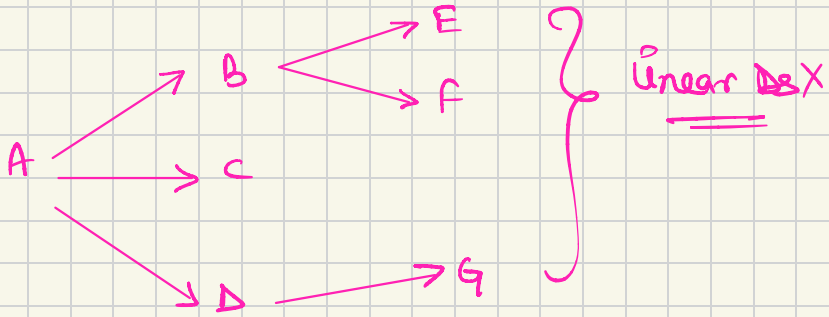
# Binary Trees

↳ Non linear data structure



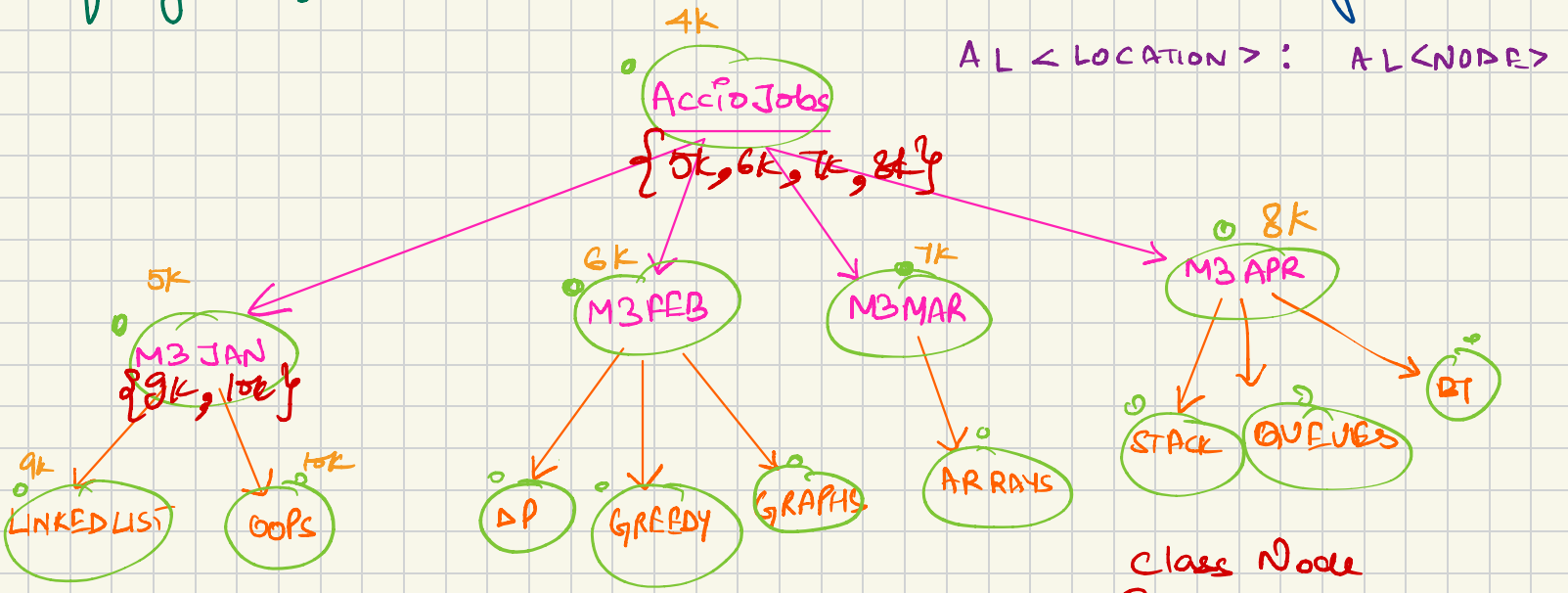

## Data

1. family tree
2. file system

file system.

