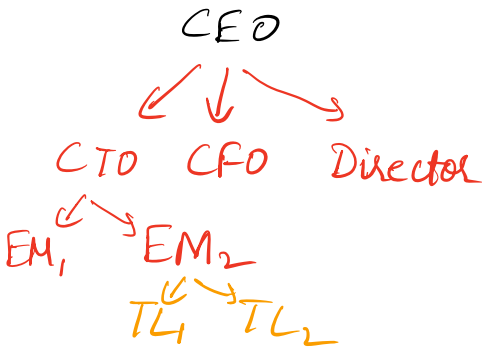
Arrays

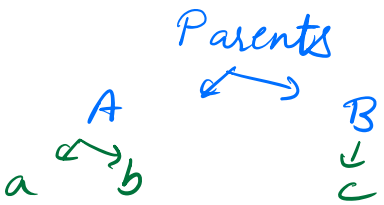
Linked Lists

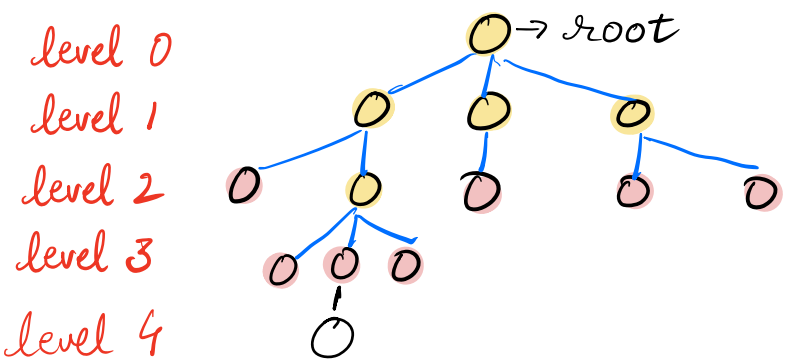
Hierarchy

Org structure



Family Structure





- Non Leaf Node
- Leaf Node

O → Node

/ → Edge

Relationships

- Parent
- Child
- Siblings
- Cousins

Only one parent for a node

multiple children

} Originating from same parent

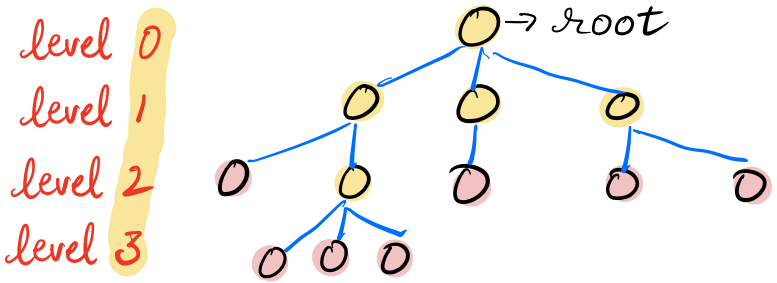
} Originating from same grand parent.

(same level is needed)

Depth

recursion

⇒ BFS

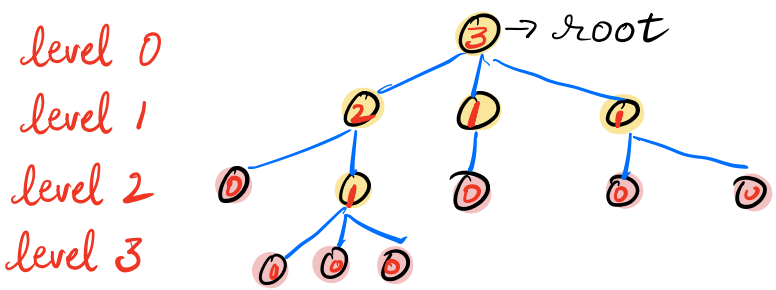


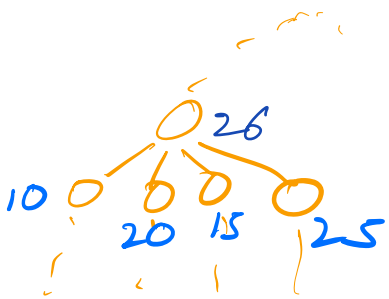
level = depth [Level is 0 index]

Depth (node) : Length of path from root to node

Depth (node) = Depth (parent) + 1

HEIGHT





Height (node) : Length of longest path from node to the deepest descendant leaf node.

