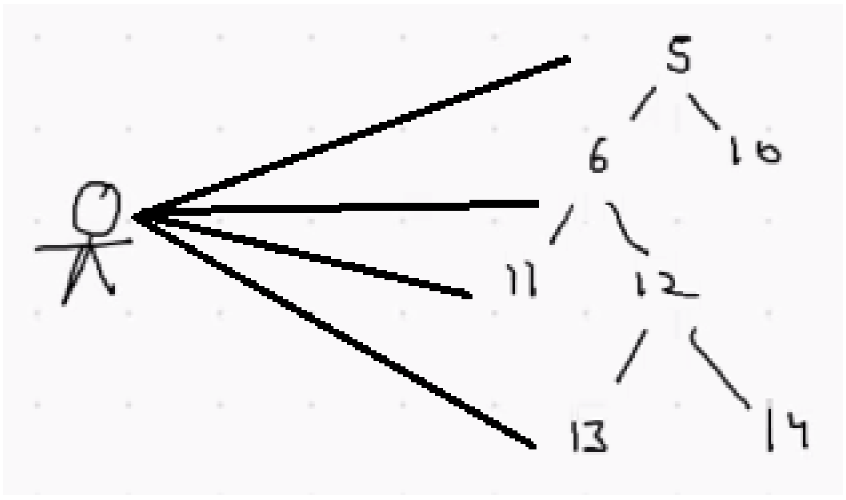
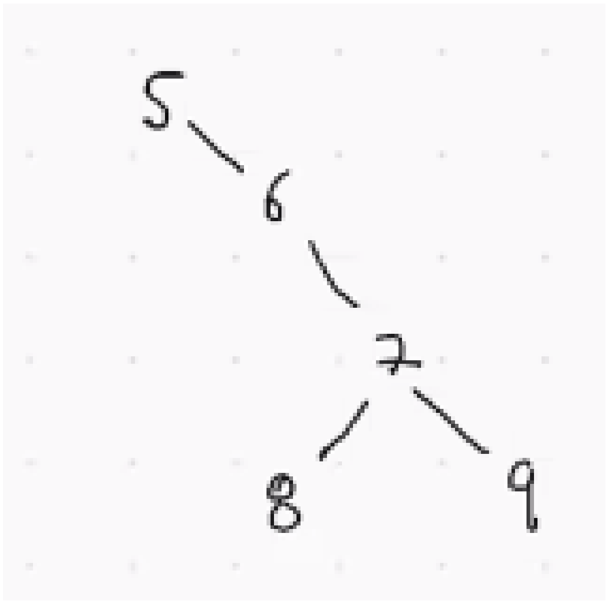
Trees 04

Q) Given a tree, print the left view of the tree.



You can see 5 6 11 13 from the left view

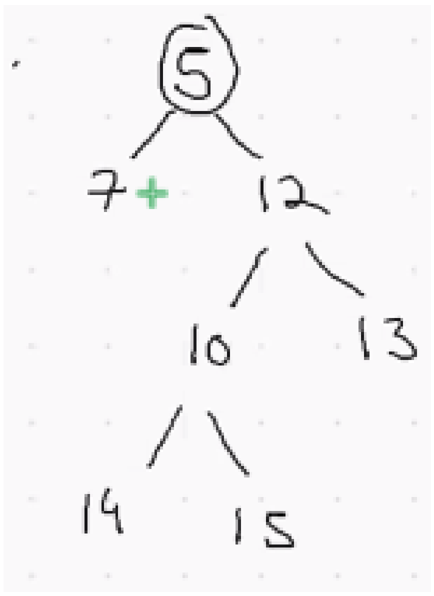


in this we can see 5 6 7 8.

We cannot see 9 as 8 would be in the path.

We can use level order traversal. You have to print the first node of each level, which gives us the left elements. The time complexity would be O(n) and space complexity would be O(n).

How to do this, without using level order traversal.