NAME : String

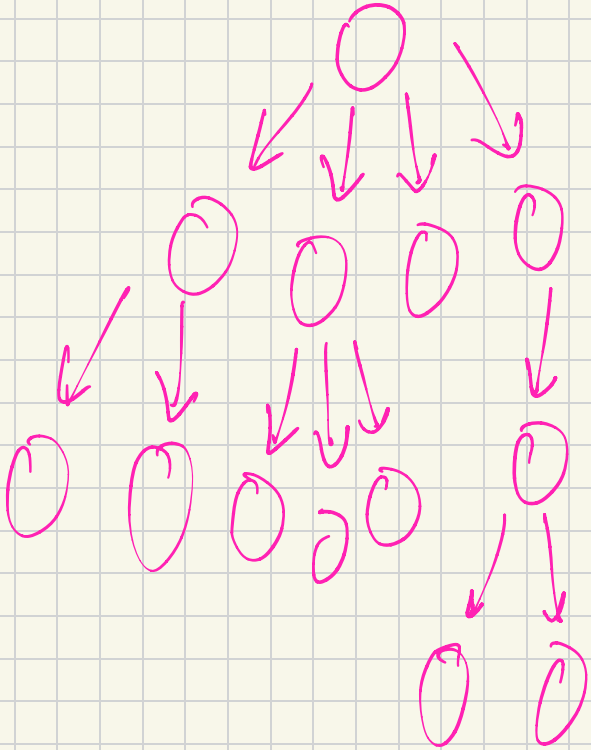
AL<LOCATION> : AL<NODE>




NOTE: We have N number of children

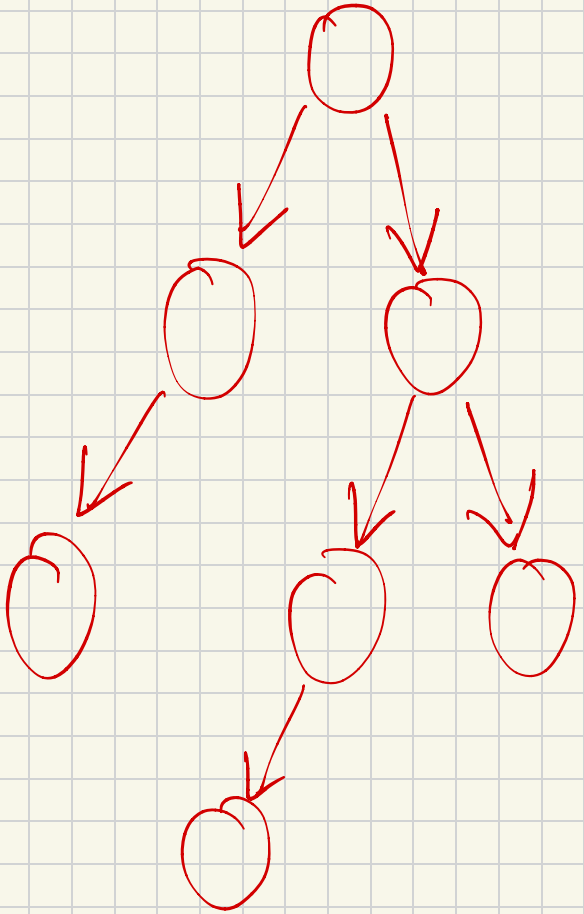
{generic tree}

```
class Node
{
    String data;
    ArrayList<Node> child;
}
```



