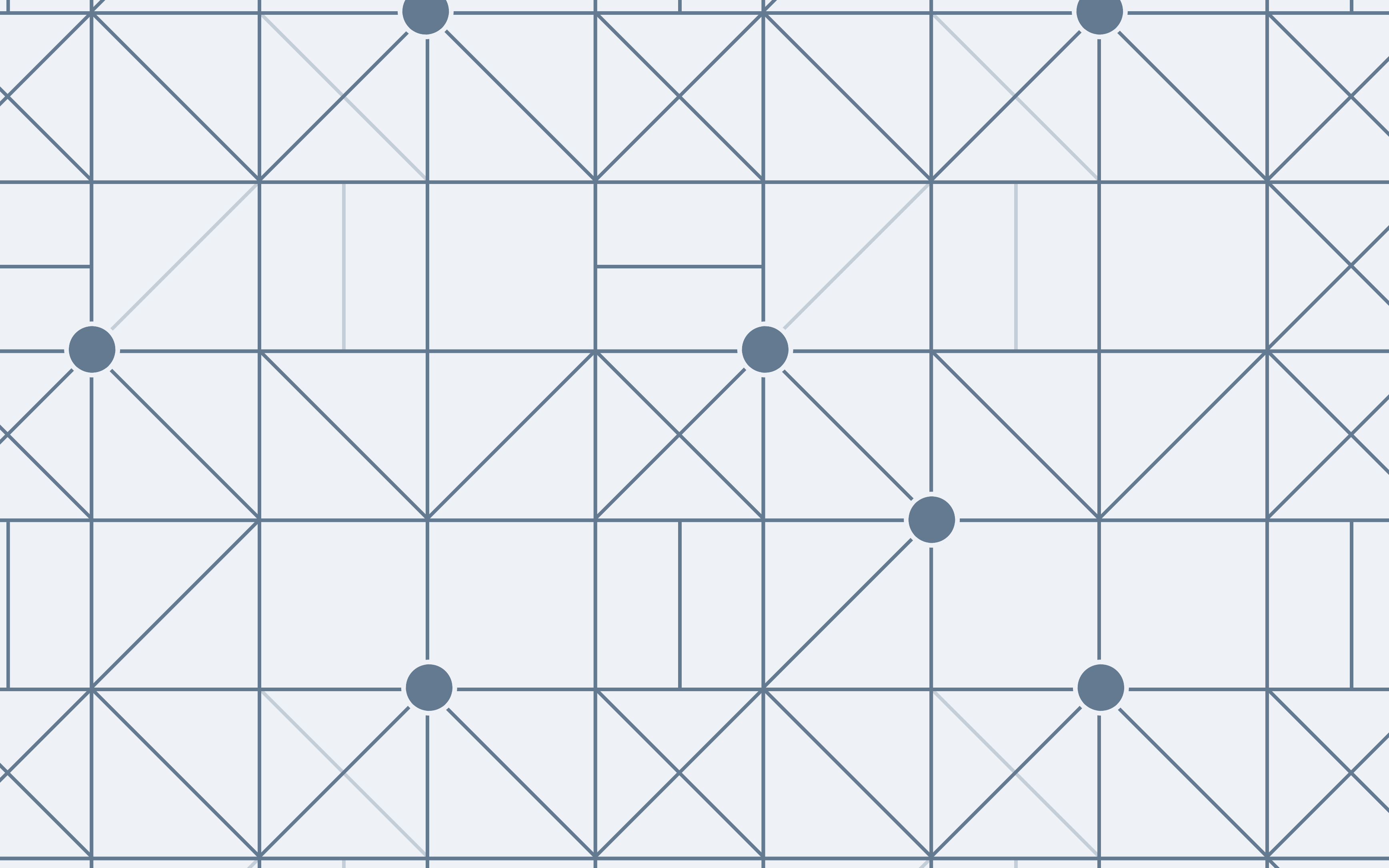
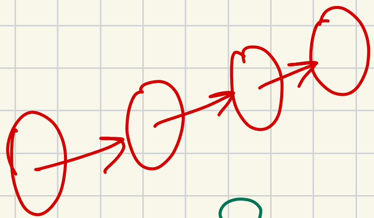


Binary Trees

→ Non-linear Data Structure



linear DS

Data:

- ↳ org. structure
- ↳ file system
- ↳ family tree

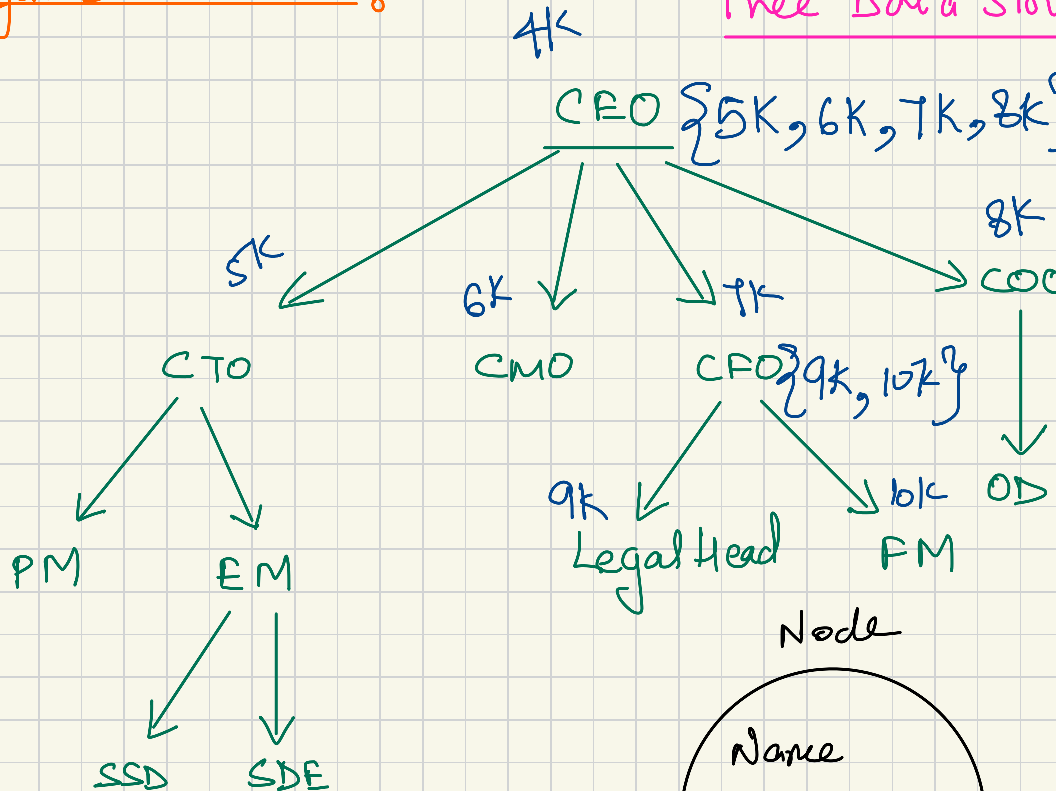
Heirarchy.