At 5, the height is zero. When we move towards 7, the height is getting increased. At 12 the height is already increased. Moving forward to left, at 10 the height is further increased and at 13, it will not increase. The same thing happens at 14.

The height is getting changed for first time when we are the left node.

At 5, the max height is 0. When I go to left, the max height is updated to 1. At 7, max height is 1. Since there is no element on the left, we will go to the right. At 8, max height would be increased to 2 as 2 > 1. As there are not elements further, we will return to 5.

From 5, moving towards right, the max height will still be 2. Moving towards right, at 12, the height level is 1 and since it is less than 2, the max height will remain same.

Moving towards 10, the height level is 2 and since it is equal to 2, the max height will remain the same.

From 10, going to left, at 14, the height level will be 3. Since 3 > 2, the max height will be updated to 3. Returning to 10, now the program will check for element on the right i.e., 15. The height at 15 is 3 which is equal to max height and hence there is no need to increase the max height value.

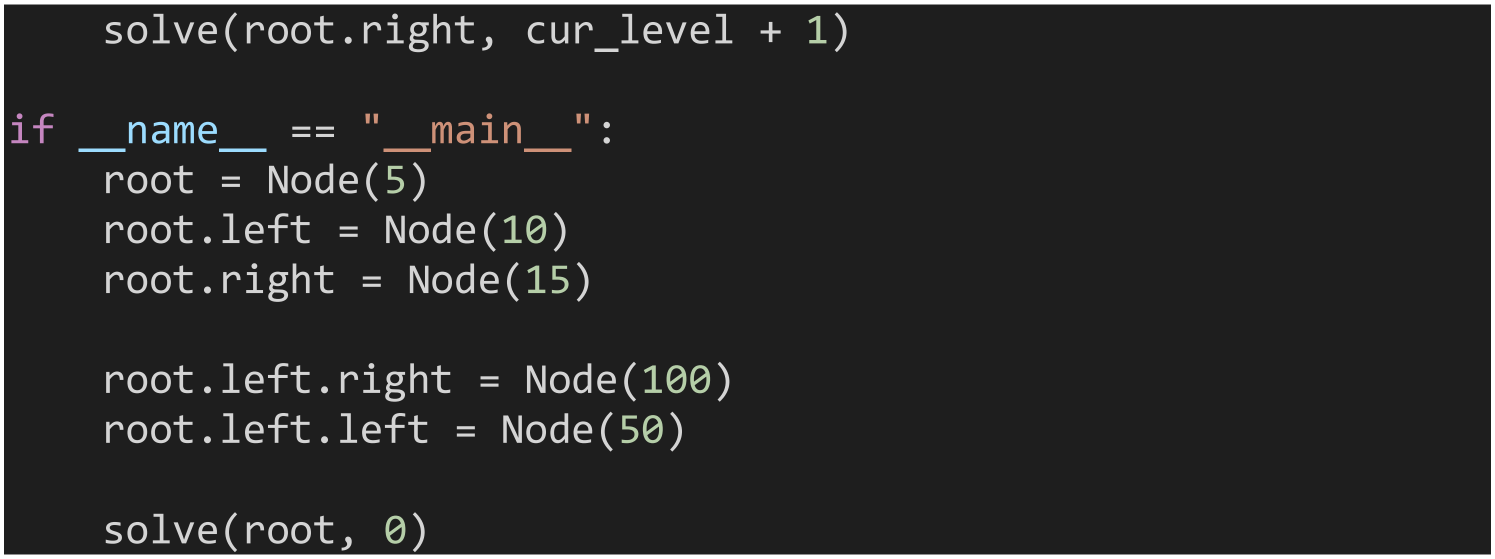
Going back to 10 as there are no elements after 15, the program will return to 12 and check for elements on the right side.

At 13, the height is 2 and since 2 < 3, the max height is not increased. Since 13 is a leaf node, the program will return to 12 and then to 5.

So, the max height will be 3 and every time max height got incremented, the node at which max height got incremented will be printed.

