29.21 Binary Search Tree

In simple K-NN implementation,

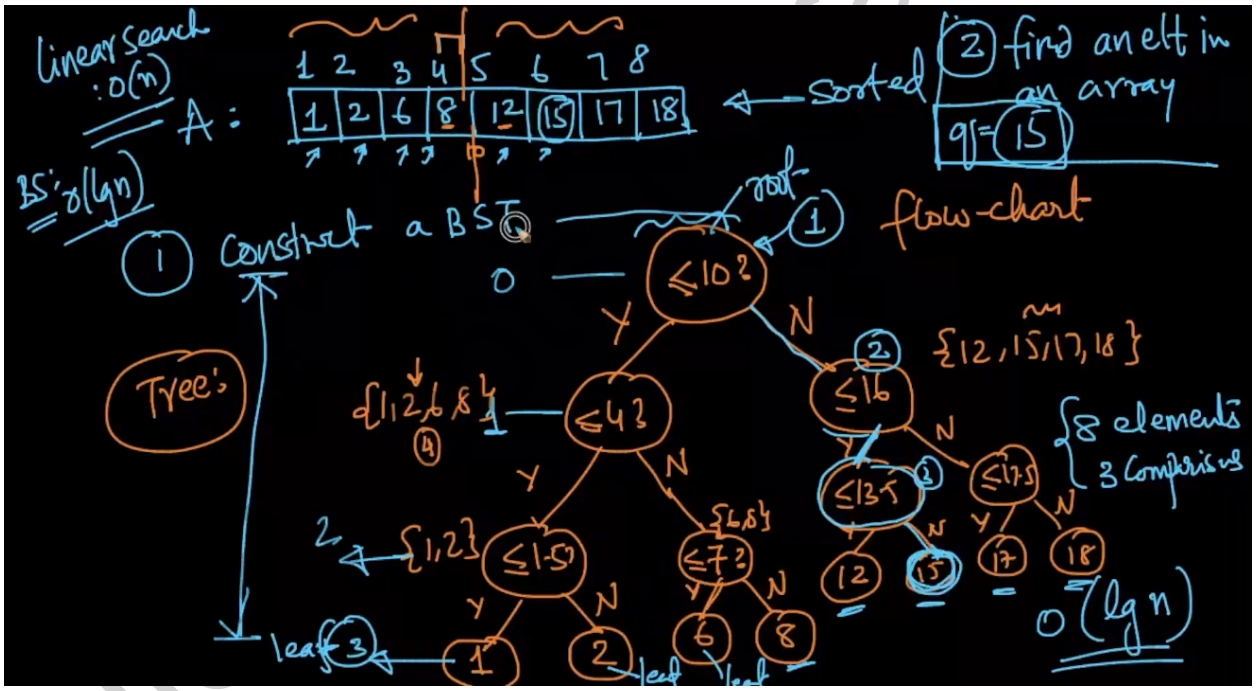
Time Complexity = O(nd) ~ O(n) (if both 'K' and 'd' are small)

Space Complexity = O(nd) ~ O(n) (if 'd' is small)

We could not reduce the space complexity of KNN, but can reduce the time complexity using a technique called KD-Tree. The time complexity reduces from O(n) to O(log(n)). For example, if there are 1024 computations to be performed, then the time complexity of simple KNN would be O(1024) whereas in case of a KD-Tree, the time complexity reduces to O(log(n)) = O(log(1024)) = O(10)

KD-Tree works similar to the way how a Binary Search Tree (BST) Works. Let us now look at how a Binary Search Tree works.

