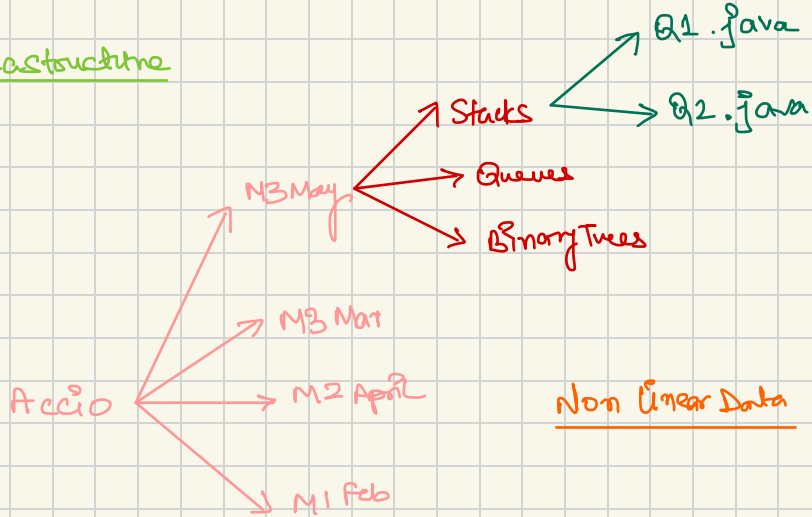
Binary Trees

Non Linear Datastructure

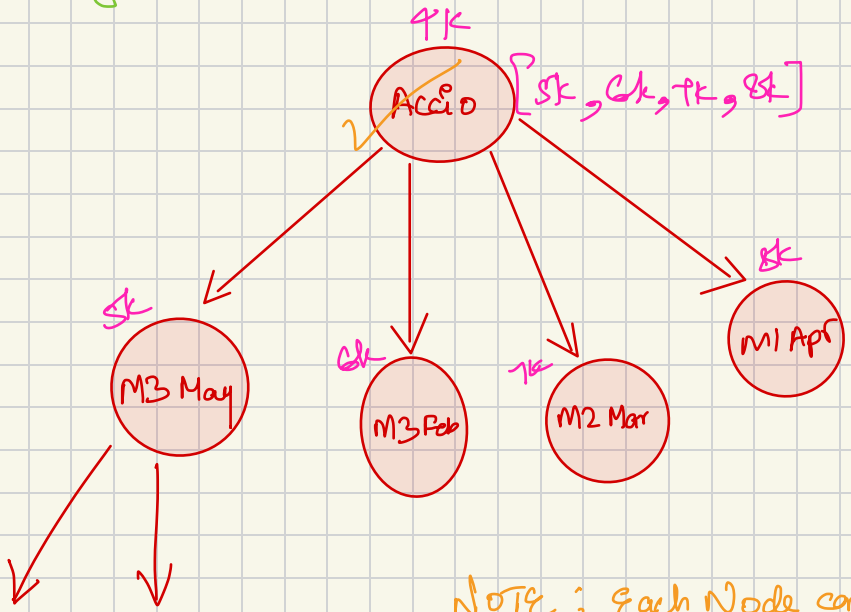
Data

- Family Tree
- File System



Non Linear Data

File System




```
class Node
{
    String Name;
    ArrayList<Node> link;
}
```

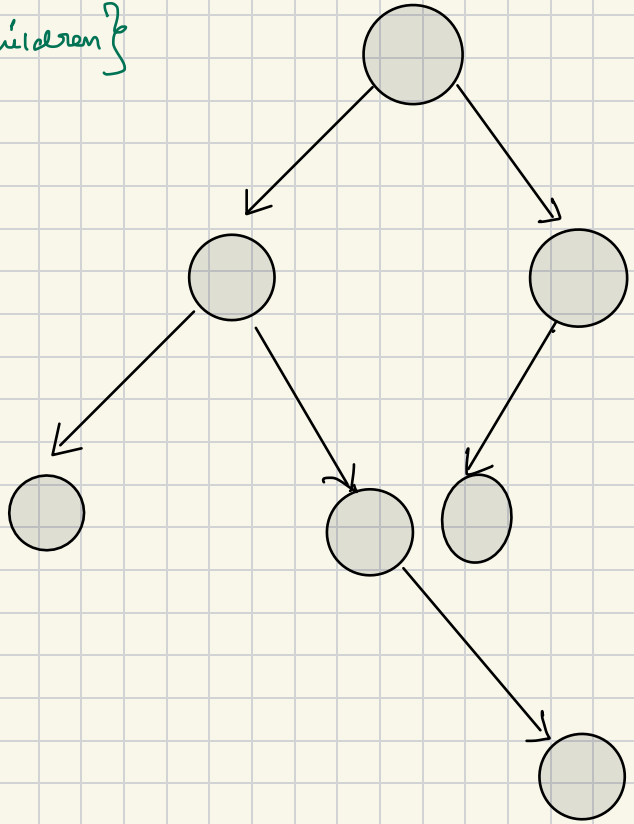
childrens

NOTE : Each Node can have N childrens.