CODE:

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


max_level = -1
def solve(root, cur_level):
    global max_level
    if root is None:
        return

    if cur_level > max_level:
        print(root.data)
        max_level = cur_level

    solve(root.left, cur_level + 1)
```

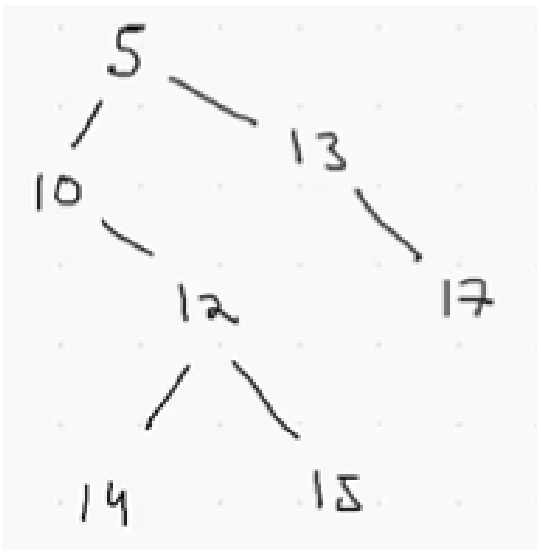


```
    solve(root.right, cur_level + 1)

if __name__ == "__main__":
    root = Node(5)
    root.left = Node(10)
    root.right = Node(15)

    root.left.right = Node(100)
    root.left.left = Node(50)

    solve(root, 0)
```

OUTPUT

```
5
10
50
```

Explanation:



For this, 5, 10, 12 and 13 would be visible.

Taking max level = -1, Using a stack, we'll call solve (5, 0).

Cur level would be 0 and root is 5. Since root is not None, and the cur_level (0) is greater than max_level (-1), we will print root.data (5).

Now, the recursive call root.left, cur_level + 1, solve(10, 1). In this root = 10 and cur_level = 1.

Since cur_level (0) is greater than max_level (1), so we will print 10.

Now, solve (none,2). Since root is none, we will return to solve (10,1).

Now the right of 10 is 12. For solve (12,2) as root is not None and cur_level (1) > max_level (2), the root = 12 is printed.

The program will be paused for solve (12,2) and solve (14,3) will be initiated. Since root is not None and cur_level (2) > max_level (3) so the root value will be printed.

Since 14 is a leaf node and there are not elements on left or right the program will return to solve (12,2). From solve (12, 2), the program will move towards right and solve (15,3).

To get right view, we need to interchange the last two lines of the function `def solve ()`

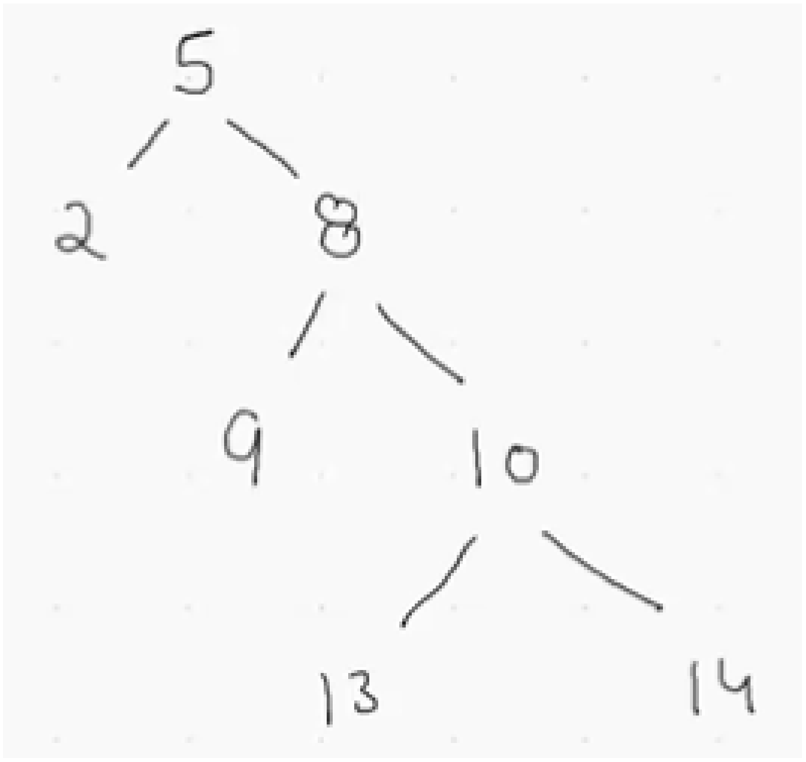
If we put a variable, it will be lost just like cur_level. So, in array if we use array's and set it's value, it will never be lost, this is called referencing in python.

Instead of using a variable, it is good to use array of size one. there won't be need for global variable if we use arrays. In other languages, we can use pointers.

CODE:

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


def solve(root, cur_level, max_level):
    if root is None:
        return

    if cur_level > max_level[0]:
        print(root.data)
        max_level[0] = cur_level

    solve(root.left, cur_level + 1, max_level)
    solve(root.right, cur_level + 1, max_level)

if __name__ == "__main__":
    root = Node(5)
    root.left = Node(10)
    root.right = Node(15)
    root.left.right = Node(100)
    root.left.left = Node(50)

    max_level = [-1]
    solve(root, 0, max_level)
```