Height (node) = 1 + max [Height (child)]

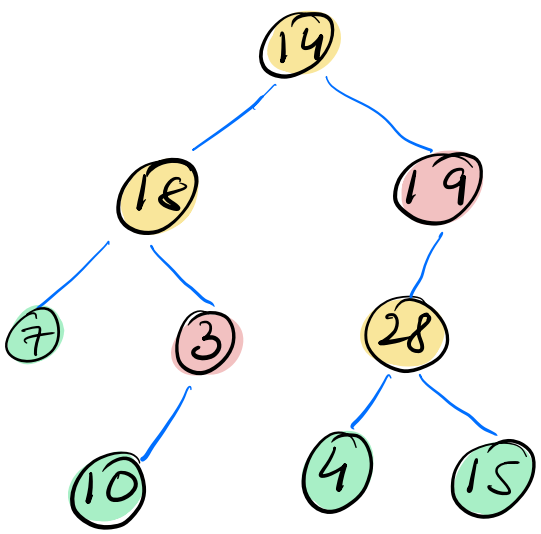
Height (leaf) = 0

Naming of trees

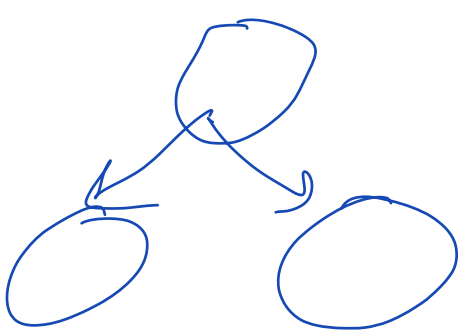
At max 2 children : BINARY TREE

At max 3 children : TERNARY TREE

At max N children : N-ary TREE



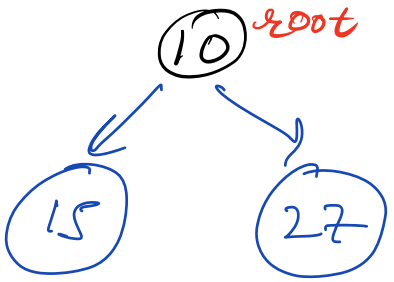
0 children
1 children
2 children

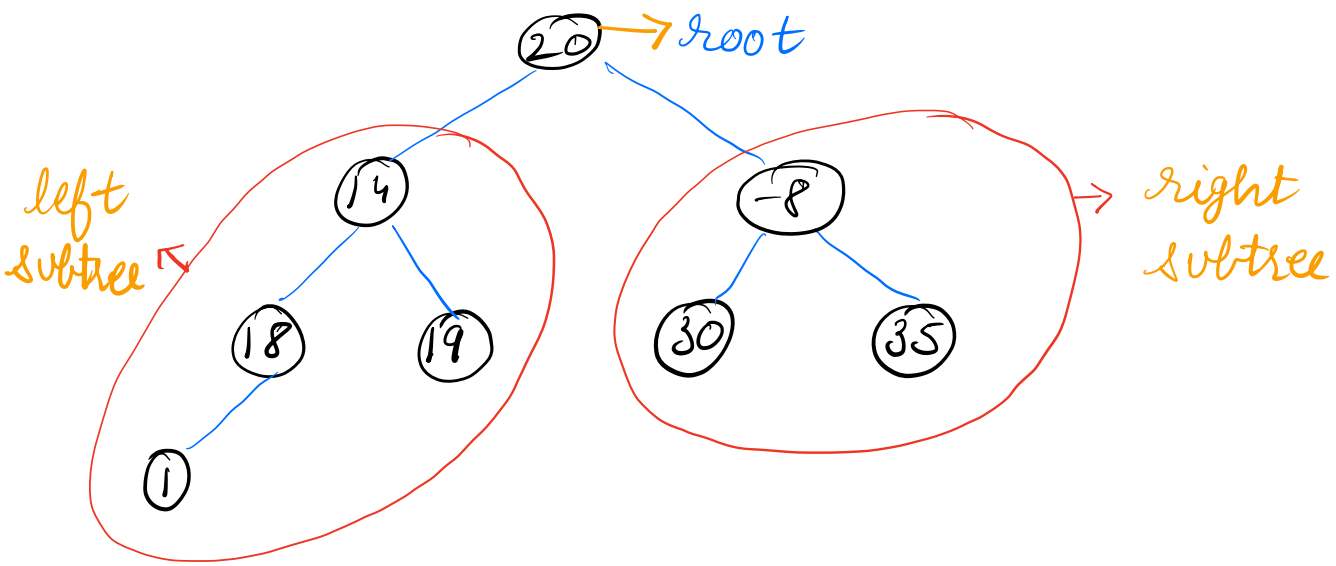


BINARY TREES

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

    Node(int x) {
        data = x
        left = null
        right = null
    }
}
```