{Generic Tree}

BinaryTree

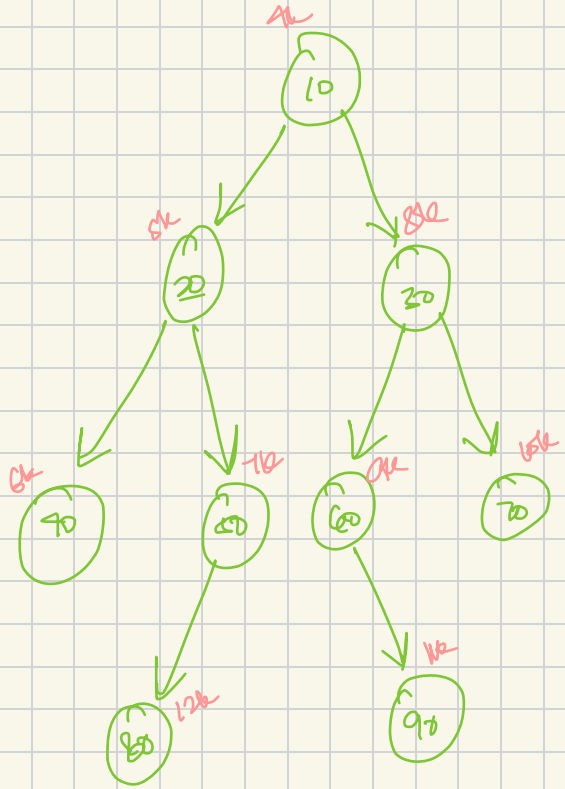
{atmost 2 children}



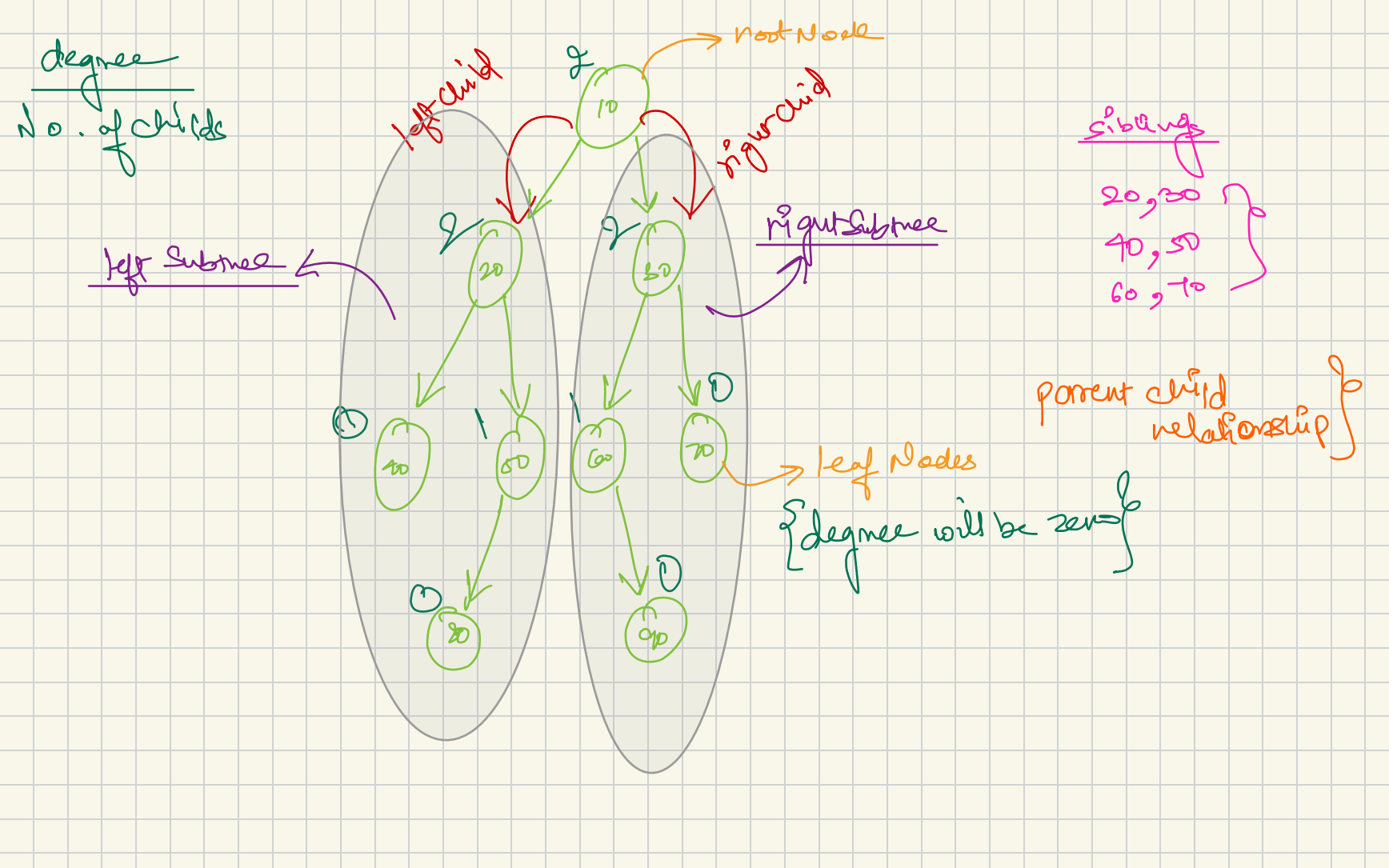
Each Node can atmost 2 children

{0, 1, 2}

Binary Trees

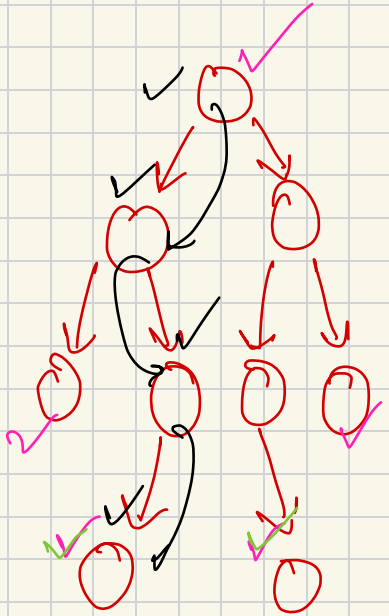



```
class Node
{
    int data;
    Node left;
    Node right;
}
```


degree
No. of childs
left child
root Node
right child
left Subtree
right Subtree
Sibling
20, 30
40, 50
60, 70
parent child relationship
Leaf Nodes
{degree will be zero}
10
20
30
40
50
60
70
80
90