file system

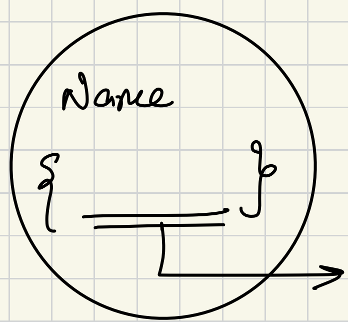
- Accio
  - M3 Dec
  - M3 Jan
    - Stacks
      - Q1.Java
      - Q2.Java
    - Queues
    - Binary Search

Organization Chart.

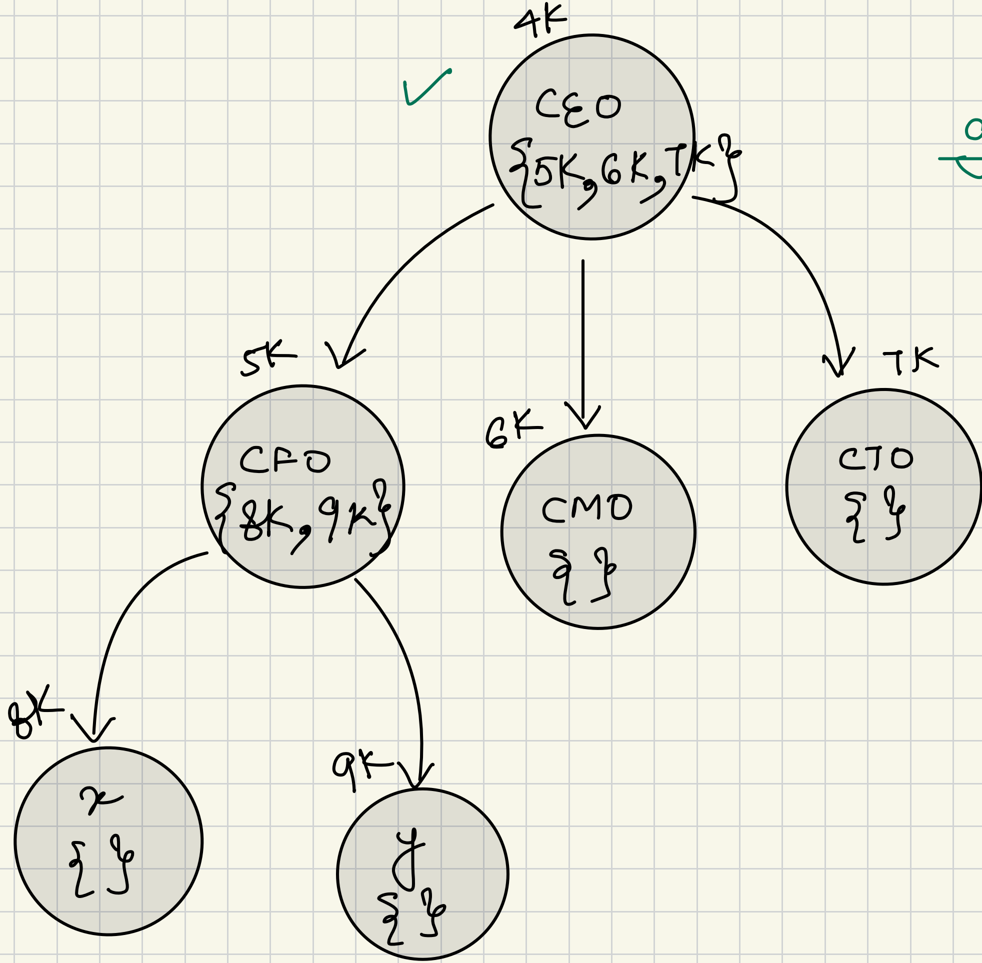
Tree Data Structure



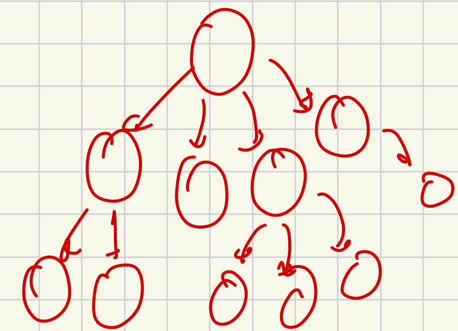

```
class Node
{
    String Name;
    ArrayList<Node> child;
}
```



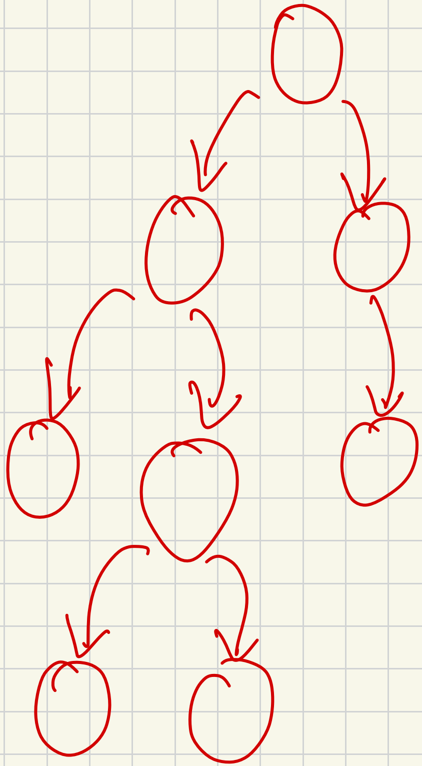
array of addresses reporting }



generic tree
↳ each node can have N child node.



population chart



{ Each person (Node) can have atmost 2 child (Node).

Binary Tree Data Structure

- Each Node atmost 2 nodes.