```
Node root = new Node(10)
root.left = new Node(15)
root.right = new Node(27)
```





left subtree → Family of the left child.

Recursion

1) **Assumption**: Decide what your function does, and assume it does exactly that
2) **Main Logic**: Solving Assumption with subproblem
3) **Base Condition**: When should code stop.

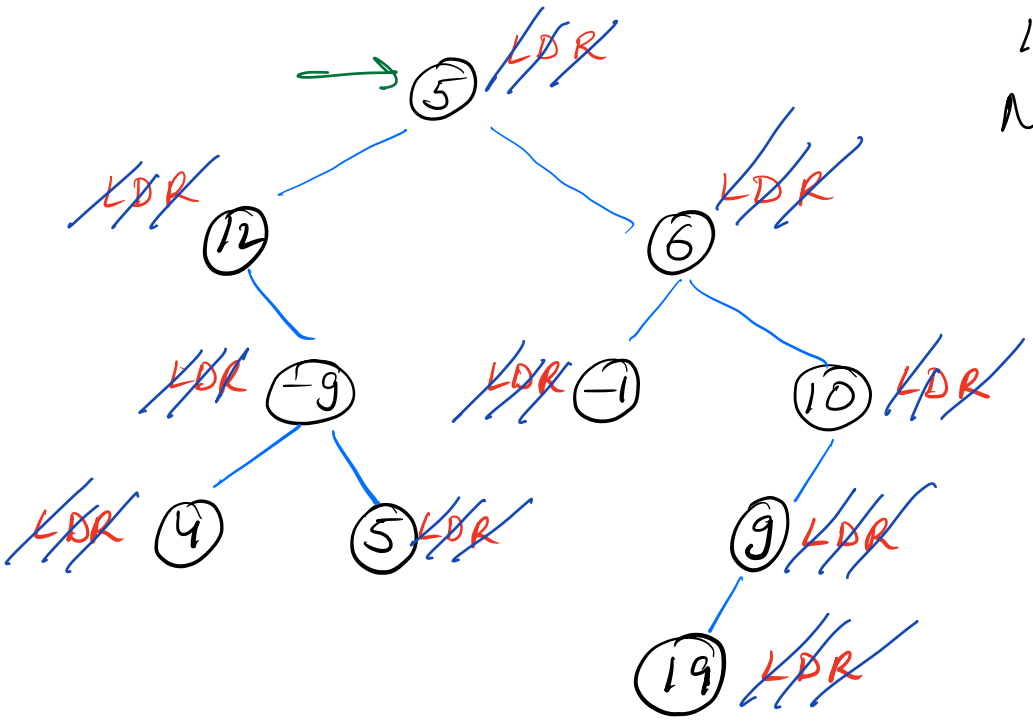
Tree Traversals

1) Preorder [data Left Right]

2) Inorder [Left data Right]

3) Postorder [Left Right data]

Inorder example

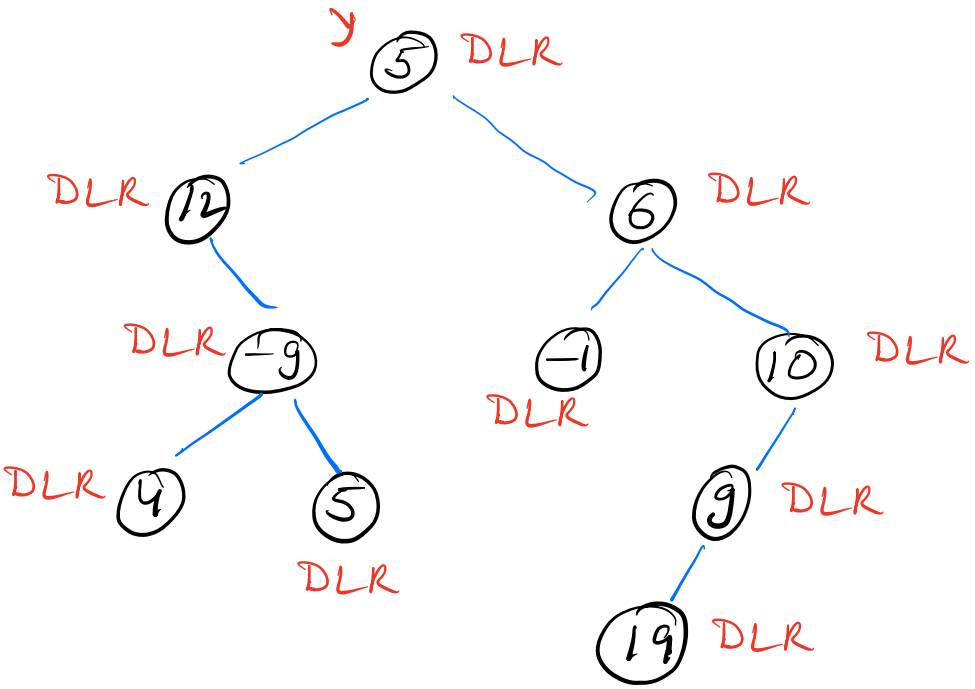


1 → 3

N → 3N

Ans: 12 4 -9 5 5 -1 6 19 9 10

Pre order example