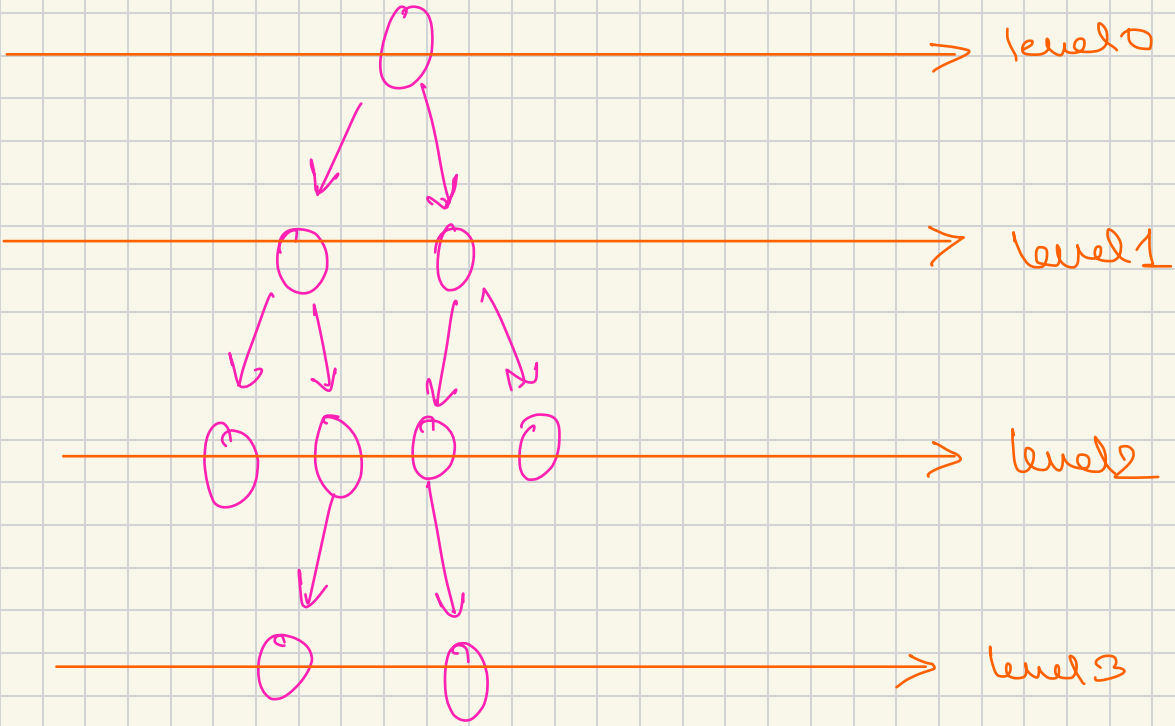
Height of Tree

dist. b/w rootNode and deepest leafNode



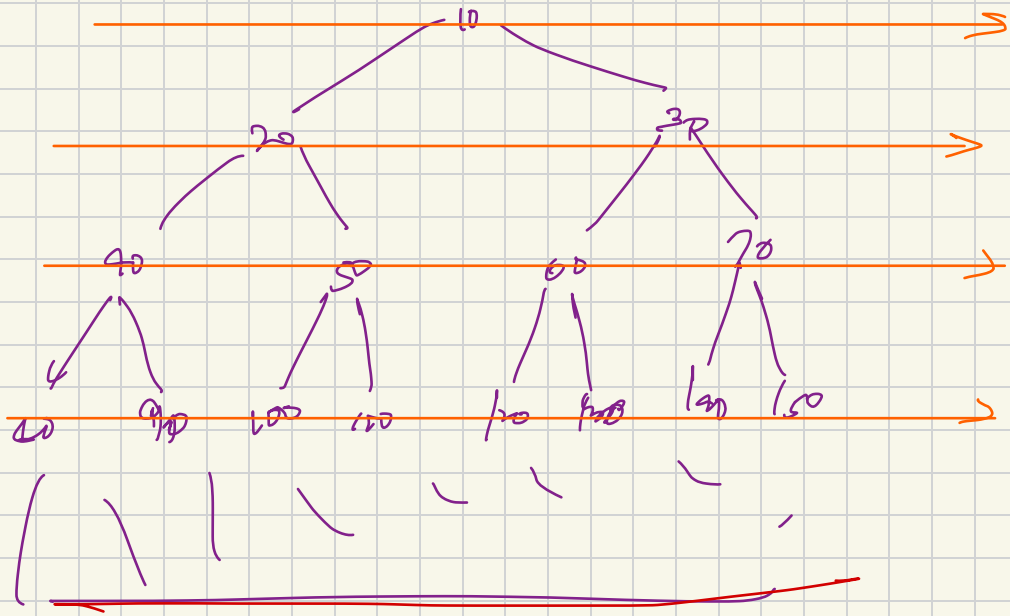
height = 4 { in terms of Nodes }

Level in a Binary Tree



Perfect Binary Tree

{No. of Node in each (l) = 2^l}



level 0 = 1 = 2^0

level 1 = 2 = 2^1

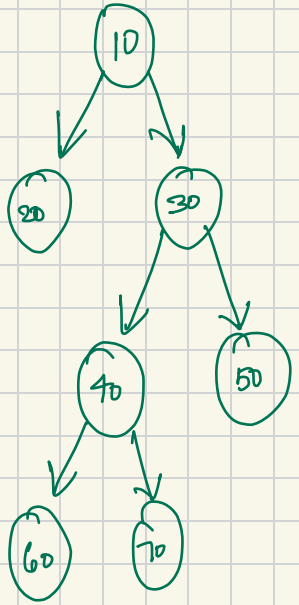
level 2 = 4 = 2^2

level 3 = 8 = 2^3

level k = 2^k

Full Binary Tree.

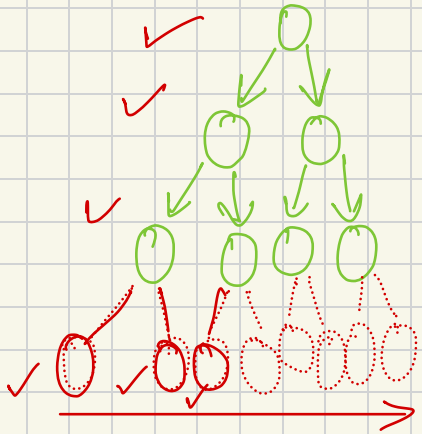