Construction and Working of a Binary Search Tree



Steps of Construction

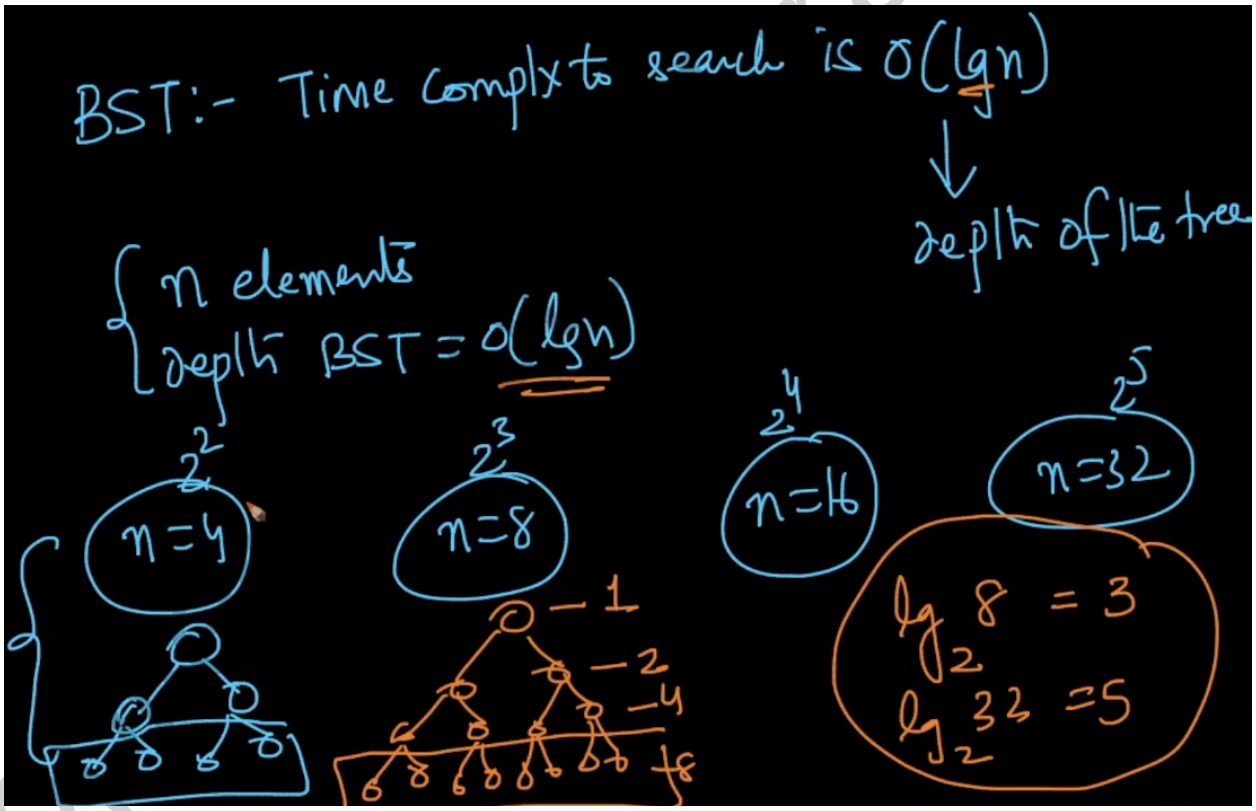
1) We have to first sort the given array/list of values in ascending order.
2) Find out the median of the values and make it a root.
3) All the values that are less than the median should go to the left side(ie., left subtree) of the root, and all the values that are greater than the median should go to the right side(ie., right subtree) of the root.

4) Repeat steps 2 and 3, until we are left with a single value in each node. All these nodes present in the bottom layer without any child nodes, are called the leaf nodes.