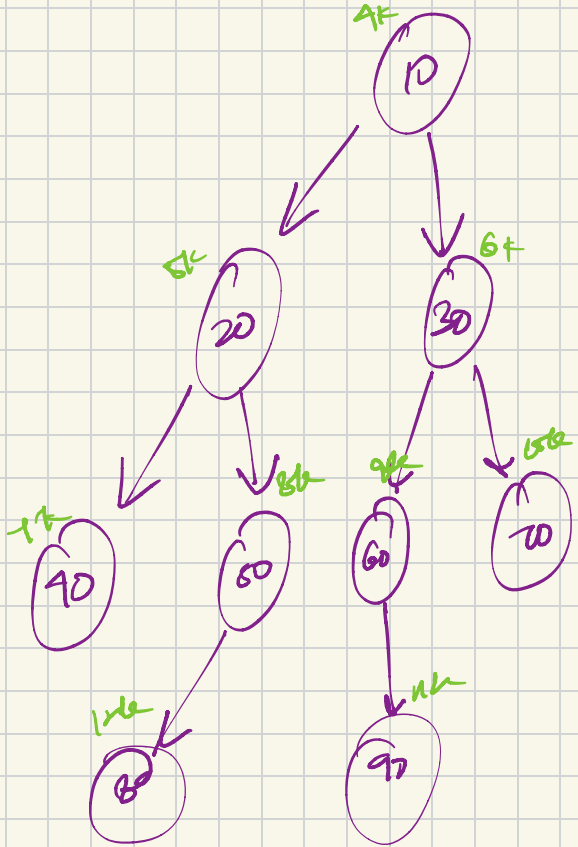
# Binary Tree

(atmost 2 children)



{ Each Node can have, 0, 1 or 2 child Nodes

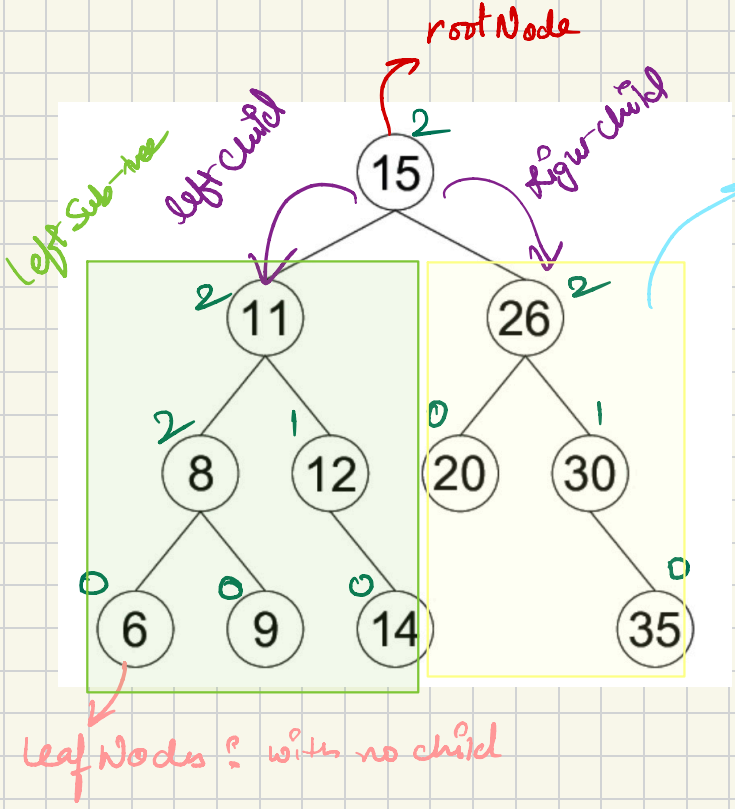
# Binary Trees




```
class Node
{
    int data;
    Node left;
    Node right;
}
```

degree : No. of children

parent : 11
child : 8, 12

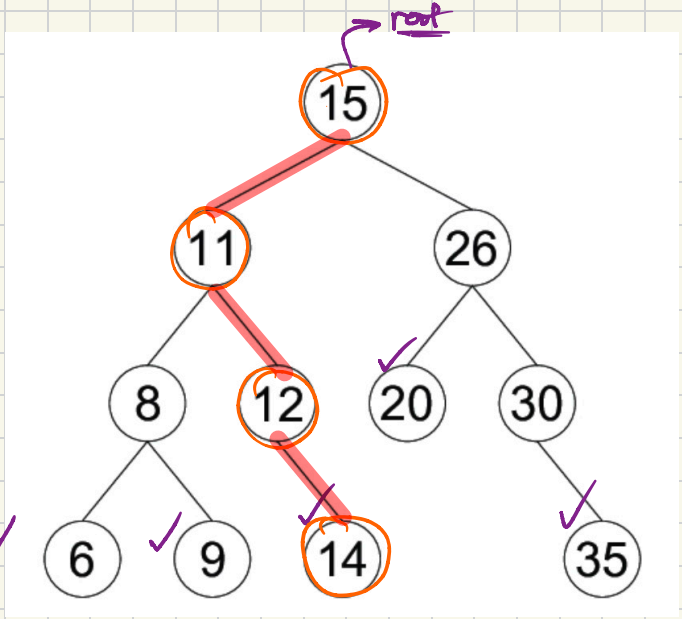



Leaf Nodes : with no child

Siblings:
(11, 26)
(8, 12)
(20, 30)
(6, 9)