→ Each Node has either 0 or 2 children



Full Binary Tree

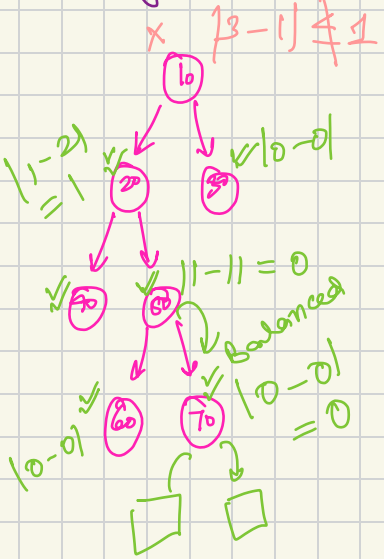
Complete Binary Tree.

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



Balanced Binary Tree

- a binary Tree where each Node is Balanced.



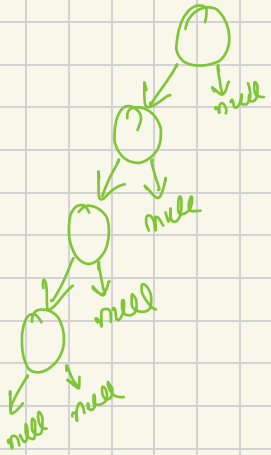
Balanced Node

$$|h_{LST} - h_{RST}| \leq 1$$

Skew Tree

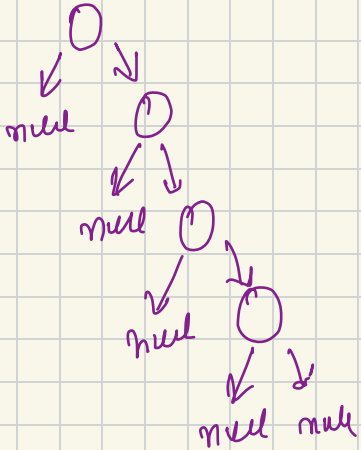
① Left Skew Tree

(left child / No child)



