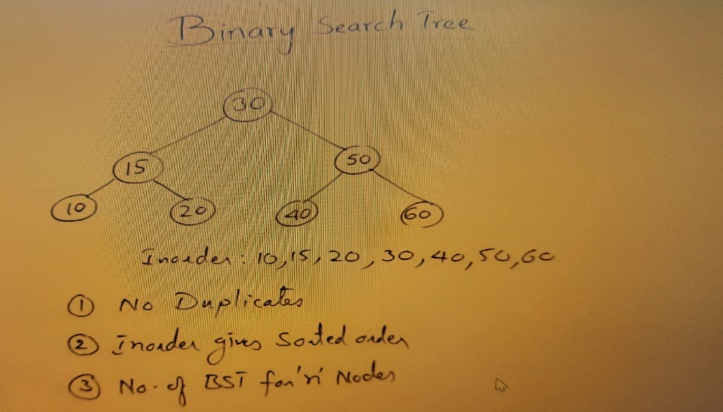
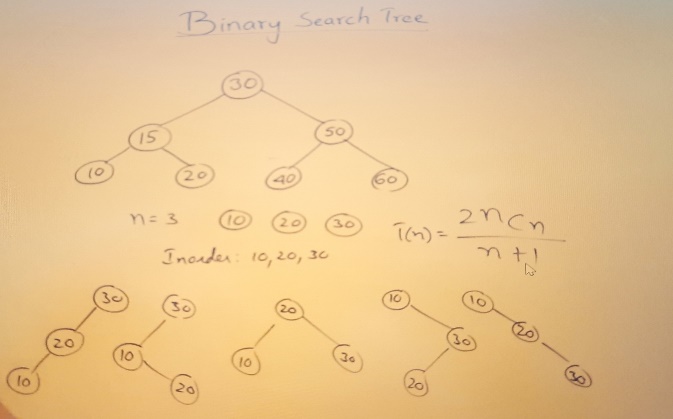
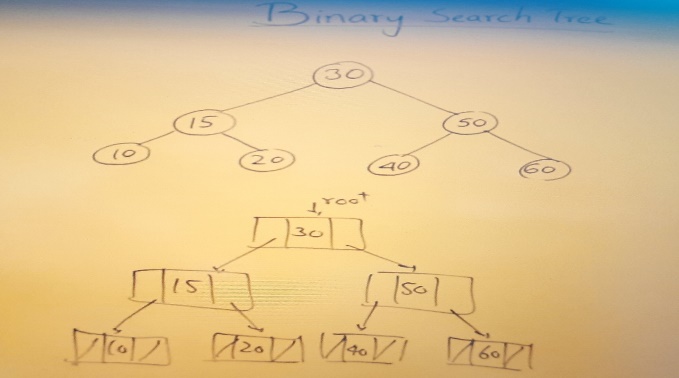
This binary tree is useful for searching that what the naming convention as BST. The left subtree node will be smaller, and the right subtree node will be greater than the root node. So, this is more like binary search.

* **This is useful for searching an element in less number of comparisons.**
* Binary search is applied on a single array in a list of elements. But this is upon binary tree. So, the behavior is similar to binary search.
* So, the search time depends on the height of a tree.
* BST will not have duplicates.
* **If we take In Order traversal of Binary Search Tree, we will get the list of elements in sorted order.**
* If we have some nodes, then how many different binary search trees can be created for the same set of nodes.