Q) Given a tree, print the top view of the tree



In this we have to print [2, 5, 8, 10, 14]

If we consider this tree element on x-axis, i.e., 5 is at 0 of the x-axis and every time we move towards right, we increment the x value by 1 and when we move towards left, the x value will be decremented by 1.

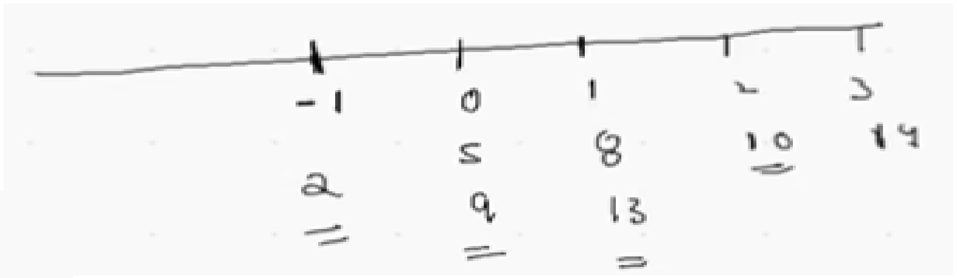
So,

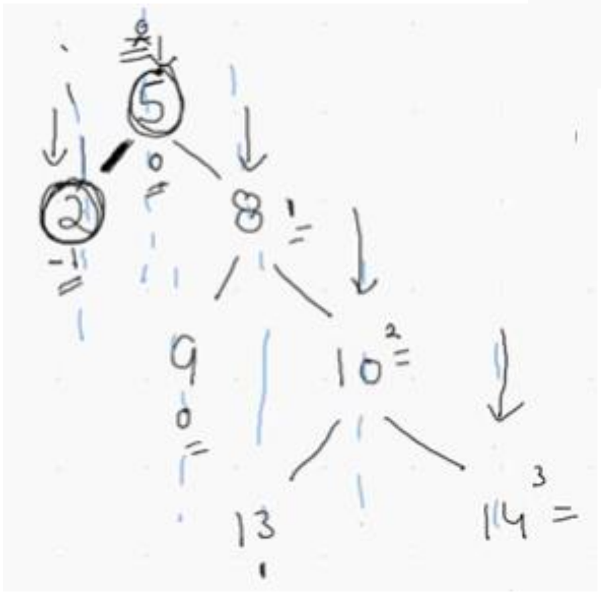
5 & 9 are at 0,
8 & 13 are at 1,
10 is at 2
14 is at 3

We can take elements on left as

2 is at -1.

