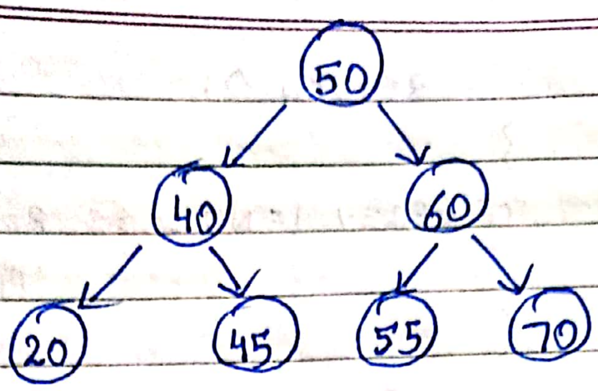
Searching in a Binary Search Tree

→ What exactly is the significance of learning about Binary Search trees?

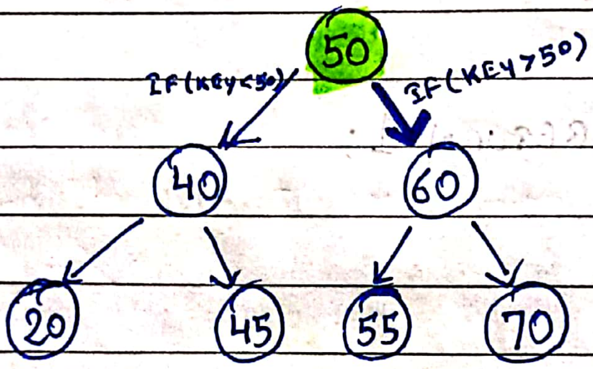
→ One of the major application of using a binary search tree, is to be able to search some key in the tree in log(n) time complexity in the best case where n is the number of nodes. Each time you compare a node with your key, you divide the search space to its half.

→ Suppose we have a Binary Search tree:

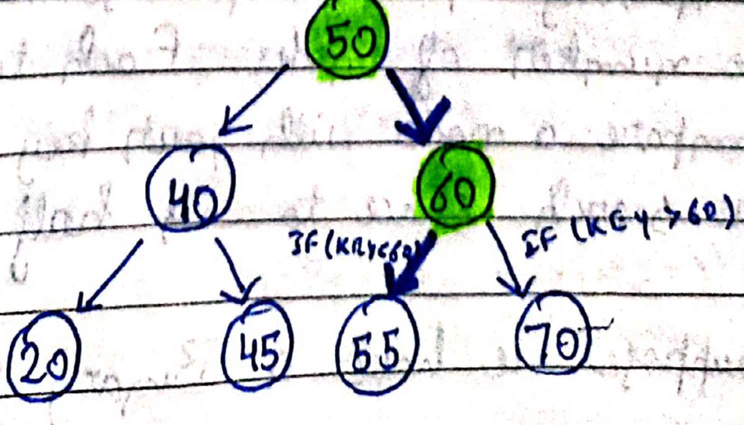
Date



And let's say the key we want to search in this binary search tree is 55. So, we'll first compare our key with the root node itself, which is 50.

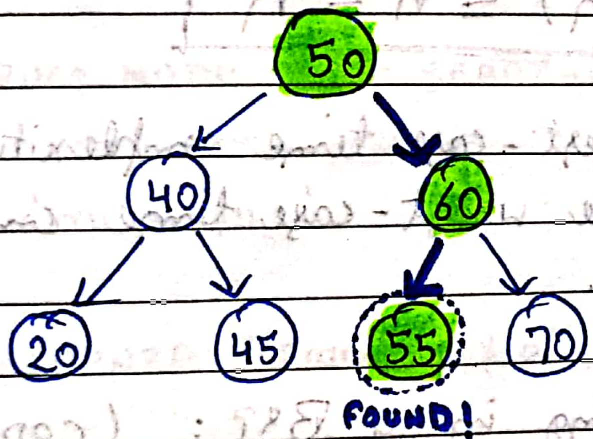


But since 50 is less than 55, ~~which side~~ ∴ We will proceed towards right. Since all the elements in the right subtree of a node are greater than that node, we'll move to the right. The first element we check our key with is 60.



Page No. ____
Date

Now, since our key is smaller than 60, we'll move to the left of the current node. The Left subtree of 60 contains only one element and since that is equal to our key, we revert the positive result that yes, the key was found. Had this leaf node been not equal to the key, and since there are no subtrees further, we would have stopped here itself with negative results, saying the key was not found.



→ Time complexity of the Search Operation in a Binary Search Tree:

Searching in a binary search tree holds O(log n) time complexity in the base case where n is the number of nodes making it incredibly easier to search an element in a binary search tree, and even operations like inserting get relatively easier.

Let's calculate exactly what happens. If you could see the above examples, the

Page No.

Date

algorithm took the number of comparisons equal to the height of the binary search tree, because at each comparison we stepped down the depth by 1. So, the time complexity $T \propto h$, that is, our time complexity is proportional to the height of the tree. Therefore, the time complexity becomes $O(h)$.

And the height of a tree ranges from log n to n, that is:

$$\boxed{(\log n) \le h \le n}$$

∴ The best-case time complexity is $O(\log n)$ and the worst-case time complexity is $O(n)$.