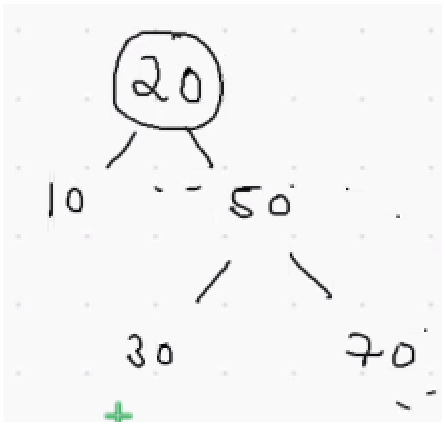
BST – Binary Search Tree

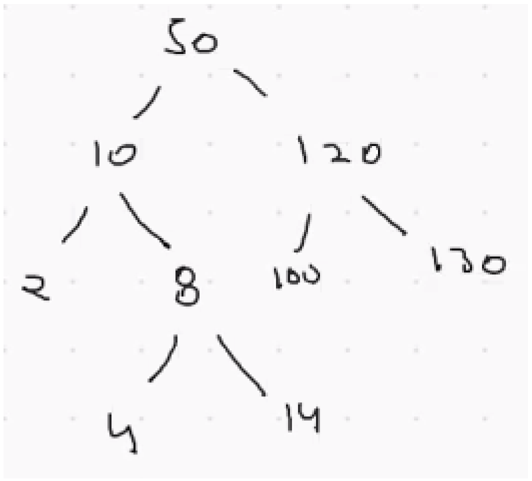
It is a tree with special property; ***all the nodes right side of root are greater than root and all nodes at left side are smaller than root.***

Left < Root < Right

This property should be present in the root recursively.



All elements on the right are greater than its root and the all the values on the left are smaller than the root.



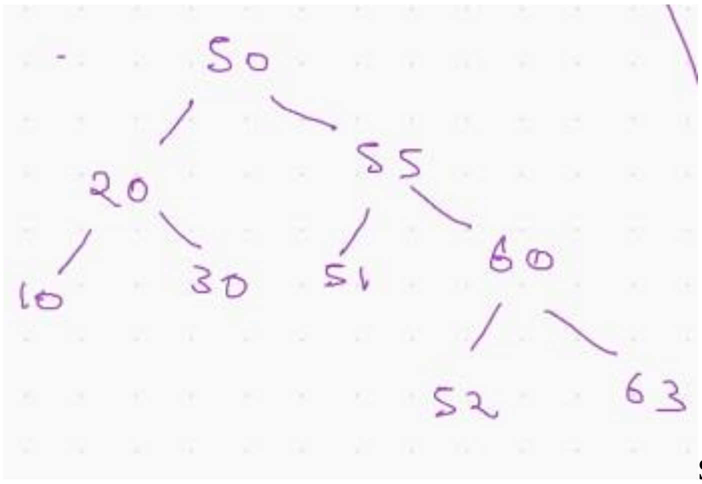
Since 8 < 10, it is not a BST

BST is used by dict() internally. In a dict(), the searching time complexity is O(logn). When the keys are small, then it is O (1), but if it is a huge dictionary, then it would be O (logn).

NOTE: Dictionary is a balance BST.

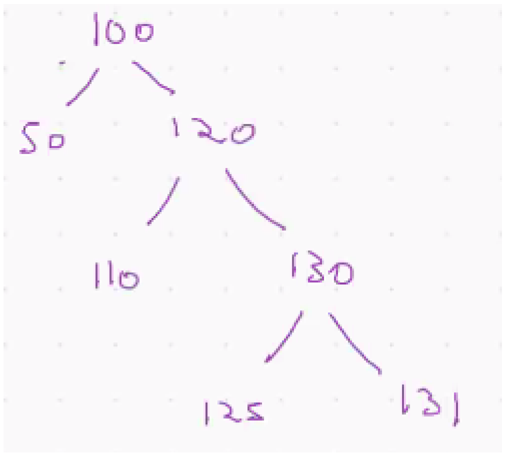
Properties of BST:

- All the values on the right should be greater than the root, recursively and on the left should be less than the root, recursively.
- In-order traversal of any BST would be sorted.



since 52 < 55 and 52 is on the right side, this is not a BST

In the below BST, how will you find the value 130.



```
def find(root):
    if root is None:
        return
    if root.val == target:
        return True
    elif find (root→left)
    elif find (root→right)
```

If we are at 100 and we are searching for 130, then we will go to right of 100 as it is a BST. So, we will search on the right of the root = 100.

At 120, we will further search towards the right side as 120 < 130.

For 125, we will go towards right from 120 and then at 130, since 125 < 130 we will check towards the left of the root.

CODE:

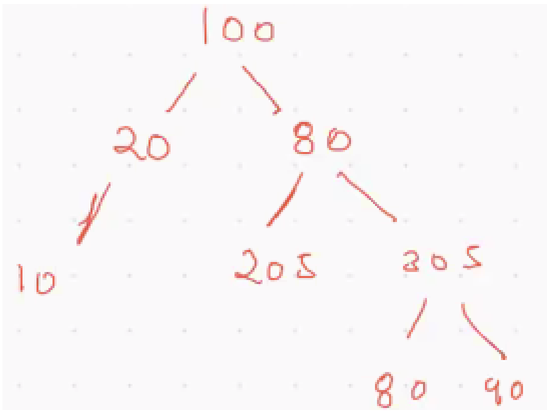
```python
def find(root, target):
    if root is None:
        return False

    if root.val == target:
        return True

    if root.val < target:
        return find(root.right, target)
    else:
        return find(root.left, target)
```

So, the time complexity for this program would be $O\ (\log_2 n)$

NOTE: In a skew tree, the BST Time complexity would be O(n).