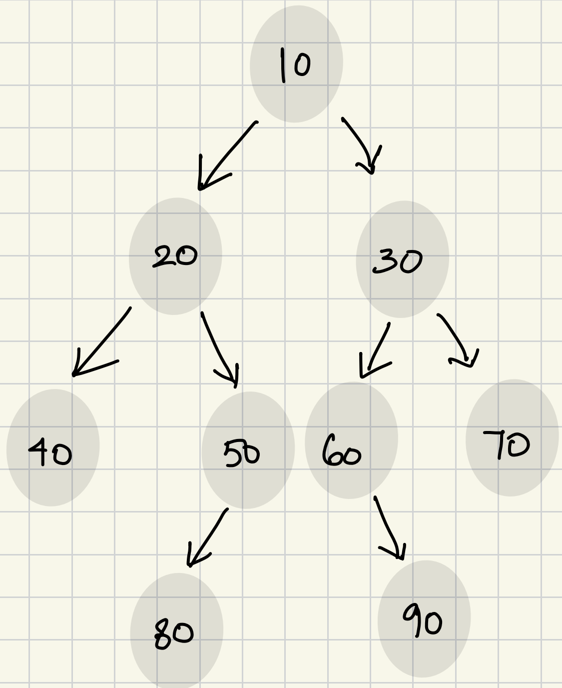
↓

i.e. 0, 1 or 2 childs

Binary Tree.

2 childs

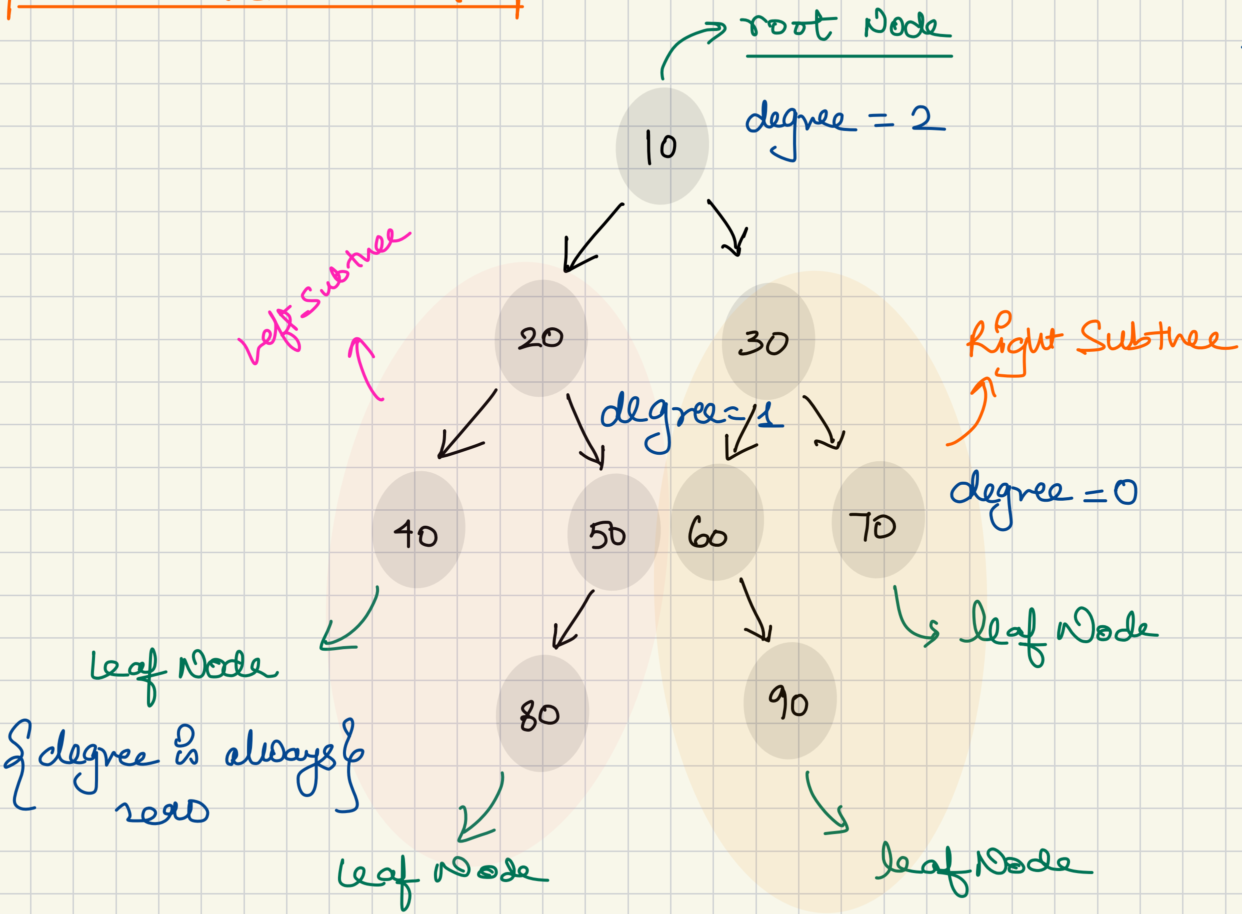
Left Right



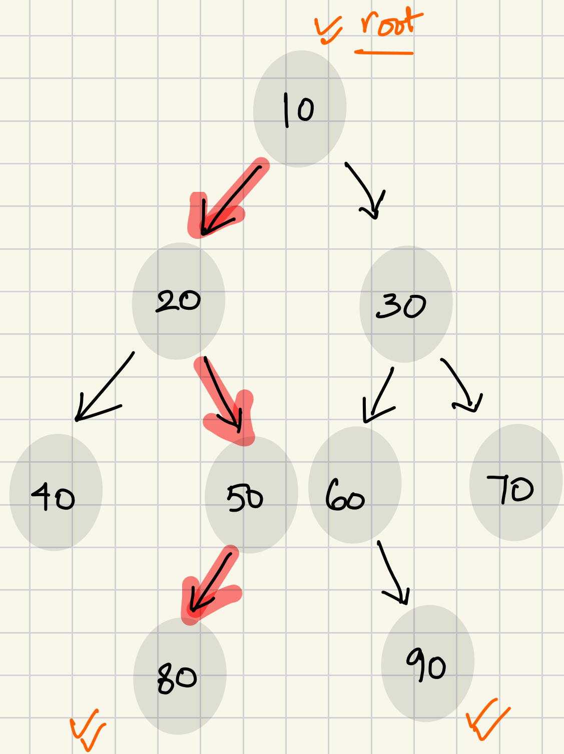

```
class Node
{
    int data;
    Node left;
    Node right;
}
```