Ans: 5 12 -9 4 5 6 -1 10 9 19

HW: Post order LR data

4 5 -9 12 -1 19 9 10

6 5

2 / 3

Q1 INORDER TRAVERSAL Code

Assumption: Print all node values in inorder fashion

Base Case: if node==null return

Logic: LDR
- left subtree
- right subtree

```
void inorder (Node node) {
    if (node == null)
        return
    inorder(node.left)
    print(node.data)
    inorder(node.right)
}
```

TC: O(N)

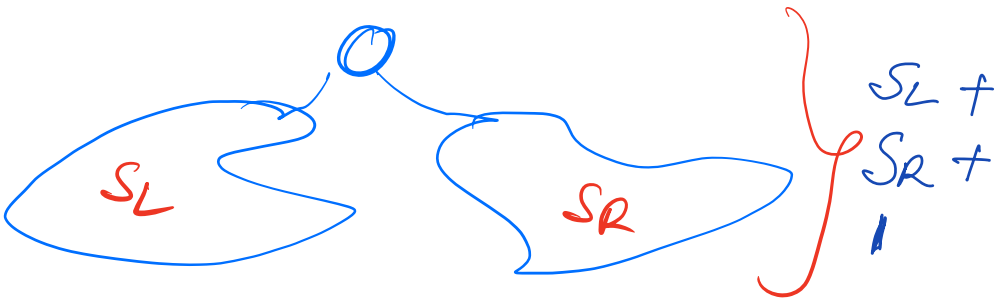
SC: O(N)

Q2 Given Binary Tree, calc size of tree

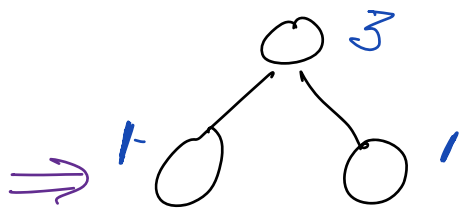
Assumption: Returns no of nodes in tree

Base Case: if root is NULL, ans = 0

Logic:




```
int size(Node node){
    if(node == NULL)
        return 0
    SL = size(node.left)
    SR = size(node.right)

    return SL + SR + 1
}
```

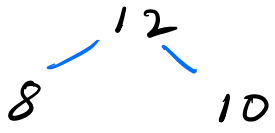


TC: $O(N)$

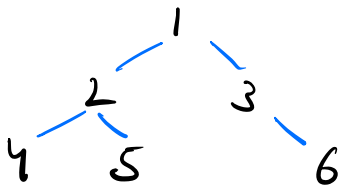
SC: $O(N)$

Q3 Given Binary Tree, find sum of node values of all nodes.

Eg:



ans = 30



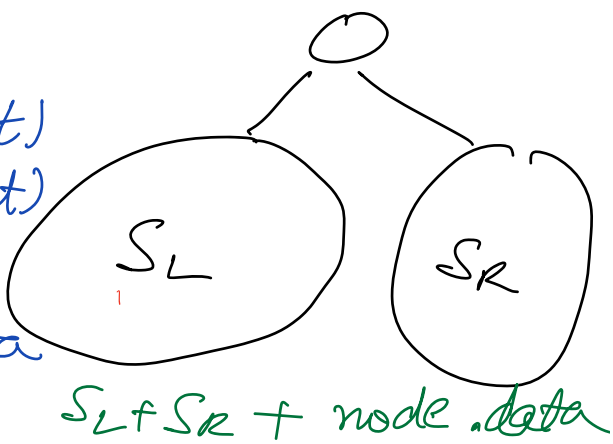
ans = 21

Assumption: Return sum of node values

Base Case: if root is NULL, ans = 0

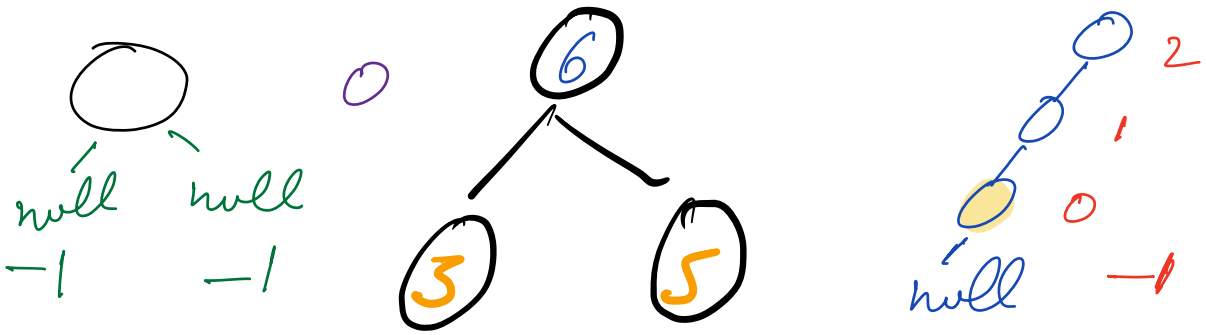
Logic:

```
int sum(Node node){
    if (node == NULL)
        return 0

    SL = sum(node.left)
    SR = sum(node.right)

    return SL + SR + node.data
}
```



TC: O(N) SC: O(N)

04) Height of a tree




```
int height (node) {
    if (node == null)
        return -1
    hl = height (node.left)
    hr = height (node.right)

    return 1 + max (hl, hr)
}
```

TC: $O(n)$
SC:

Edge 4