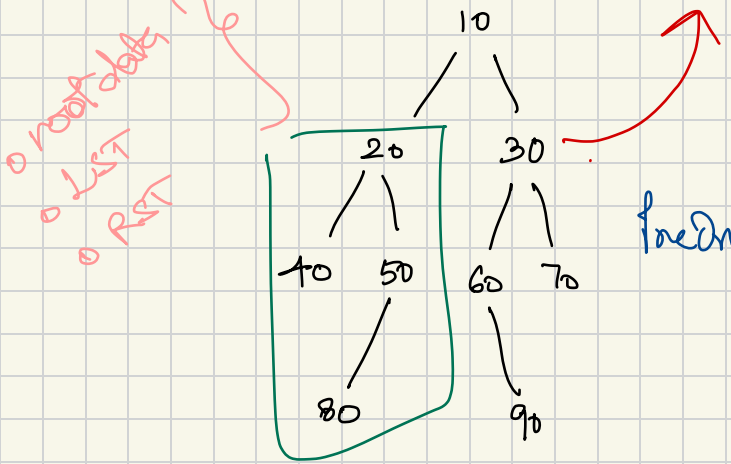
② Right Skew Tree

(right child / No child)

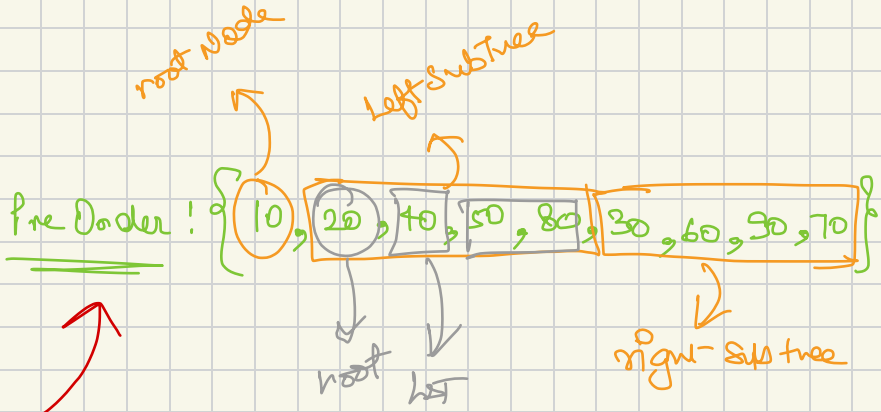


Traversal over a Tree

- Pre Order Traversal



- root data
- LST
- RST



PreOrder = root.data + PreOrder(LST)
+ PreOrder(RST)

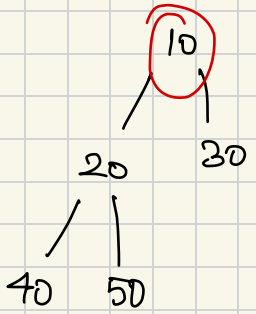
Recursion

faith: print preorder of a BT starting from given root.

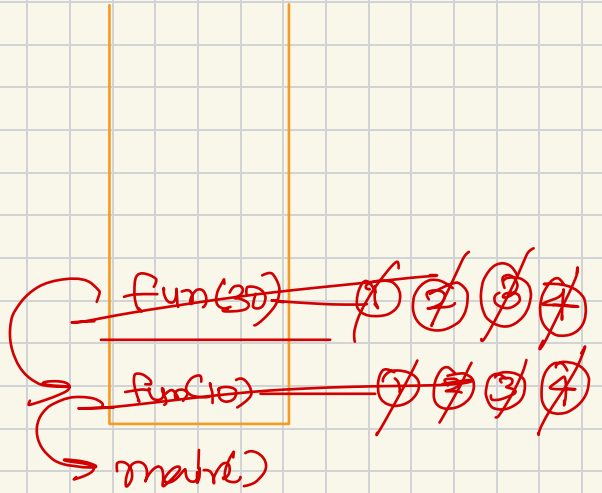
```
void printPreOrder(Node root)
{
    if (root == null) return;

    print(root.data);
    printPreOrder(root.left);
    printPreOrder(root.right);

}
```

faith: print preorder of a BT starting from given root.

```
void printPreOrder(Node root)
{
(1)   if (root == null) return;

(2)   print(root.data);
(3)   printPreOrder(root.left);
(4)   printPreOrder(root.right);
}
```