In all these axis, we have to print the first element.





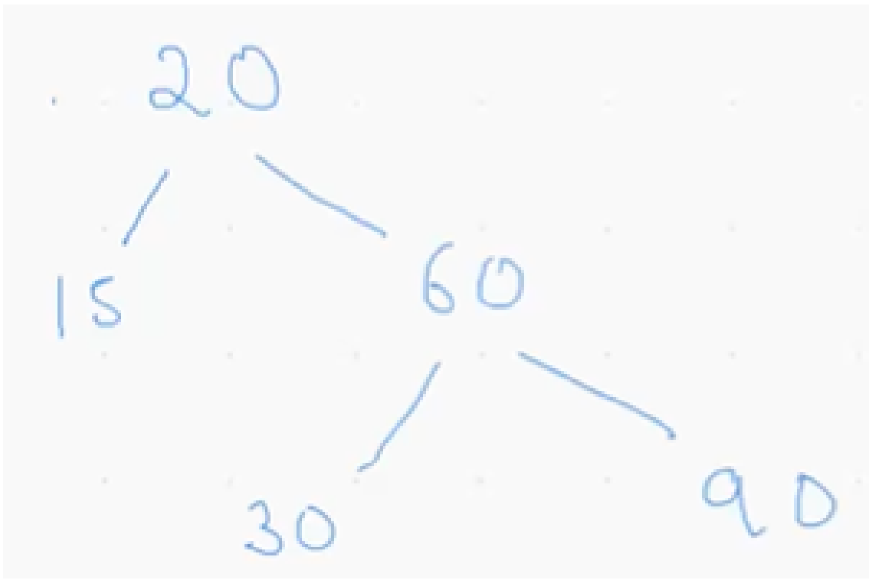
2, 5, 8, 10, 14.

If we consider everything to be vertical axis, with 2 on one axis, then 5, 9 on the next axis, 8 & 13 on the next axis and with 10 and 14 on two new vertical axis.

Using a dict `()`, the key will be access and the value will be list of axis. We have to print the first element in the axis.

We are using vertical order traversal.

If we go to left, the vertical axis is decremented by 1 and when it is going towards right, the vertical axis is incremented by 1.



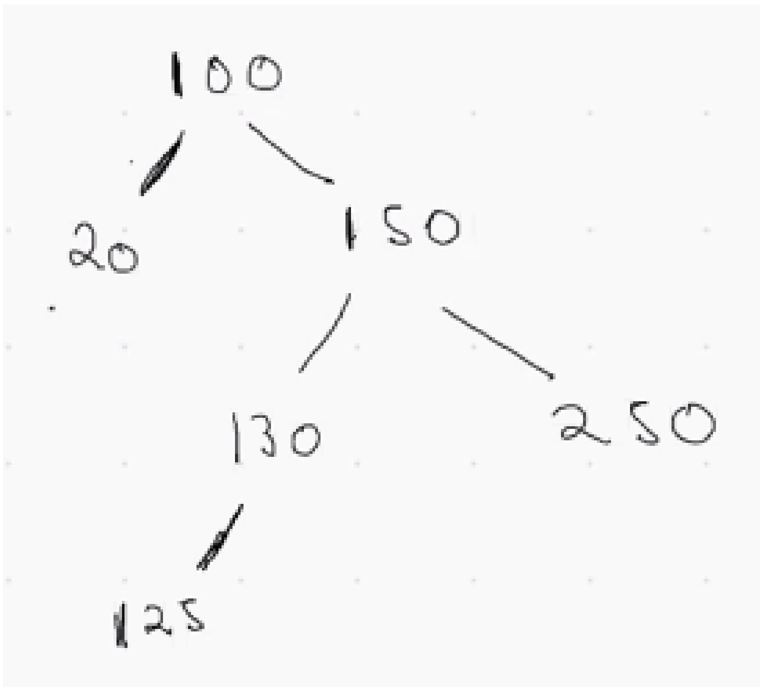
20 will be 0 axis. And we go to left, the axis will be decremented by 1 and to right, the axis will be incremented by 1. So, at 15 the axis will be -1. At 60 the axis will be increased by 1 and at 30, the axis value will be 0 and at 90 the axis value will be 2.