parent - child relationship

degree
↳ No. of childs



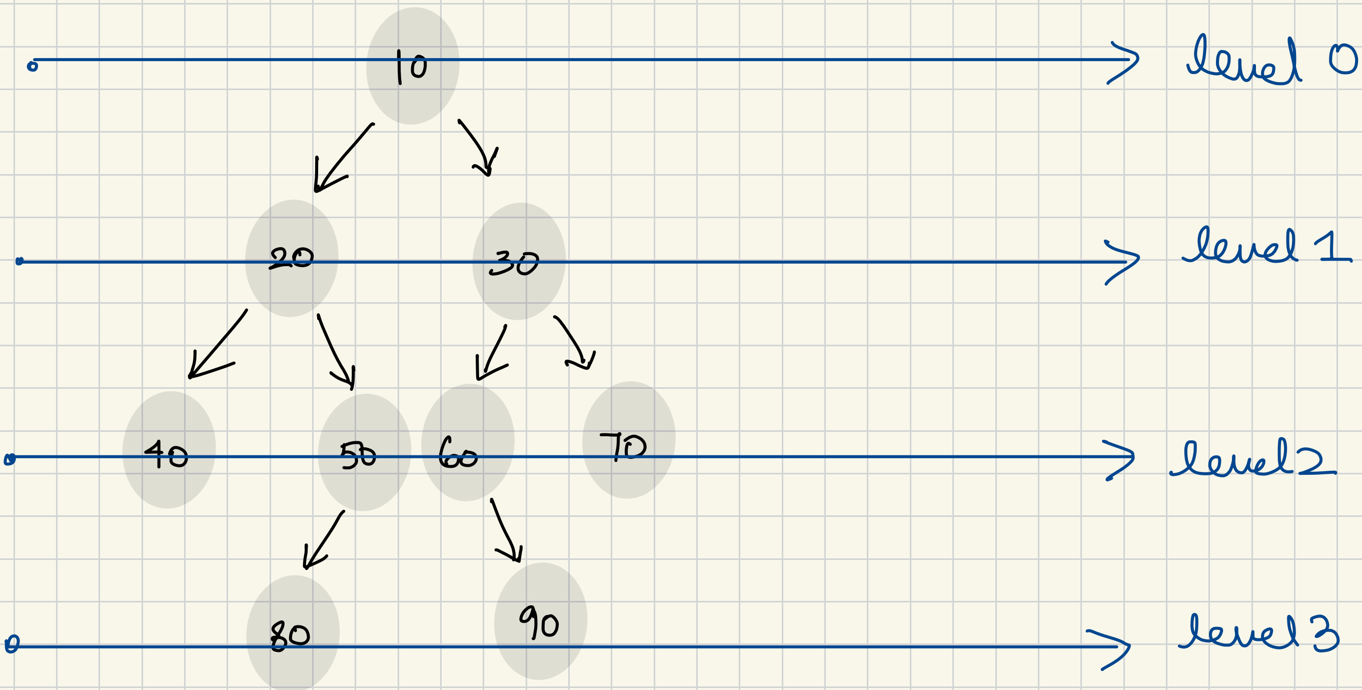
Height of a Binary Tree. {Distance b/w root node and deepest leaf Node}



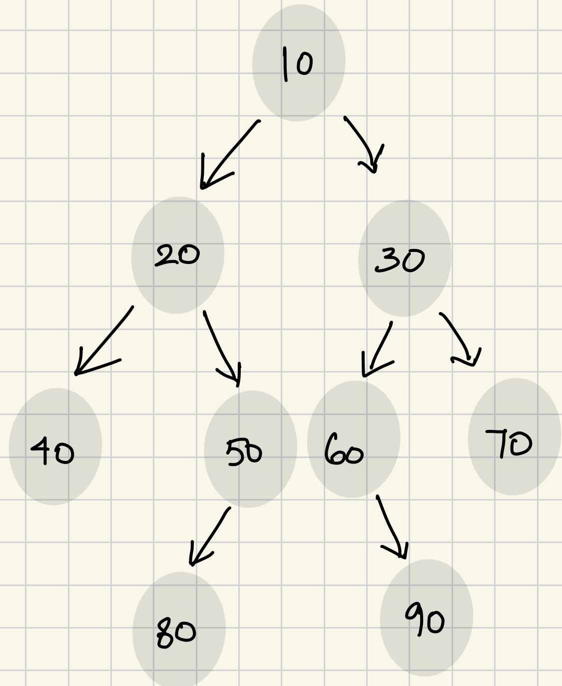
height = 4 {in terms of Node

height = 3 {in terms of Edges

Levels in a Binary Tree.



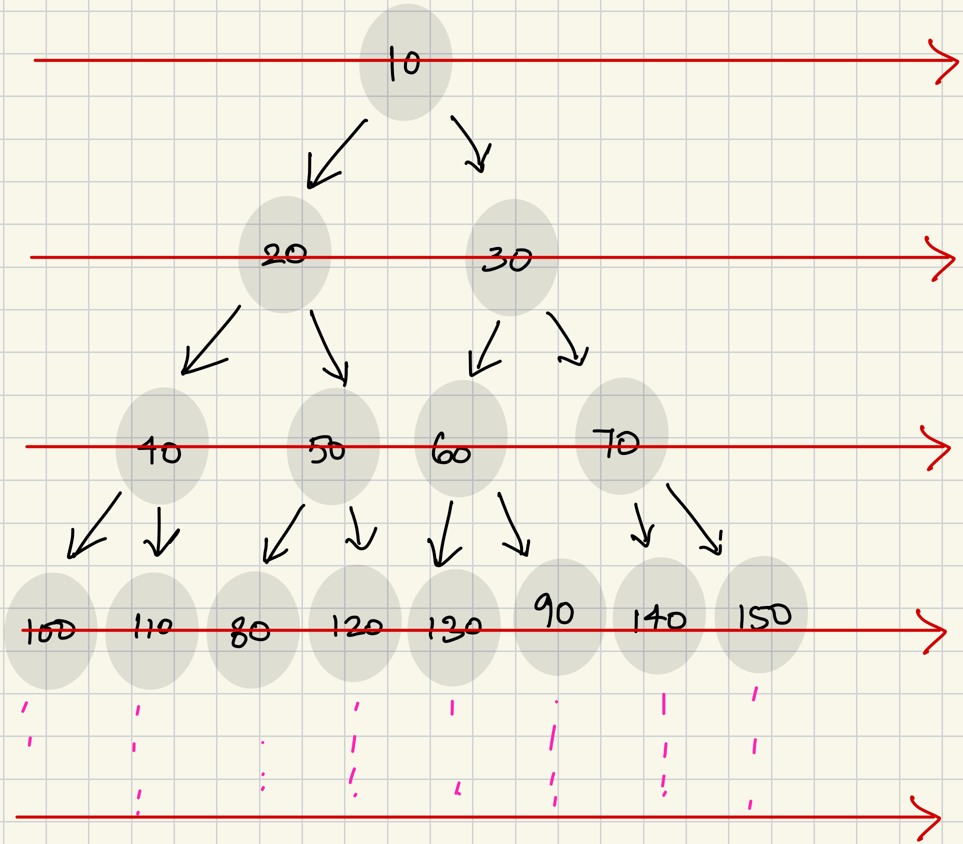
Binary Tree Image {For teaching purpose}