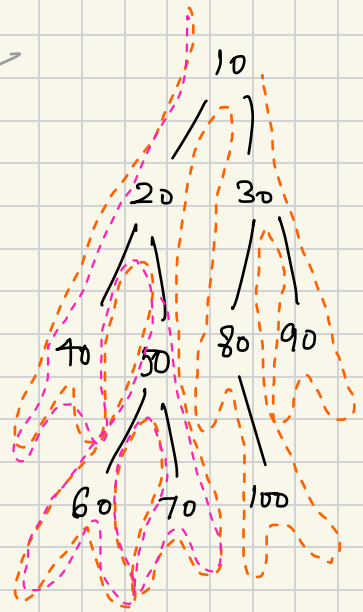


Call stack

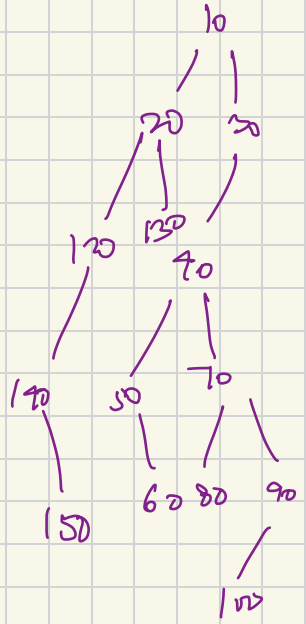


10 20 40 50 30

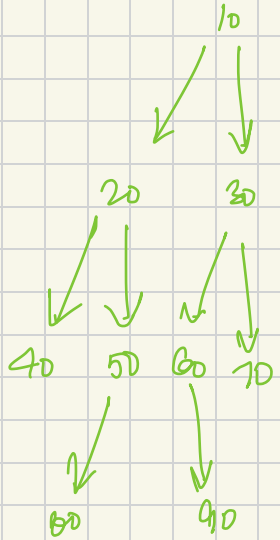
Euler path



{10, 20, 40, 50, 60, 70, 80, 90, 100, 90}



Inorder Traversal



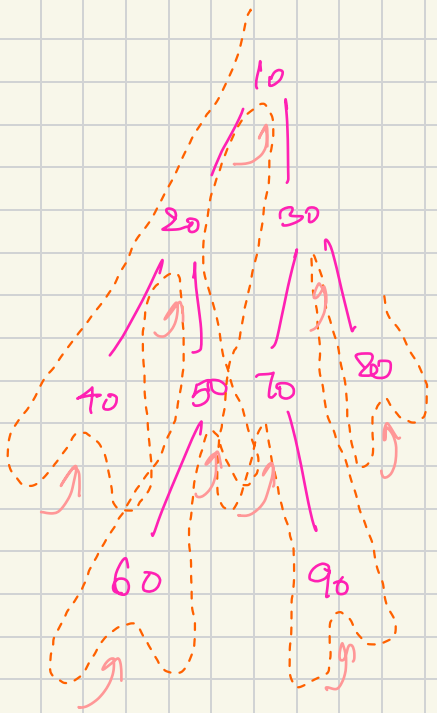
Inorder : 40, 20, 80, 50, 10, 60, 90, 30, 70

inorder LST

inorder RST

Faith; print inorder of BT starting from root

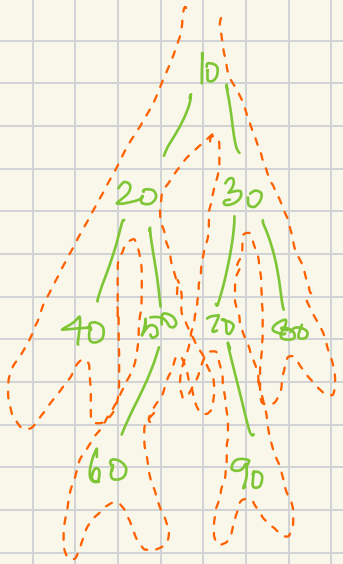
```
void fun(Node Root)
{
    if(root == null) return;

    fun(Root.left)
    print(Root.data)
    fun(Root.right)
}
```



40, 20, 60

NOTE: print on 2nd visit, wherever you see a dip

Post Order Traversal



Post order

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

- Post (left)
- Post (Right)
- Post (root)

Size of a Binary Tree

print no. of Nodes in a BT

faith: returns size of the BT.

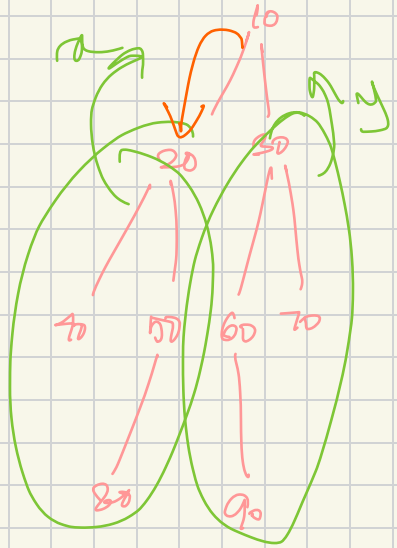
```
int size(Node root)
{
    if(root == null) return 0;

    int a = size(root.left)
    int b = size(root.right)
    return a + 1 + b;
}
```