Balanced BST:

A BST where the absolute value of left height – right height <= 1 recursively, such time of tree is called Balanced BST.



For 100, left height = 2 and right height = 3. The difference is 1

For 20, left height = 1 and right height = 0. The difference is 1

For 10, left height = 0 and right height = 0. The difference is 0

For 80, left height = 1 and right height = 2. The difference is 1

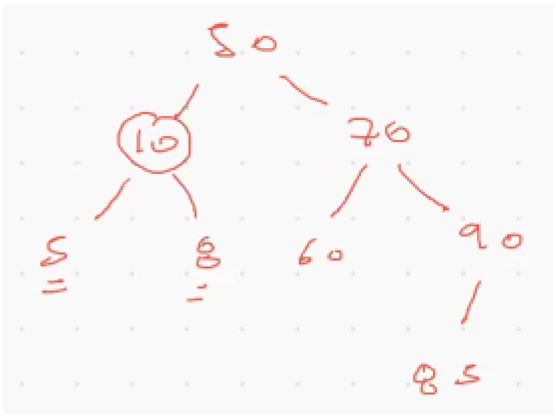
For 205, left height = 0 and right height = 1. The difference is 1

For 305, left height = 1 and right height = 1. The difference is 0

For 80 & 90, left height and right height are 0. The difference is 0

So, this is a balanced tree.

Give a BST find the kth smallest element in it.



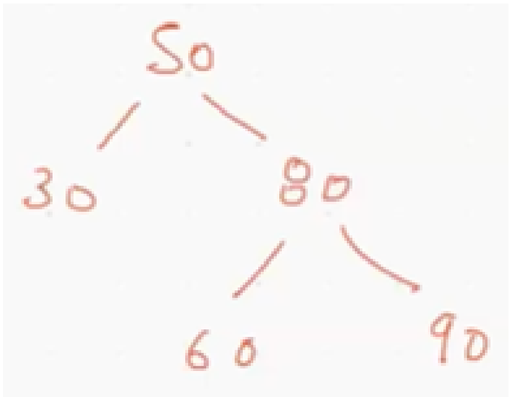
Do a preorder traversal, then sort the list, and then print k-1 index value, TC = O(nlogn).

Do an in-order traversal, to get a sorted list and then find the k-1 index. TC = O(n)

How to get this done, without using an array?

```
counter = 0

def findKthSmallestValue(root, k):
    if root is None:
        return None
    if counter = k:
        return root.val
    findKthSmallestValue(root.left, k)
    # logic
    coutner += 1
    findKthSmallestValue(root.right, k)
```

This is what we need to do. we can have a counter as global counter and increase it.



In stack we will be at root = 50, counter = 0.

At stack we will be at root = 30, counter = 1.

Since there is no element on right so return 30 and return to 50 and the counter will become 2.

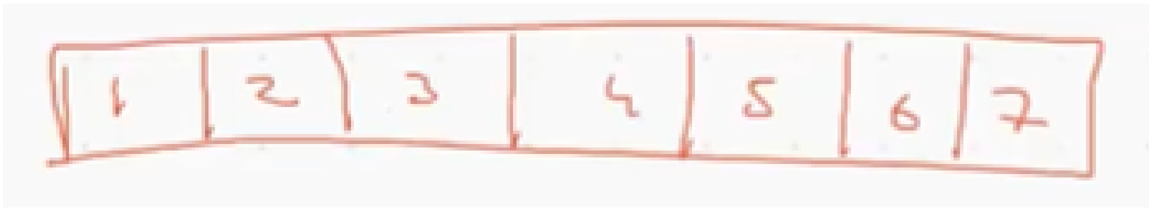
Now, go to 80 and counter = 2, then towards left = 60 with counter = 2. Its left is none so will return to 60 and counter = 3.

At 60, k = 3 do we will return the value of k. As this is in recursion, the time complexity is O(n).

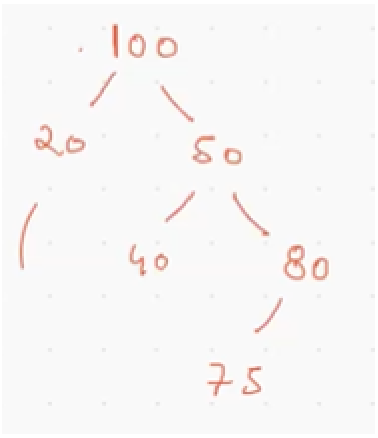
In an indorder traversal, at 30, counter will be 1,

At 30, counter = 1
At 50, counter = 2
At 60, counter = 3
At 80, counter = 4
At 90, counter = 5

Instead of counter == k, we can say counter == n – k.



The 3rd largest element, 7 – 3 = 4.



```python
counter = 0

def solve(root,k):
    if root is None:
        return None

    solve(root.left, k)

     # logic
    counter += 1

    solve(root.right, k)
```

https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/

https://www.geeksforgeeks.org/data-structure-gq/binary-search-trees-gq/