Ignore this page

Perfect Binary Tree.

$\{$ No. of Nodes at any level (l) $= 2^l \}$



level 0 = 1 2^0

level 1 = 2 2^1

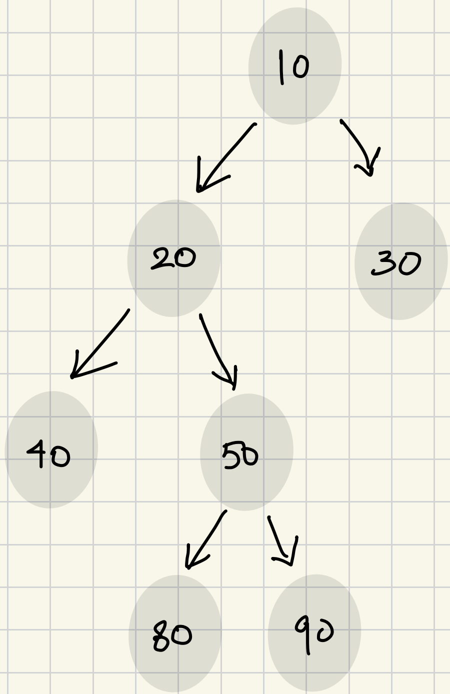
level 2 = 4 2^2

level 3 = 8 2^3

level K = 2^K

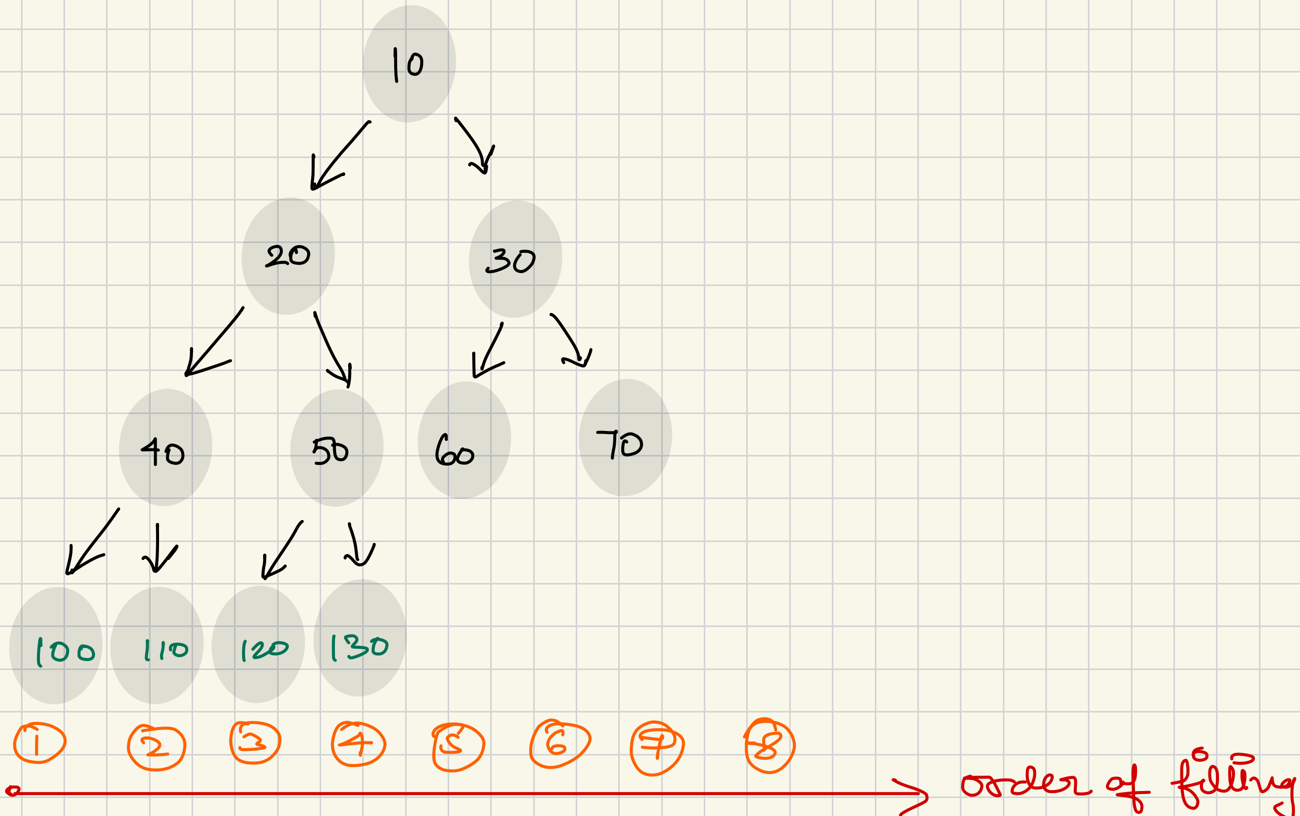
Full Binary Tree.