# Height of the tree

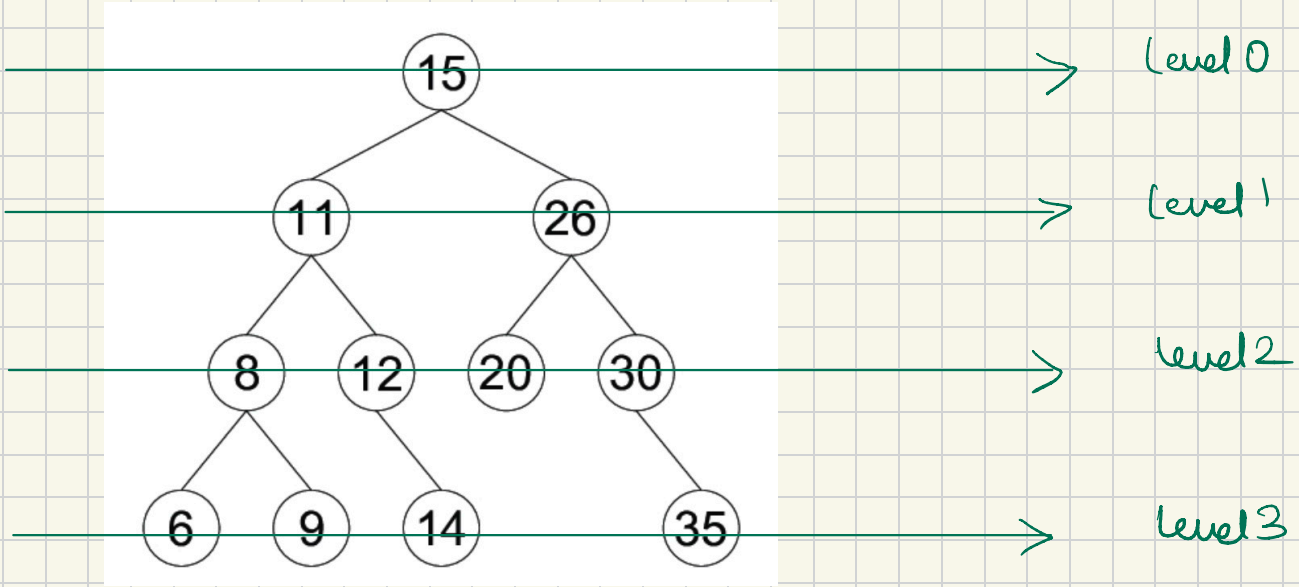
dist b/w root Node and the deepest leaf Node.