Base Case

Sum of BT

Sum = 450

x + root.data + y



faith: returns sum of BT

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

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

    return a + root.data + b;
}
```

Diameter of Tree

- Max^m dist. b/w any two leaf Nodes

{40, 60} → 4

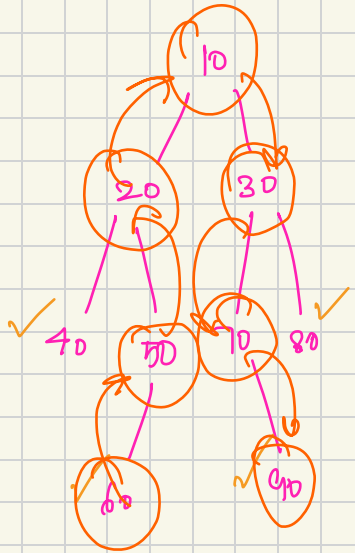
{40, 90} → 6

{40, 80} → 5

{60, 90} → 7

{60, 80} → 6

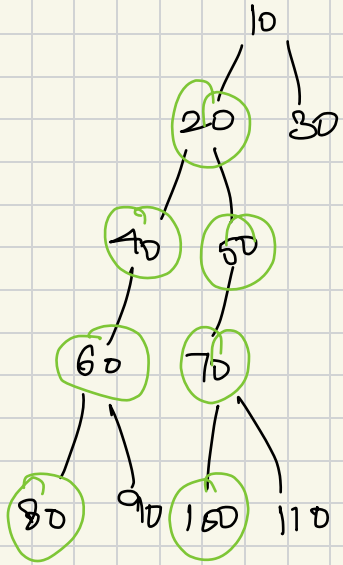
{90, 80} → 4



diameter = 7

Diameter = LST_h + RST_h + 1

WRONG

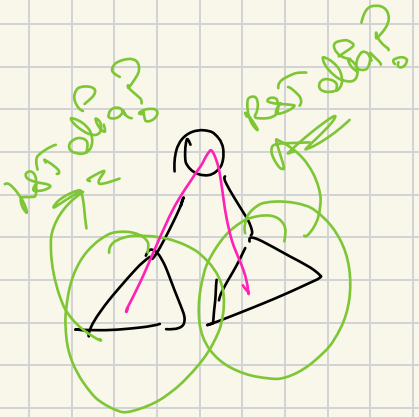


4+4+1 = 6

NOTE : diameter doesn't pass through root always

tenth: returns diameter

TC: $O(N^2)$ | SC: $O(H)$ → height of tree