↳ where each Node have either 0 or 2 childs



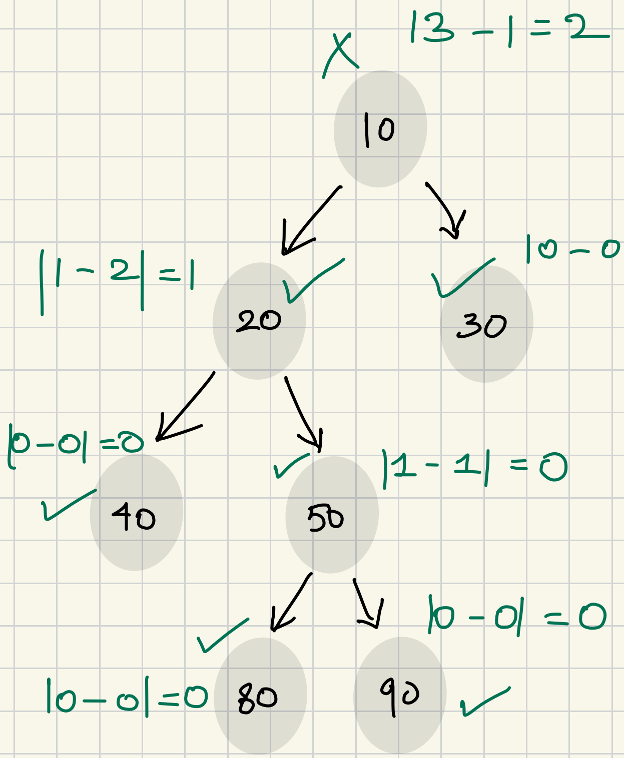
Complete Binary Tree

Where each level is completely filled, except last level, where nodes are as left positioned as possible



Balanced Binary Tree.

A Binary tree where each node is balanced.



Balanced Node.

abs. diff. of LST height and RST height ≤ 1

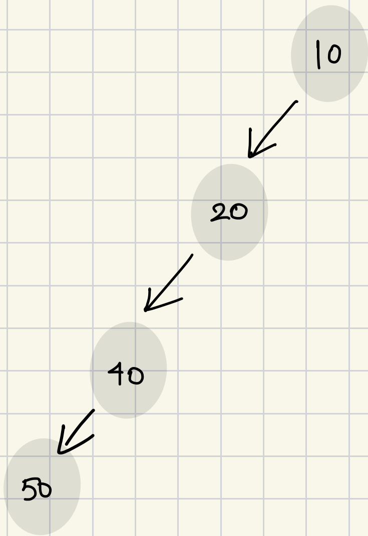
$$\left| height(LST) - height(RST) \right| \le 1$$

This tree, is not balanced!

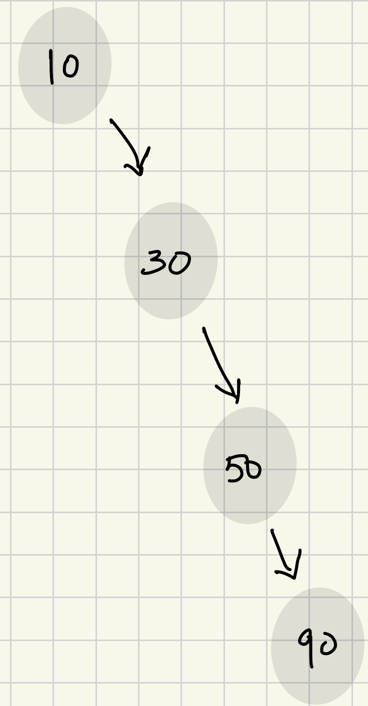
Skew tree.

1. Left Skewed Tree

{a node can have left child or no child}



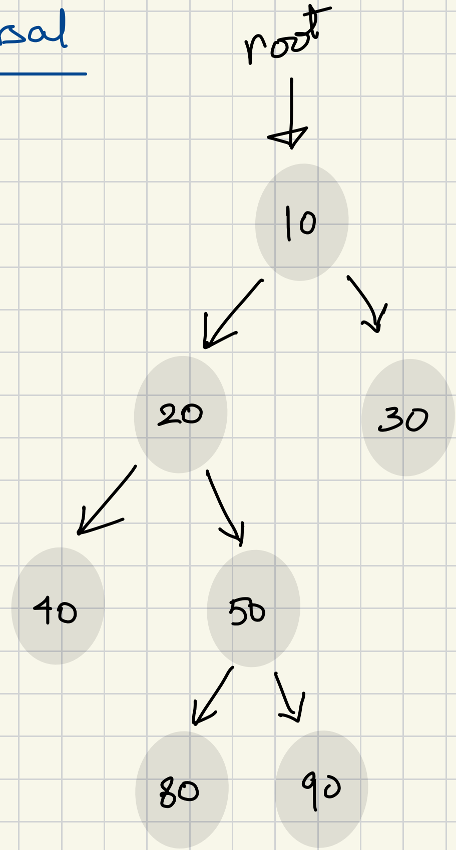
2. Right Skewed tree



Traversal Over a Tree.

• Pre order traversal

- print root
- pre-order LST
- pre-order RST



root LST RS RST

preOrder: 10, 20, 40, 50, 80, 90, 30

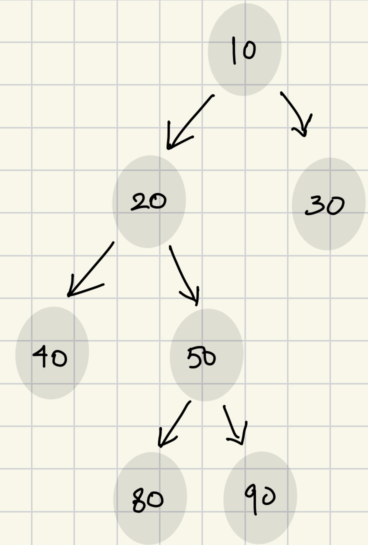
root LST

Recursion.

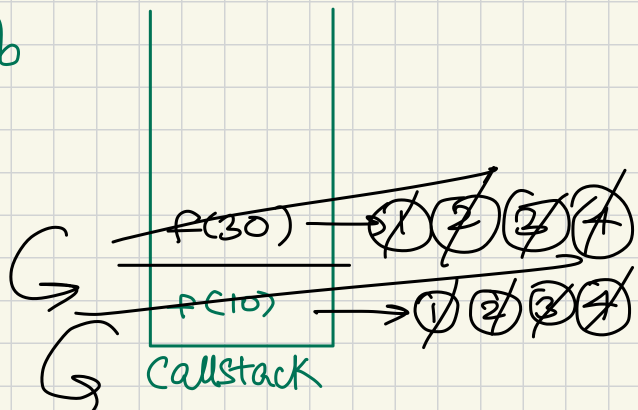
faith: prints preorder of a tree starting from root

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

    print(root.data);

    printPreOrder(root.left)
    printPreOrder(root.right)

}
```

```java
/*
    preorder Binary tree = print (root's data) + preorder (LST) + preorder (RST)