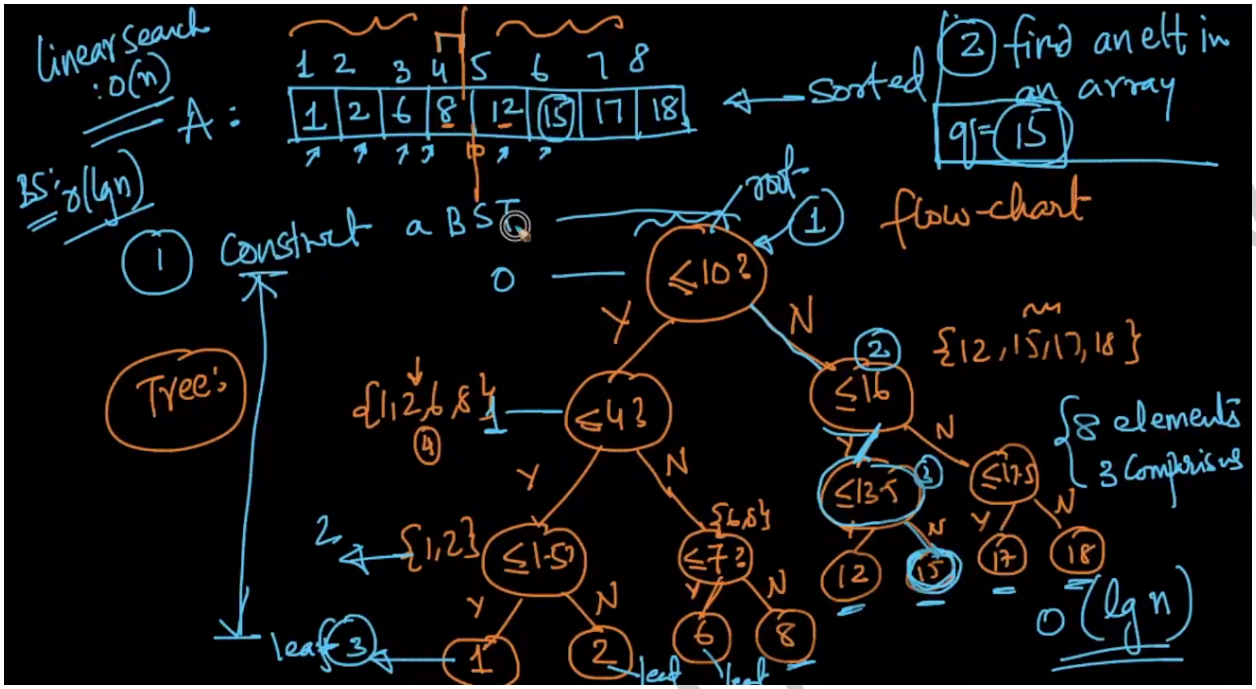
Steps for Searching an element

1) If an element is given, we have to compare it with the value in the root node first.
2) If the element is the same as the value in the root node, return True. If the element is less than the root node value, then move into the left subtree. Otherwise move into the right subtree.
3) Repeat steps 1 and 2, either until the given element is found or all the leaf nodes in the traversed path are also checked. If the given element is not found, return False.

Here as we are first sorting the values and then checking only a portion of the tree, rather than the entire tree. As we are just checking only a half portion in every subtree, the time complexity reduces from O(n) to O(log(n).



Let us check if the numbers 2, 5, 6, 16, 19 are present in the Binary Search Tree.



Checking for Number 2

a) The root value is 10. Comparing '2' with '10'. As 2<10, we have to check the left subtree, with '4' as the root.
b) The root value is now 4. Comparing '2' with '4'. As 2<4, we have to check the left subtree, with '1.5' as the root.
c) Now the root value is 1.5. Comparing '2' with '1.5'. As 2>1.5, we have to check the right subtree, with '2' as the root.
d) Now the root value is 2. As 2==2 returns **True**, the whole result turns out to be **True**. It means the specified element '2' is present in the Binary Search Tree.

Checking for Number 5

a) The root value is 10. Comparing '5' with '10'. As 5<10, we have to check the left subtree, with '4' as the root.
b) The root value is now 4. Comparing '5' with '4'. As 5>4, we have to check the right subtree, with '7' as the root.
c) The root value is now 7. Comparing '5' with '7'. As 5<7, we have to check the left subtree, with '6' as the root.
d) The root value is now 6. Comparing '5' with '6'. As 5<6, we have to check the left subtree. But here '6' is a leaf node and it has no more subtrees/branches. So as we couldn't find the element '5', it returns **False**.

Checking for Number 6

a) The root value is 10. Comparing ‘6’ with ‘10’. As 6<10, we have to check the left subtree, with ‘4’ as the root.
b) The root value is now 4. Comparing ‘6’ with ‘4’. As 6>4, we have to check the right subtree, with ‘7’ as the root.
c) The root value is now 7. Comparing ‘6’ with ‘7’. As 6<7, we have to check the left subtree, with ‘6’ as the root.
d) Now the root value is 6. As 6==6 returns **True**, the whole result turns out to be **True**. It means the specified element ‘6’ is present in the Binary Search Tree.

Checking for Number 16

a) The root value is 10. Comparing ‘16’ with ‘10’. As 16>10, we have to check the right subtree, with ‘16’ as the root.
b) Now the root value is 16. As 16==16 then we move to the left subtree of node 16. As the root node value of the left subtree is 13.5 which is less than 16. So we return **False.**

Checking for Number 19

a) The root value is 10. Comparing ‘19’ with ‘10’. As 19>10, we have to check the right subtree, with ‘16’ as the root.
b) The root value is now 16. Comparing ‘19’ with ‘16’. As 19>16, we have to check the right subtree, with ‘17.5’ as the root.
c) The root value is now 17.5. Comparing ‘19’ with ‘17.5’. As 19>17.5, we have to check the right subtree, with ‘18’ as the root. But here ‘18’ is a leaf node and it has no more subtrees/branches. So as we couldn’t find the element ‘19’, it returns **False**.