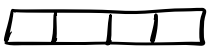
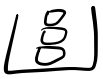
Arrays



Linked Lists

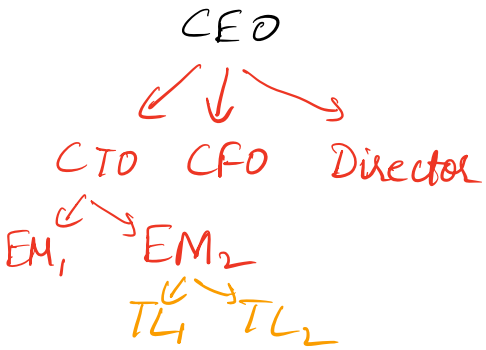
Stacks



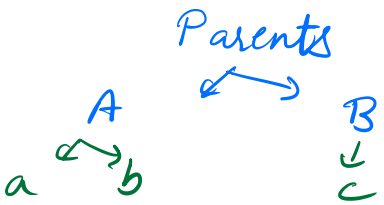
Queue

Hierarchy

Org structure



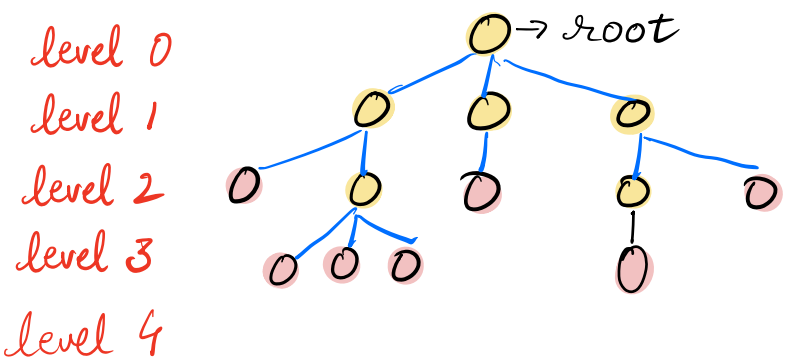
Family Structure



Vishal

|

Akshay



- Non Leaf Node
- Leaf Node
  - no children

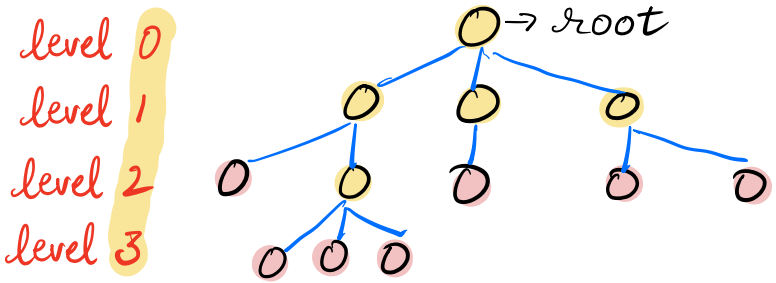
O → Node

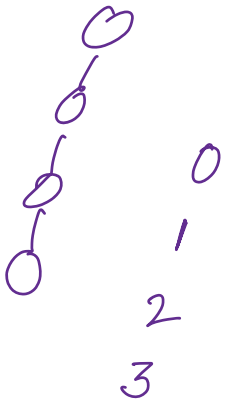
/ → Edge

Relationships

- Parent — Only one parent for a node
- Child — multiple children
- Siblings — originating from same parent
- Cousins — originating from same grand parent. (same level is needed)

Depth



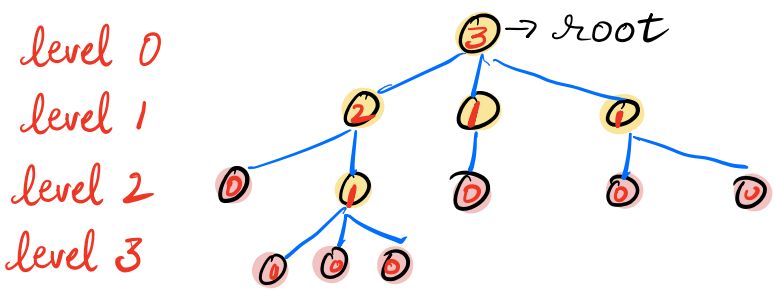


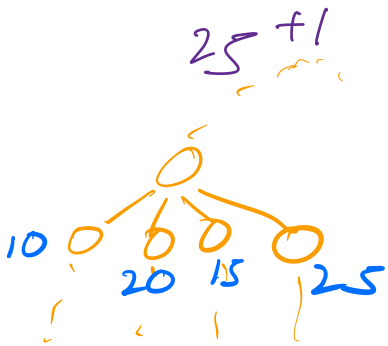
level = depth [Level is 0 index]