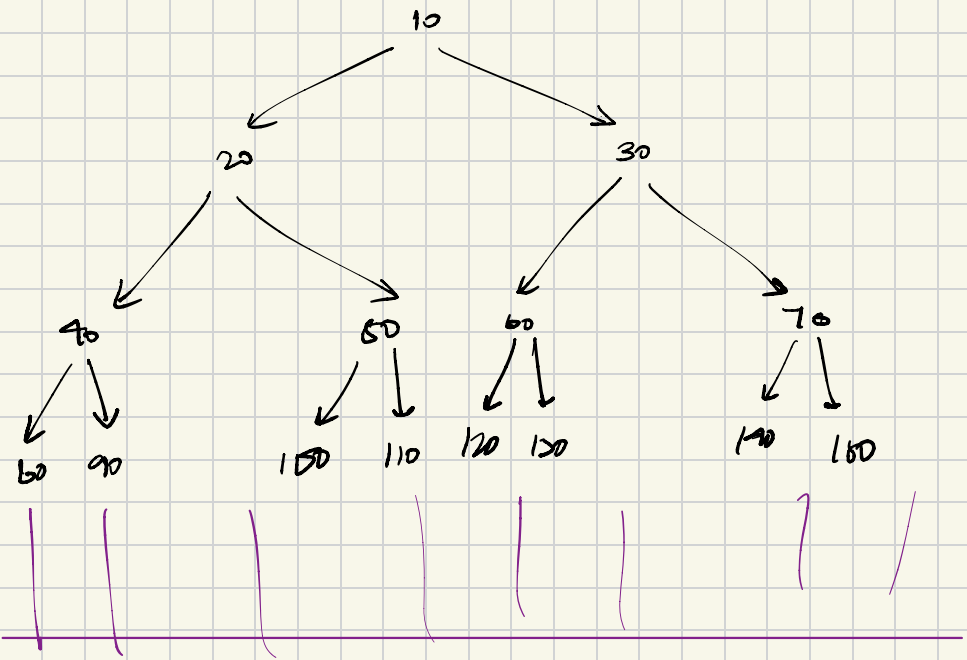

height = 4 {in terms of Nodes}

height = 3 {in terms of edges}

# Level in a Binary Tree



## Perfect Binary Tree { No. of nodes at each level $(l) = 2^l$ }




level 0 = 1 $\rightarrow 2^0$

level 1 = 2 $\rightarrow 2^1$

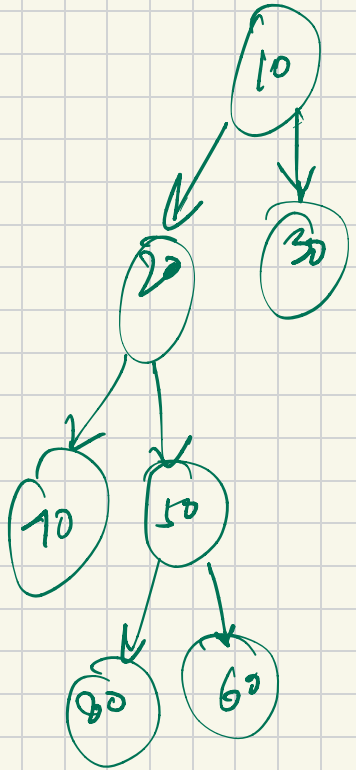
level 2 = 4 $\rightarrow 2^2$

level 3 = 8 $\rightarrow 2^3$

level k = $2^k$

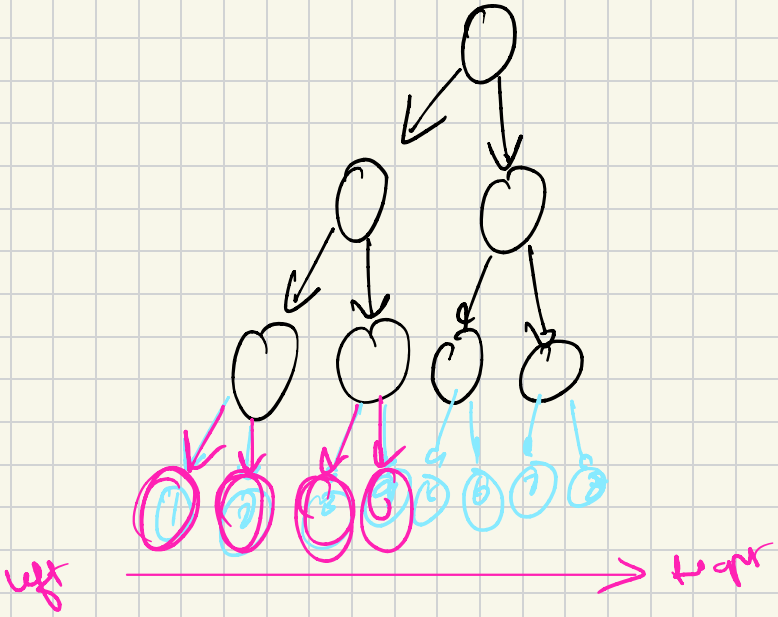
## Full Binary Tree

↳ Each Node either have 0 or 2 children.



