Binary search Trees or Sorted or Ordered Binary trees are those who store items in “memory”. Binary search trees can have max 2 children namely left child and right child. There can be either child or no child for parent node.

The node that has no children is leaf node.

Binary Search Trees has faster insertion and lookup as it is sorted and in-memory.

Let us take the above example.

|  |  |
| --- | --- |
| **Insert Data** | **Trace down in tree to insert node.** |
| Insert 8 | No root. 8 is new root. |
| Insert 3 | 3 < 8. 3 is left child of 8. |
| Insert 1 | 1 < 8 go left, 1 < 3 go left. Now 1 is left child of 3. |
| Insert 10 | 10 > 8, go right. 10 is right child of 8. |
| Insert 6 | 6 < 8, go left. 3 < 8 go right. 6 is right child of 3. |
| Insert 4 | 4 < 8. Go left. 3 < 4 go right. 4 < 6 go left. 4 is left child of 6 |
| Insert 14 | 14 > 8 go right. 14 > 10 go right. 14 is right child of 10. |
| Insert 13 | 13 > 8 go right. 13 > 10 go right. 13 < 14 go left. 13 is left child of 14. |
| Insert 7 | 7 < 8 go left. 7 > 3 go right. 7 > 6 go right. 7 is right child of 6 |

So above binary tree is of size 9 depth 3 with 8 as root.

Leaf nodes are 1, 4, 7, 13.

There are many types of Binary tree such as AVL tree, Red-Black tree, Splay Tree, Tango Tree. We will now learn how to create structure of the tree and insert nodes.

The node in a tree consists of 3 elements:

Data

Left Child link

Right Child link

**public** **class** Node

We declare a new class called **Node**.

Now taking one step further we built 3 data member for a node.

**public** **int** iData;

**public** Node rightChild;

**public** Node leftChild;

|  |  |
| --- | --- |
| **Data Member** | **Description** |
| iData | Contains integer value. |
| Node leftChild | Points to left child of the current node. |
| Node rightChild | Points to right child of the current node. |

We will also define one constructor that will that value as parameter.

**public** Node(**int** a) {

iData = a;

}

Now let us write entire Node class all together.

**public** **class** Node {

**public** **int** iData;

**public** Node rightChild;

**public** Node leftChild;

**public** Node(**int** a) {

iData = a;

}

**public** **void** displayNode() {

System.***out***.println(" { " + iData + " } ");

}

}

Now let us write code to insert data into the tree. Below code is heavily commented. So you will have no problem in tracing it.

**public** **void** insert(**int** id) {

/\*\*We get a value to be inserted in the tree

make a new Node for it using Node(int) constructor.\*\*/

Node newNode = **new** Node(id);

/\*\*Base condition if root node does not exists.\*/

**if** (root == **null**)

root = newNode;

**else** {// root occupied

/\*\*We don't want to move root pointer so we save it in

\* current and move current instead.

\* We also use parent pointer to save parent.

\* \*/

Node current = root;

Node parent;

**while** (**true**) {

/\*\*Save current node in parent pointer\*/

parent = current;

/\*\*if input value is less than current value then we need

\* to traverse to left sub-tree.\*/

**if** (id < current.iData) {

/\*\*Traverse to leftchild.\*/

current = current.leftChild;

/\*\*If left child is now null so we found out place.

\* Insert our node as parent's leftchild.\*/

**if** (current == **null**) {

parent.leftChild = newNode;

**return**;

}

}

/\*\*if input value is greater than current value then we

\* need to traverse to right sub-tree\*/

**else** {

/\*\*Traverse to rightchild.\*/

current = current.rightChild;

/\*\*If left child is now null so we found out place.

\* Insert our node as parent's rightchild.\*/

**if** (current == **null**) {

parent.rightChild = newNode;

**return**;

}

}

}

}

}