Binary Tree 2

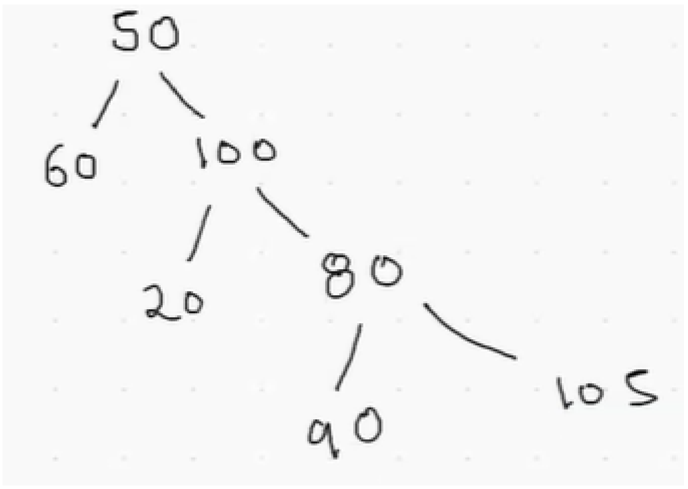
Post Order (left—right—root)

First, the program will go to left = 50. From there the program will check for any element towards left. Since there is no other element, it will check if there is a right. Since there is no right, the program will return back to 50 and will print 50.

50

It will get back to 20 and now will check for right of the root. The program will come to 70 and root is also shifted to 70. At root = 70, the program will look for elements on the left and will now move the root to 80. At 80, moving recursively towards left, the program will come to 20. There is no element either on the left or on the right of the element and hence after checking on left and right, the program will return to root = 20 and will print 20.

Q) Given a binary tree print the leaf nodes



We have to write a function, to print leaf nodes. So leafnodes are, 60, 20, 90, 105. If we want to travel to each node in linked list, we have 3 order. Pre, post and inorder traversals.

If we want to travel from **bottom to up**, then we use **post-order traversal**

If we want to travel from **top to bottom**, then we use **pre-order traversal**.

To write a recursive solution, we need a base condition. So, if our root is none, we can just return. For pre-order traversal, we print the root and then we call leaf nodes root.left and root.right.

```
def printleafnode(root)
    if root is none:
        return
    {           }
    print leafnodes(root.left)
    print leafnodes(root.right)
```

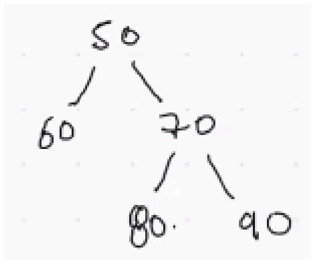
Instead of printing the root, we have to check if the node is leaf node or not. These two things will make us traverse the tree.

```
if (root.left is none and root.right is none):
    print(root.data)
```

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

def printLeafNode(root):
    if root is None:
        return

    if root.left is None and root.right is None:
        print(root.data)

    printLeafNode(root.left)
    printLeafNode(root.right)


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

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

    printLeafNode(root)
```

The time complexity for is O(n) and space complexity would be O(n).

Q) Given a Binary Tree, print the no. of nodes in the tree.



= 5 nodes.

Implementation:

```
def size(root, no):
    if root is none:
        return none
    num += 1
    size(root.left, num)
    size(root.right, num)
size(root, 0)
```