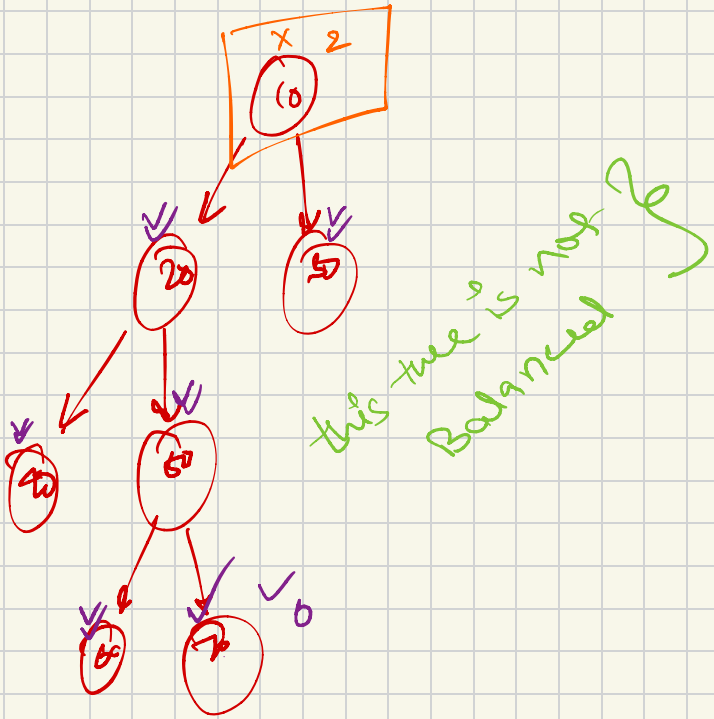
# Complete Binary Tree

- where each level is completely filled except the last level, where nodes are as left positioned as possible.



# Balanced Binary Trees

- A BT where each Node is balanced.




## Balanced Node

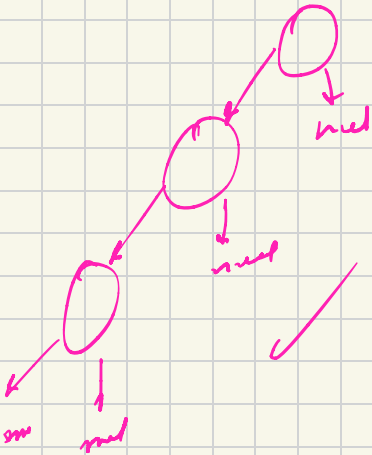
abs. diff $h_{LST}$ and $h_{RST}$ $<= 1$

