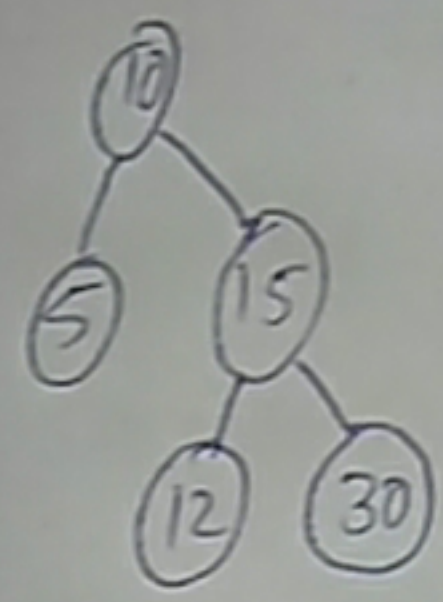
Day 25

floor in binary search tree

input : x = 14 and root of the below tree



-----) output

12

* A simple solution is to traverse a tree using (inorder, preorder, postorder) and keep track closet smaller or same element.

* Time Complexity of this solution - O(n)

Implementation :

```
class Node :

    def __init__(self, data):
        self.data = data
        self.right = None
        self.left = None
```

```
def insert(root, key):
    if (not root):
        return int_max
    if (root.data == key):
        return root.data
    if (root.data > key)
        return floor(root.left, key)
    return
    floor value = floor(root.right, key)
    return floorvalue if floorvalue <= key else
                                        root.data
```

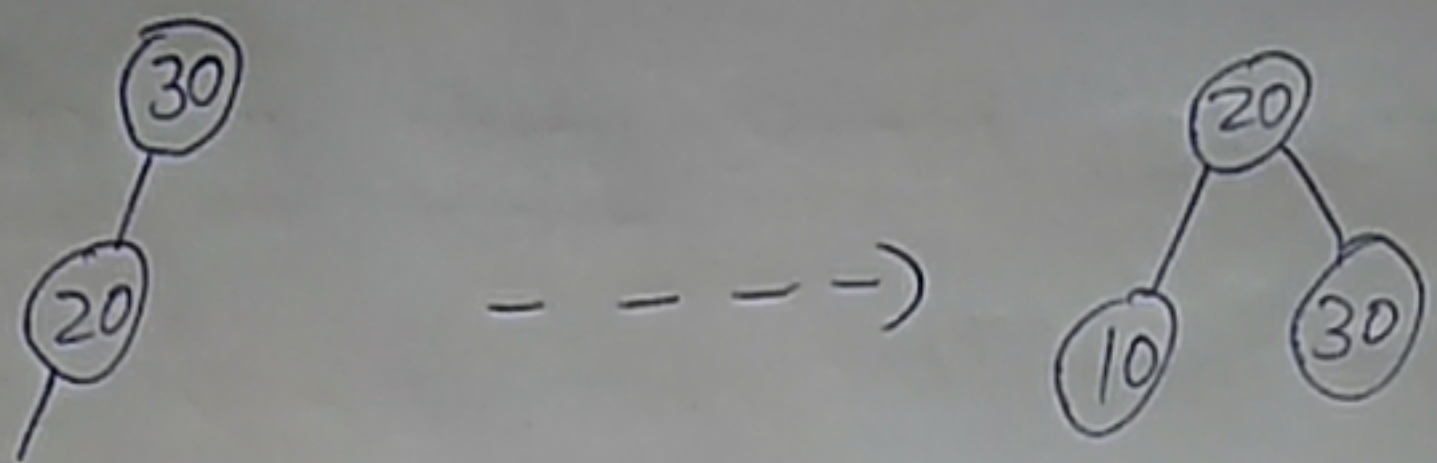
Input :

```
root = None
root = insert(root,7)
insert(root,10)
insert(root,5)
insert(root,3)
insert(root,6)
insert(root,8)
insert(root,12)
print(floor(root,9))
```

Output:

8

Convert Bst to Balanced BST



* A simple solution is to traverse nodes in inorder and one by one into a self-balancing BST like AVL tree. Time Complexity $O(n \log n)$

* An efficient solution can construct balanced BST in $O(n)$.

- Traverse given BST in inorder and store result in an array. This step takes $O(n)$ time. Note that this array would be sorted as inorder traversal of BST always produce sorted sequance.

- Build a balanced BST from the above created sorted array using reavrsive approch.

Implementation:

```
import sys
import math

class Node:

    def __init__(self, data):

        self.data = data

        self.left = None

        self.right = None

def storeBSEnodes(root, nodes):

    if not root:

        return

    storeBstnode(root.left, nodes)

    nodes.append(root)

    storeBstnode(root.right, nodes)

def preorder(root)

    if not root:

        return

    print("{}".format(root.data), end = " ")

    preorder(root.left)

    preorder(root.right)
```

```
def buildtreeutill(nodes, start, end)
    if start > end:
        return None
    mid = (start + end)//2
    node = Nodes[mid]
    node.left = buildtreeutill(nodes, start, mid-1)
    node.right = buildtreeutill(nodes, mid+1, end)
    return node

def buildtree(root):
    if not root:
        return
    nodes = []
    storebstnodes(root, nodes)
    n= len(nodes)
    return buildtreeutill(nodes, 0, n-1)
```

Input:

```
root = Node(10)
root.left = Node(8)
root.left.left = Node(7)
root.left.left.left = Node(6)
root.left.left.left.left = Node(5)
root = buildtree(root)
preorder(root)
```

Output:

7 5 6 8 10