child of **Root** == **NULL**

Declare **temp** save **Root** address

**Root** = **Right** child of **Root**

Delete (Temp)

Else if **Right** child of **Root** == **NULL**

Declare **temp** and initialize it as **Root**

**Root** = **Left** child of **Root**

Delete (Temp)

Else

**Data** of **Root** = Minimum in **Right** sub tree of **Root**

**Right** child of **Root** = Call **Delete** pass **Data** of Root and **Right** child of Root

Return (Current)