$$|h(LST) - h(RST)| \leq 1$$

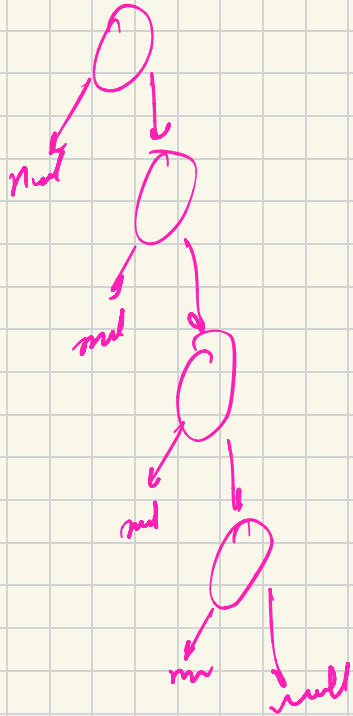
# Skew Tree

## ① Left Skew Tree

left child, no child



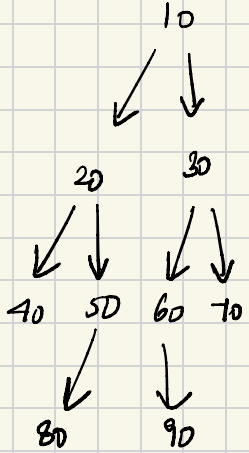

## ② Right Skew tree



# Traversal over a tree

## • Pre Order Traversal

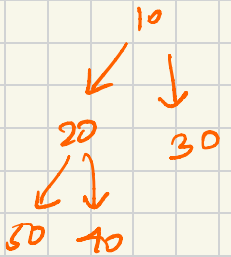
{ 10, 20, 40, 50, 80, 20, 60, 90, 70 }



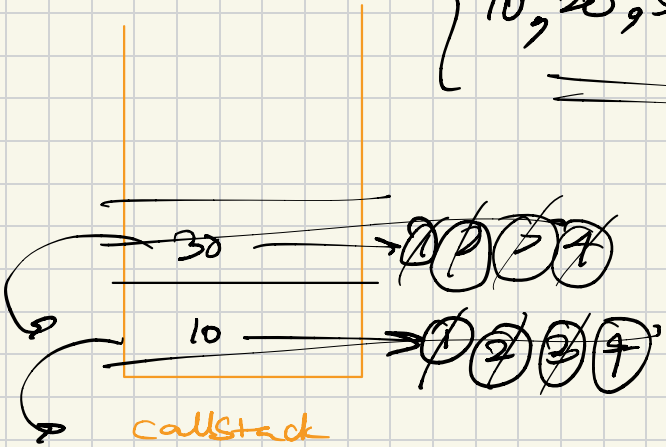

- root
- preorder LST
- preorder RST

faith: print pre-order of the given BT from root  10, 20, 50, 40, 30

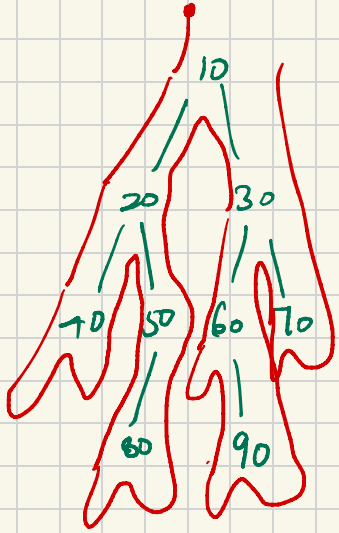
```
void preOrder(Node root)
{
(1) if(root == null) return;
(2) syso(root.data);
(3) preOrder(root.left);
(4) preOrder(root.right);
}
```




{10, 20, 50, 40, 30}

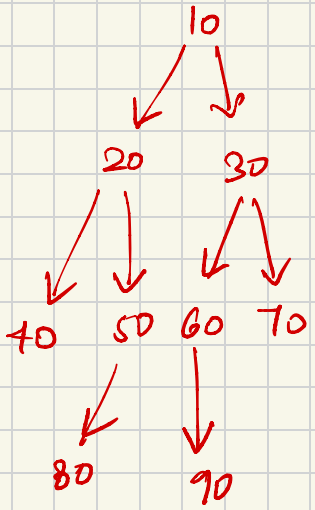


Eular path!