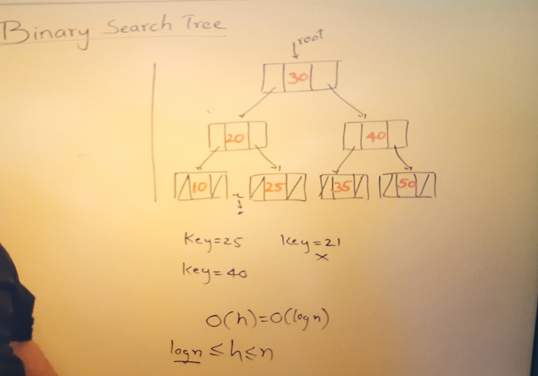


**Let’s us see how we can represent the BST**

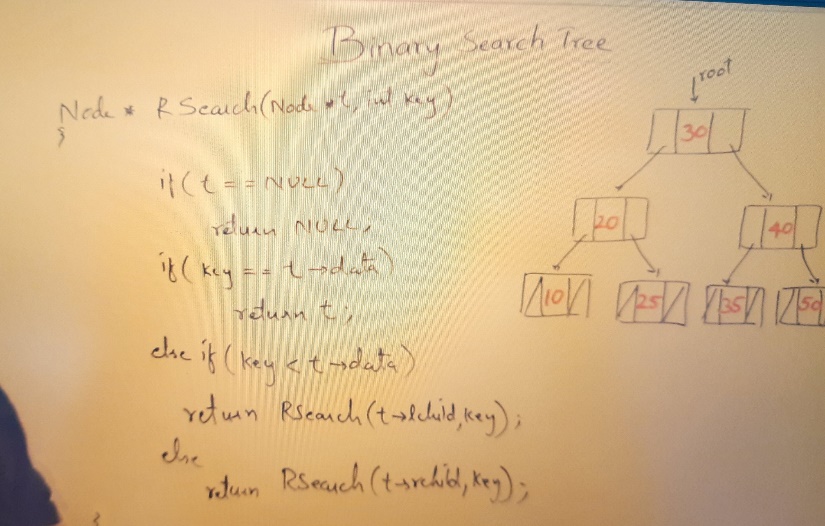
BST represented using Linked representation.



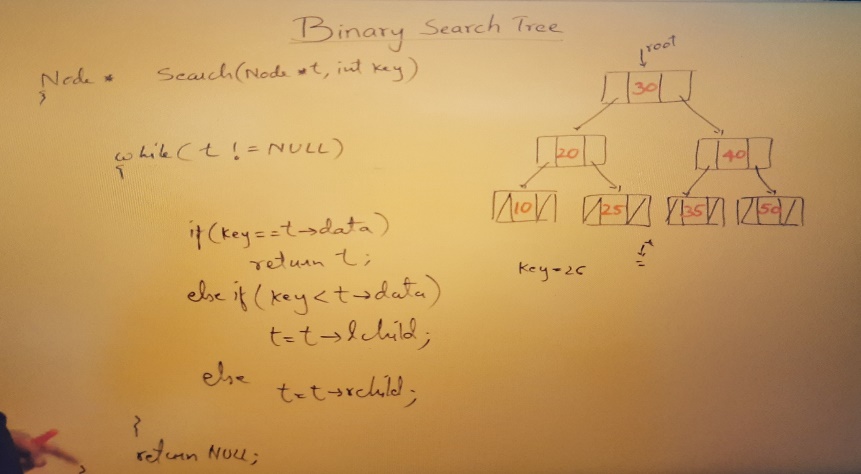
* **Searching in a Binary Search Tree**



* Searching in a BST using Recursion—



* Searching in a BST using Iterative approach---

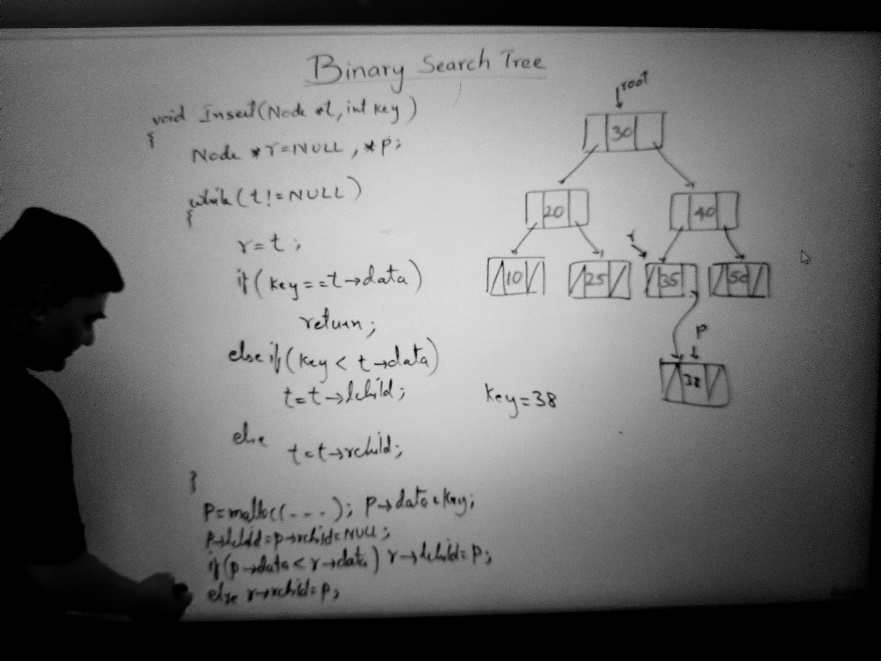


* **Inserting in a Binary Search Tree**

The first step for insertion is search for key value 38. If this 38 is already present in binary search, then we should not insert it. So, first of all, find out whether it is there or not there so far that we have to search. So, the first step is search.