    Faith: prints preorder of a binary tree starting from given root
**/
public static void preorderTraversal(Node root) {
    // base case
(1) if (root == null) {
        return;
    }

(2) System.out.print(root.data + " ");

    // print preoder of LST
(3) preorderTraversal(root.left);

    // print preorder of RST
(4) preorderTraversal(root.right);
}
```



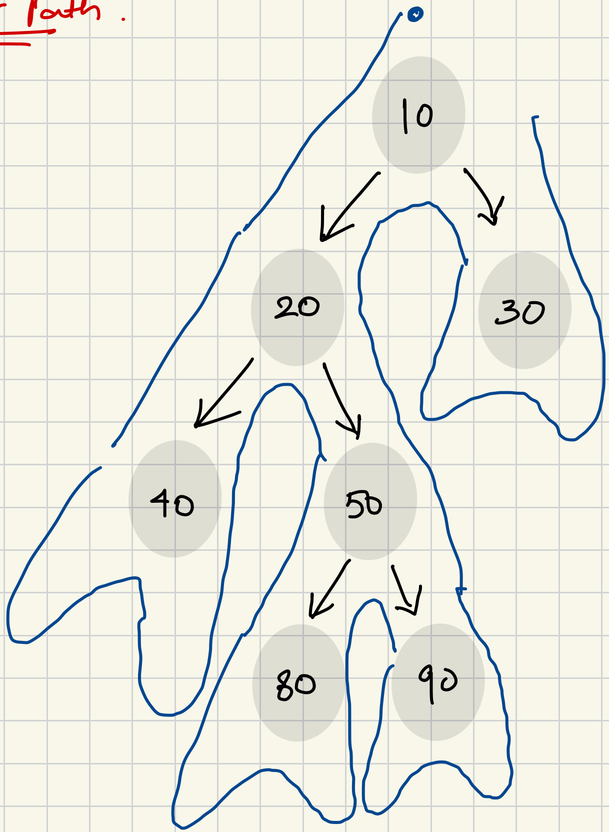
$$\left.\begin{aligned} TC &: O(N) \\ SC &: O(H) \end{aligned}\right\}$$



$\{10, 20, 40, 50, 80, 90, 30\}$

technique to Preorder quickly

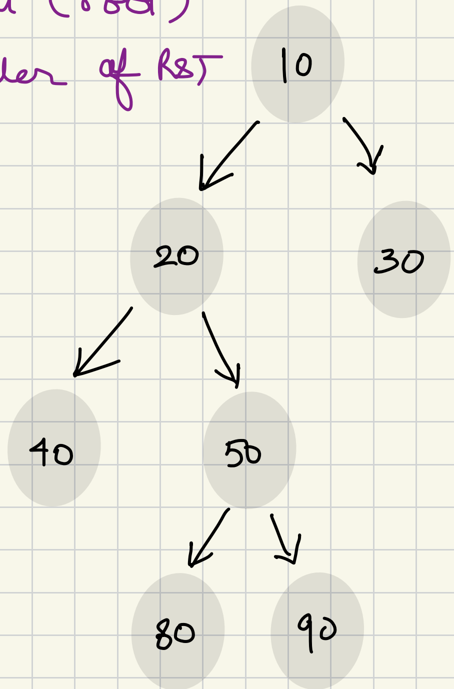
Eular Path :



$\{10, 20, 40, 50, 80, 90, 30\}$

In order traversal

Inorder of LST
print (root)
Inorder of RST

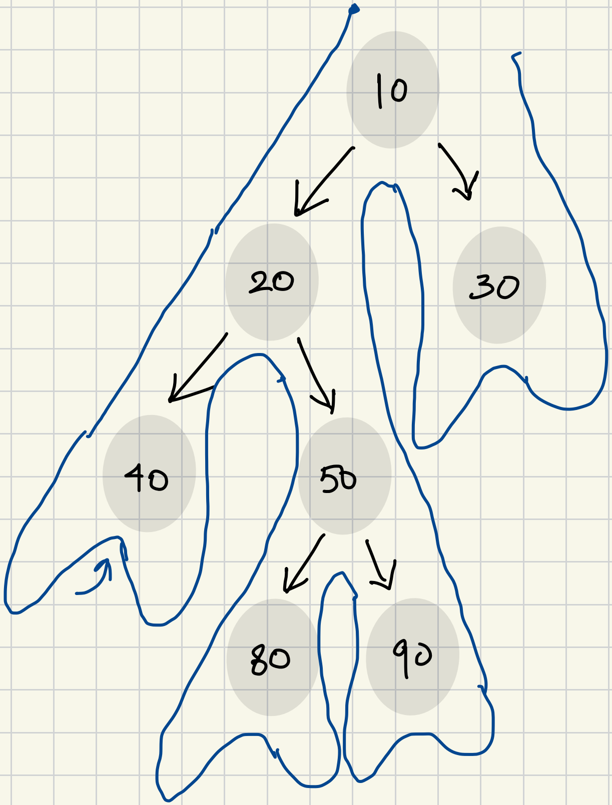


Inorder traversal : 40, 20, 80, 50, 90, 10, 30

LST root RST

LST root RST

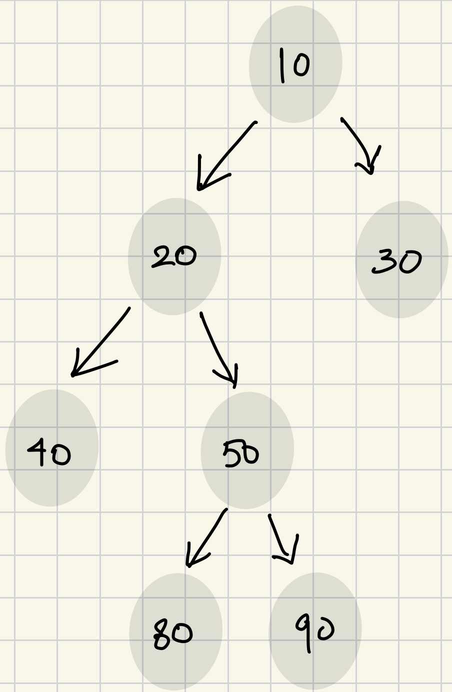
Eular Path



{40, 20, 80, 50, 90, 10, 30}

Post order traversal

postOrder (LST)
postOrder (RST)
root



LST RST root

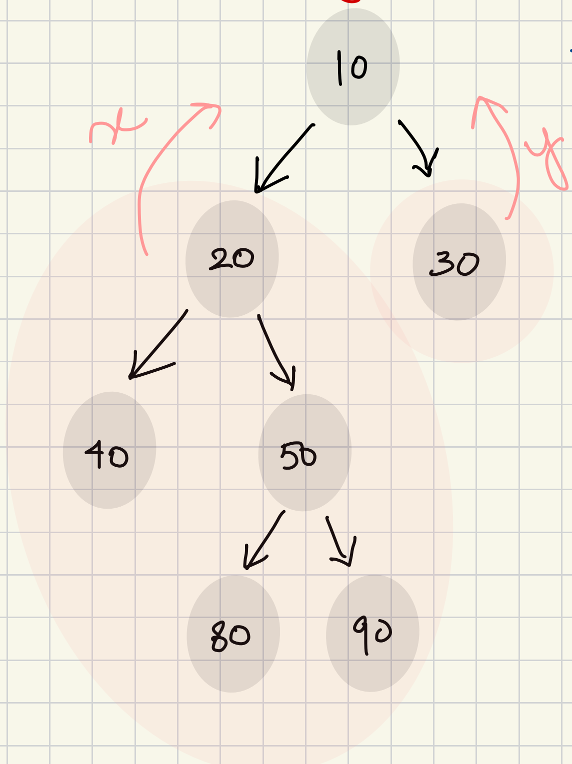
postorder : 40, 80, 90, 50, 20, 30, 10

LST RST root

Size of a binary tree

→ No. of Nodes in a binary tree

$size = x + 1 + y$



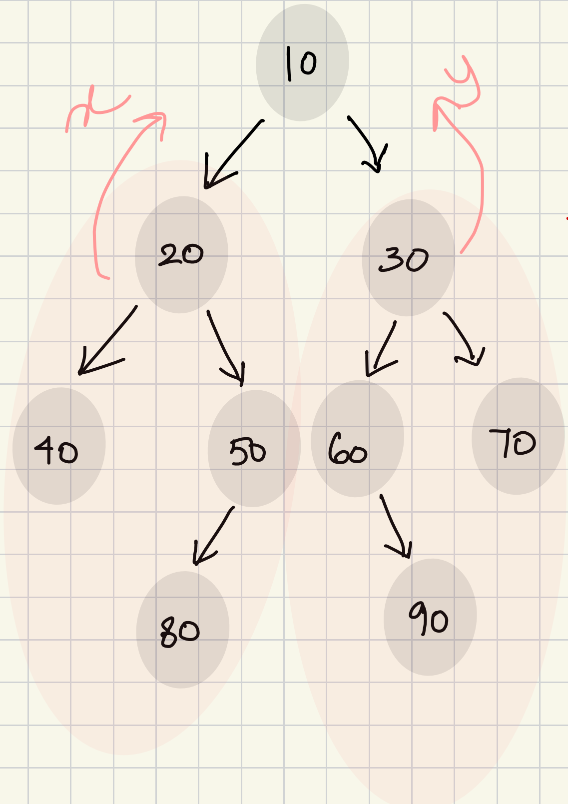
faith: return size of the bt staring from given root

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

    int x = size(root.left);
    int y = size(root.right);
    return x+1+y;
}
```