In dict (),

-1: [15]
0: [20, 30]
1: [60]
2: [90]. We have to print all the first values of the keys.

```
d = dict ()
def traverse (root, axis):
    if root is none:
        return
    d[axis].appened(root.val)
    traverse (root.left, axis -1)
    traverse (root.right, axis +1)

for key in range (-1, 3):
    print(d[key][0])
```

Q) Give a tree tell if it is a balanced BST or not.



Is this BST balanced?

Right – left should be less than or equal to 1 for a tree to be a balanced BST.

Is this BST or not?

A tree is said to be BST if the in-order traverse gives us a list of sorted elements.

But how to know if the tree is BST or not without using in-order traversal.

At 125, the max value or the min value will be 125.

The max value at 130 > max value in the left part = 125
The min value at 130 < min value in the right part = 130

The max value at 150 > max value in the left part = 150
The min value at 150 < min value in the right part = 250

The program will return to 100.

CODE:

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


def checkIfValidBST(root):
    if root is None:
        return {
            "min_val": None,
            "max_val": None,
            "isBST": True
        }

    if root.left is None and root.right is None:
        return {
            "min_val": root.data,
                "max_val": root.data,
            "isBST": True
        }

    left_ans = checkIfValidBST(root.left)
    right_ans = checkIfValidBST(root.right)

    if left_ans["isBST"] and right_ans["isBST"] and root.data > 
 left_ans["max_val"] and root.data < right_ans["min_val"]:
        return {
            "min_val": min(root.data, left_ans["min_val"]),
            "max_val": max(root.data, right_ans["max_val"]),
            "isBST": True
        }
    else:
        return {
            "min_val": min(root.data, left_ans["min_val"]),
            "max_val": max(root.data, right_ans["max_val"]),
            "isBST": False
        }


if __name__ == "__main__":
    root = Node(5)
    root.left = Node(10)
    root.right = Node(15)

    root.left.right = Node(100)
    root.left.left = Node(50)

    ans = checkIfValidBST(root)
    print(ans["isBST"])
```

LeetCode: 98. Validate Binary Search Tree

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right

class Solution:

    def checkIfValidBST(self, root):
        if root is None:
            return {
                "min_value": None,
                "max_value": None,
                "isBST": True
            }

        if root.left is None and root.right is None:
            return {
                "min_value": root.val,
                "max_value": root.val,
                "isBST": True
            }

        left_ans = self.checkIfValidBST(root.left)
        right_ans = self.checkIfValidBST(root.right)

        if left_ans["isBST"] and right_ans["isBST"] and root.val > left_ans["max_value"] and root.val < right_ans["min_value"]:
            return {
                "min_value": min(root.val, left_ans["min_value"]),
                "max_value": max(root.val, right_ans["max_value"]),
                "isBST": True
            }
        else:
            return {
                "min_value": min(root.val, left_ans["min_value"]),
                "max_value": max(root.val, right_ans["max_value"]),
                "isBST": False
            }
    def isValidBST(self, root: TreeNode) -> bool:
        ans = self.checkIfValidBST(root)
        return ans["isBST"]
```

LeetCode: 230. Kth Smallest Element in a BST:

```
class Solution:
    cnt = 0
    ans = -1

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

        self.solve(root.left, k)
        Solution.cnt += 1
        if Solution.cnt == k:
            Solution.ans = root.val
            return

        self.solve(root.right, k)

    def kthSmallest(self, root: TreeNode, k: int) -> int:
        Solution.cnt = 0
        Solution.ans = -1
        self.solve(root, k)
        return Solution.ans
```