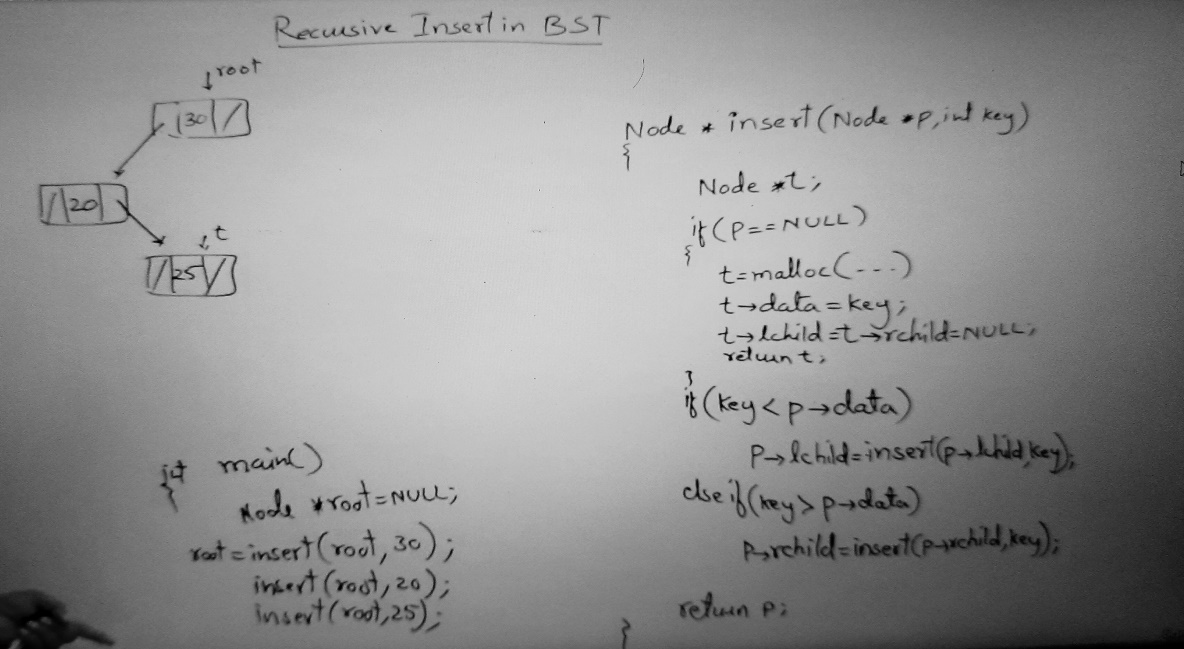
**So here for insertion we need 2 pointers one pointer t for searching and another tailing pointer r for pointing to the new inserted position node.**

P is a newly created node pointing, which is going to be inserted.



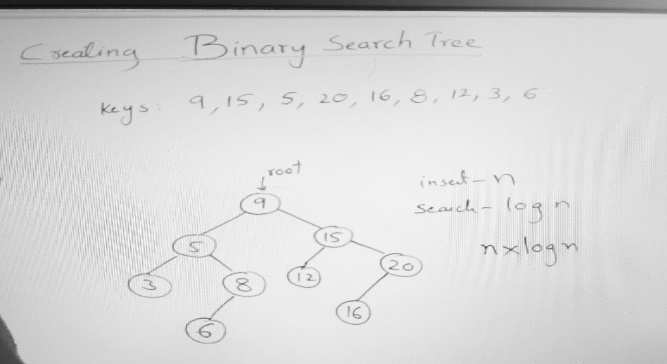
So, insertion is more like searching only. So, the **time taken for inserting is O(log n)** only. That is the height of a binary search Tree.