Sum of a Binary Tree

sum of data of all Nodes in a binary tree

$sum = x + 10 + y$



sum = 450

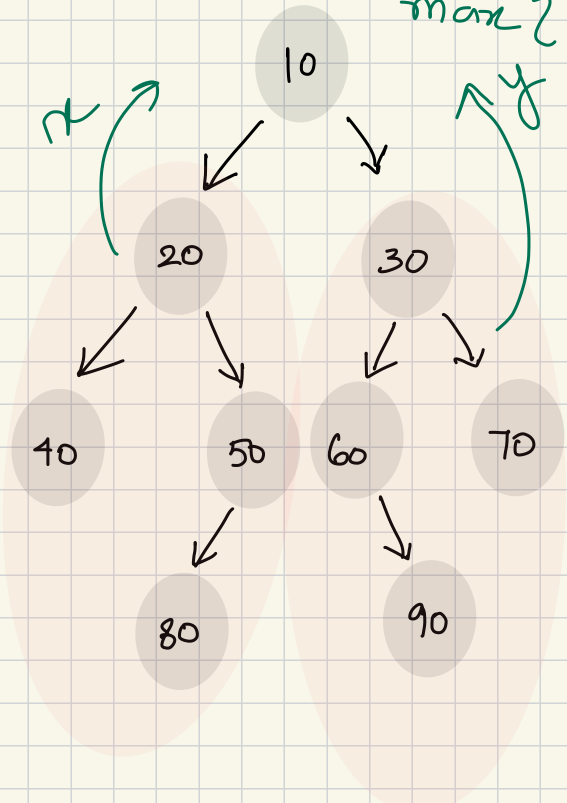
faith: returns sum of data of all nodes in bt from the given root

```
int sum(Node root)
```

Maximum in a Tree

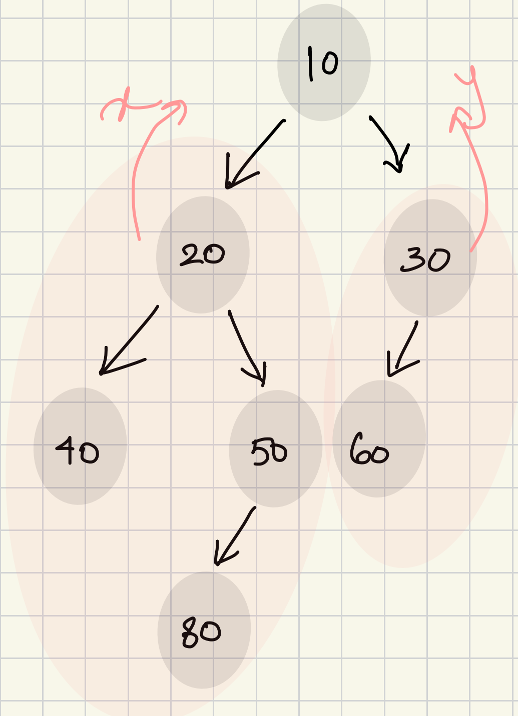
→ maximum value present in a tree

$\max\{x, y, r.data\}$



Height of a binary tree

$$h = \max(x, y) + 1$$



faith : returns height of the bt from given root

```
int height(Node root)
```