```
int diameter(Node root)
{
    if(root == null) return 0;
    int x = diameter(root.left)
    int y = diameter(root.right)
    int lHt = height(root.left)      O(N)
    int rHt = ____
    int z = lHt + 1 + rHt;
    return max{x, y, z}
}
```

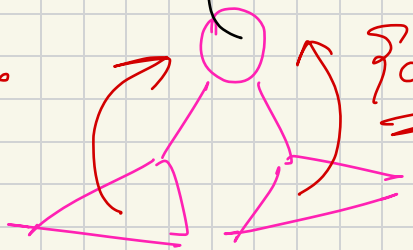
TC: $O(N \times N) = O(N^2)$

{ TCP/IP } man { Left dia, Right dia, Left height }

man { Left height, Right height }

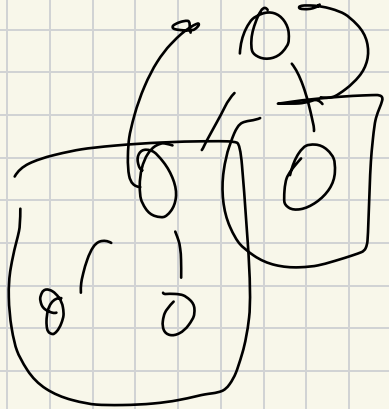
class hair
{
int dia;
int height;
}

{ dia, h }



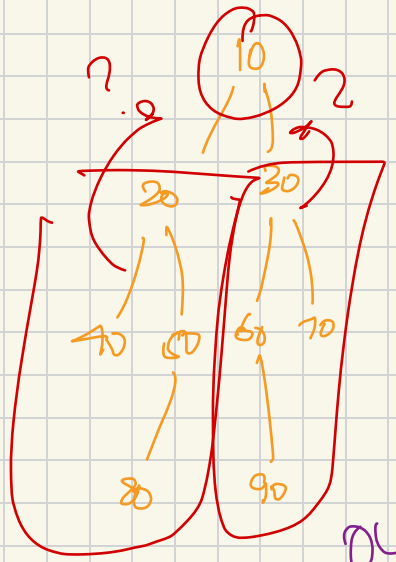
{ dia, h }

{ dia, height }



Balanced Tree

fun: returns if tree is balanced or not.

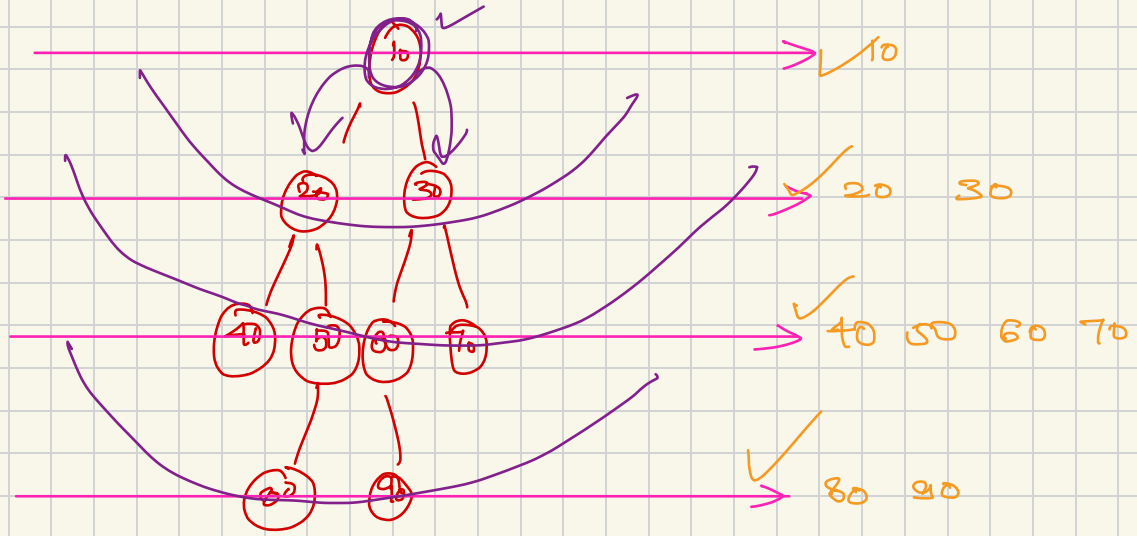


TC: O(N²)

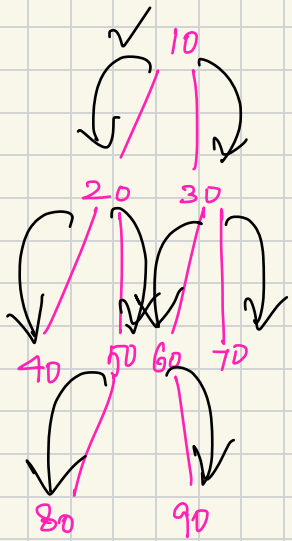
SC: O(H)

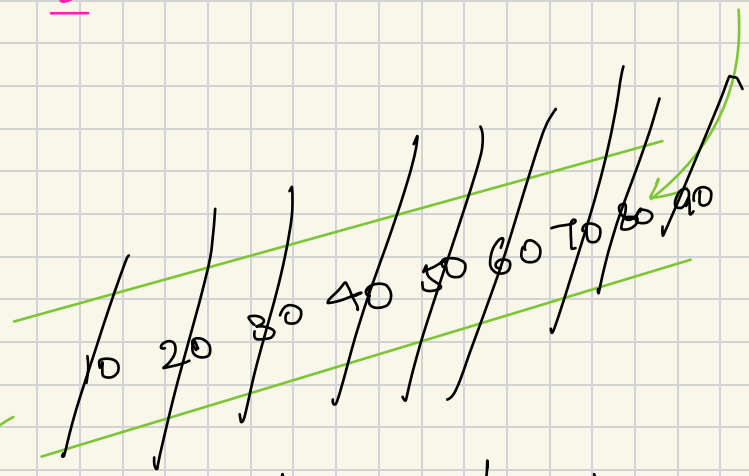
```
boolean isBalanced(Node Root)
{
  if(root==null
    return true;

  boolean f1 = isBalanced(Root.left);
  boolean f2 = isBalanced(Root.right);
  int hLeft = height(Root.left);
  int hRight = height(Root.right);
  boolean f3 = false
  if(Math.abs(  ) <= 1)
  {
    f3=true;
  }
}
```

Level Order Traversal {BFS}



BFS



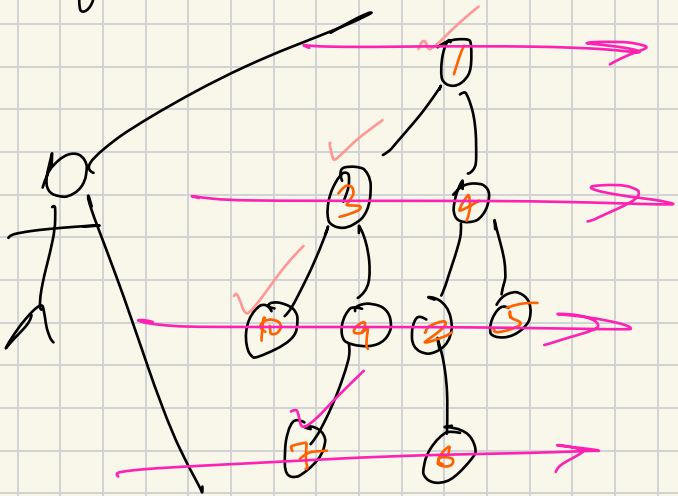


level = 0 1 2 3 4 size = 1 2 4 8 16

level order traversal

10
20 30
40 50 60 70
80 90

Left View



{1, 3, 10, 7}

{ first Node of Each Level }