* **Recursive Insert in a Binary Search Tree**

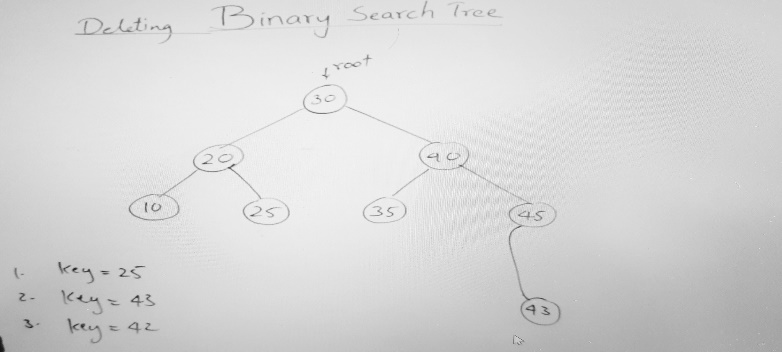


* **Creating a Binary Search Tree**

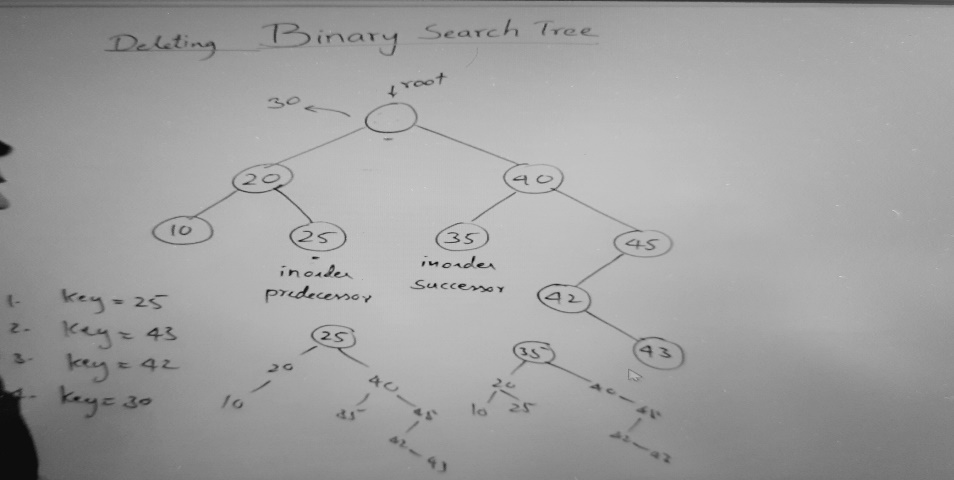
What is the time taken for creating a binary search tree. see we have inserted n elements. We have inserted n elements. Then each element we were searching and finding its position and inserting it, each element, so searching takes how much time searching takes **log n** time. We are assuming that the height of a tree is **log n,** so search takes log n times. So, searching we have done for every element so, n times we have searched so time taken is **n \* log n.** creating a node and linking that will take constant time.



* **Deleting from Binary Search Tree**



1. If you are deleting a key **25** so the procedure is search for a key 25 if found, then delete it. If u delete this node, then u have to make this parent link as null, and this node will be deleted.
2. If we want to delete a key **43** so, search for 43 so delete 43 and make this right child link as null.
3. If I want to delete a key **42** so search for 42 now the problem occurs so, if I delete **42** then I need to modify the link of parent. But **42** is also having a child. So, since it is having only one child so this node will deleted and this 43 child will take its place that’s become a left child of 45.

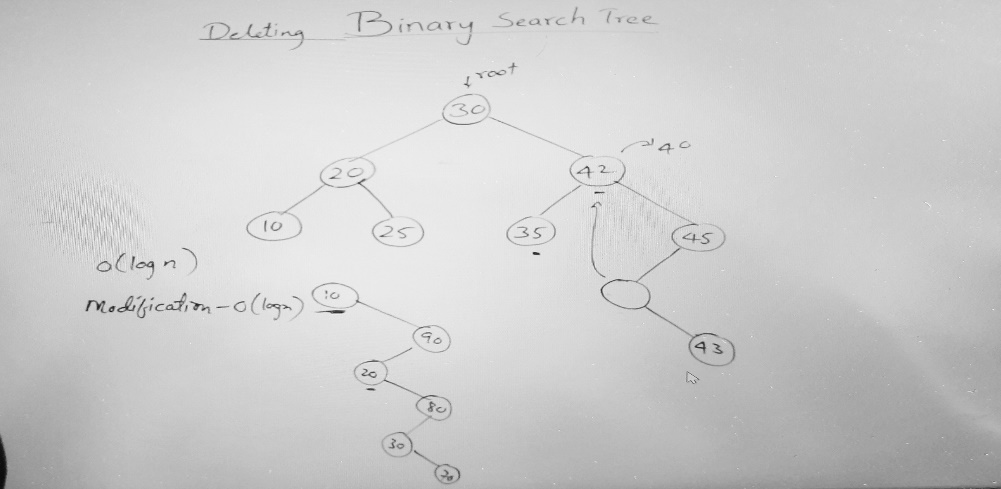


Now what if I want to delete **30** then after deletion who will take place 20 or 40 then who will take the place of 20 or 40. So, in this case so many changes we have to made.

Instead of doing this way the other way is for 30 find out **In Order Predecessor** . which means if u perform in order, then which node will come before 30. So, 25 come before 30. This is in order predecessor. Then who will **In Order Successor** which means who will come after 30 so 35 is in order successor.

\*\*\* So, if you are deleting 30 then either In Order Predecessor will take place or In Order Successor will take place. So, leaf node will come at that place so no need to disturb so many nodes. So, the result of this tree can be either 25 or 35 will take that place.

\*\*\* So how to find in order predecessor or successor for in order predecessor for deleting a node go to its left subtree and then go to right, right, right. So, Right most child of a left subtree and in order successor means Left most child of a Right subtree



Suppose I want to delete 40 then after deleting 40 who will take this place. So, in order predecessor 35 and in order successor 42. We were taking in order predecessor or successor because we wanted to avoid multiple modifications in the tree. So, if the in order predecessor and successor is not a leaf node, then you may have to make multiple modifications. So, time taken is **O(log n)** and number of modifications would be **log n**.