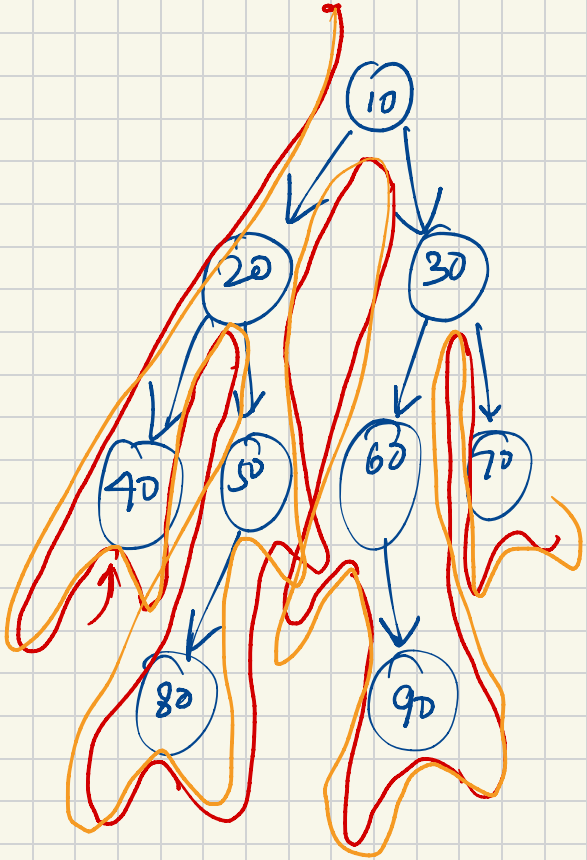

{10, 20, 40, 50, 20, 30, 60, 90, 70}

# Inorder traversal



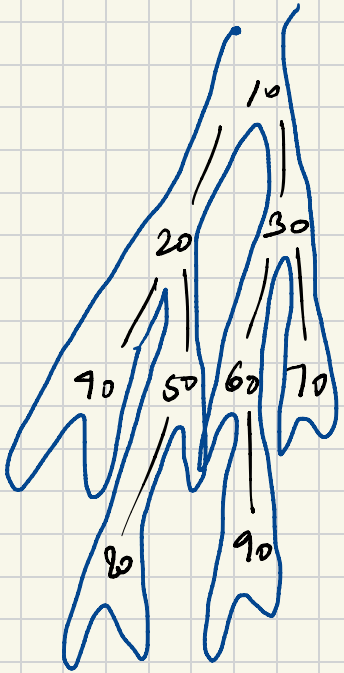

in: 40, 20, 80, 50, 10, 60, 90, 30, 70

- inorder(LST)
- print root's data
- inorder(RST)




40, 20, 80, 50, 10, 60, 90, 30, 70

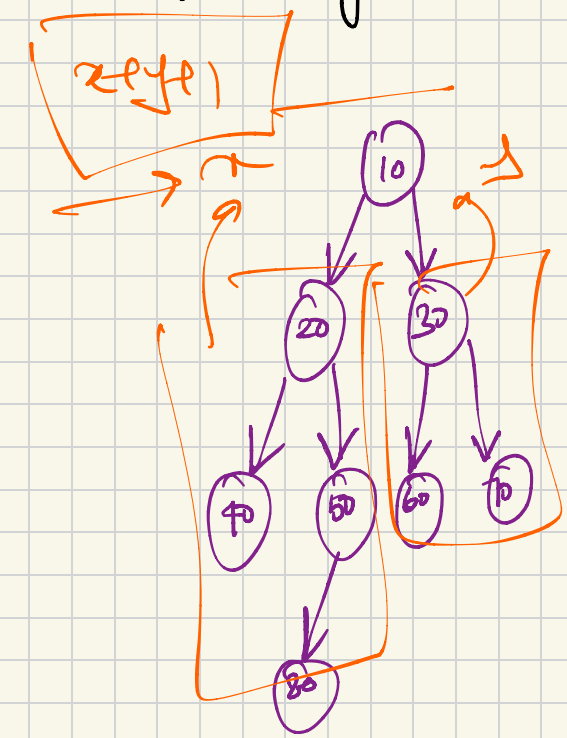
# Post Order Traversal




40, 80, 50, 20, 90, 60, 70, 30, 10

- Post (LST)
- Post (RST)
- root data

# Size of Binary Tree



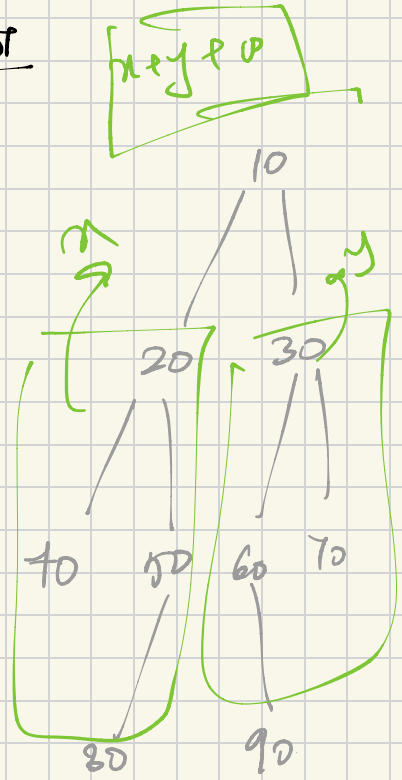

fxn: returns size of tree starting from root

```
int size(Node root)
{
    if(root == null) return 0;

    int a = size(root.left)
    int b = size(root.right)

    return(a+b+1);
}
```

Sum of BT




Sum = 450

```
int Sum(Node root)
```

fallu! return sum of the nodes