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

Depth (node) = depth (par) + 1

HEIGHT





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

Height (node) = max (height (child)) + 1

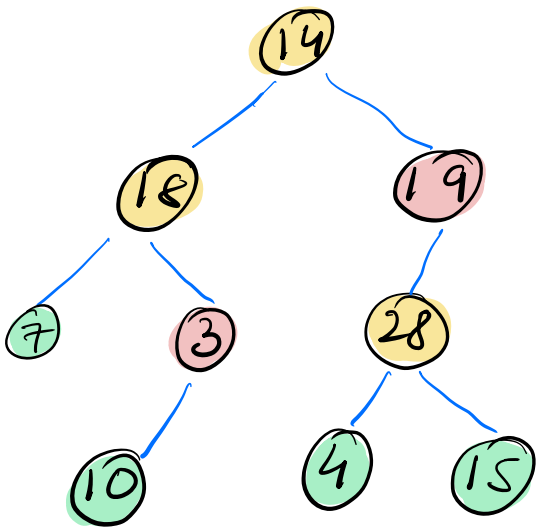
Height (leaf) = 0

Naming of trees

At max 2 children : BINARY TREE

At max 3 children : TERNARY TREE

At max N children : N-ary TREE

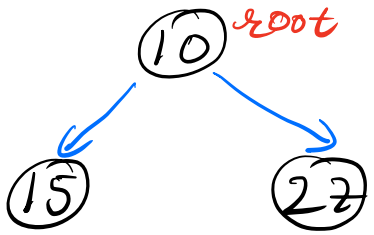


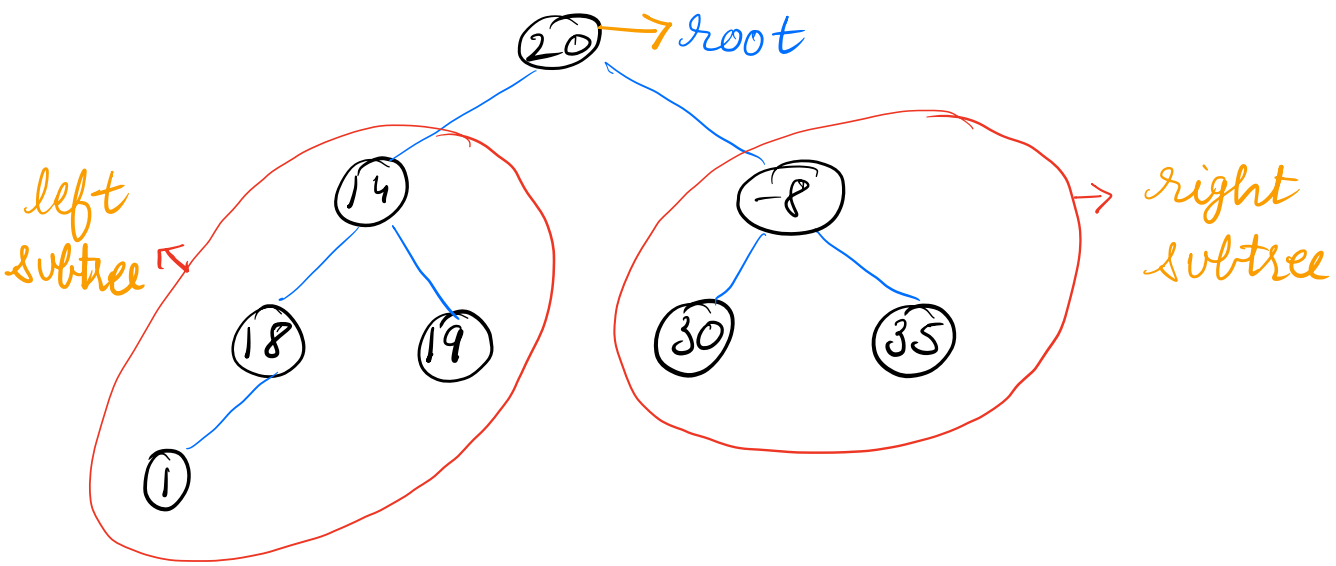
0 0 children
0 1 children
0 2 children

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

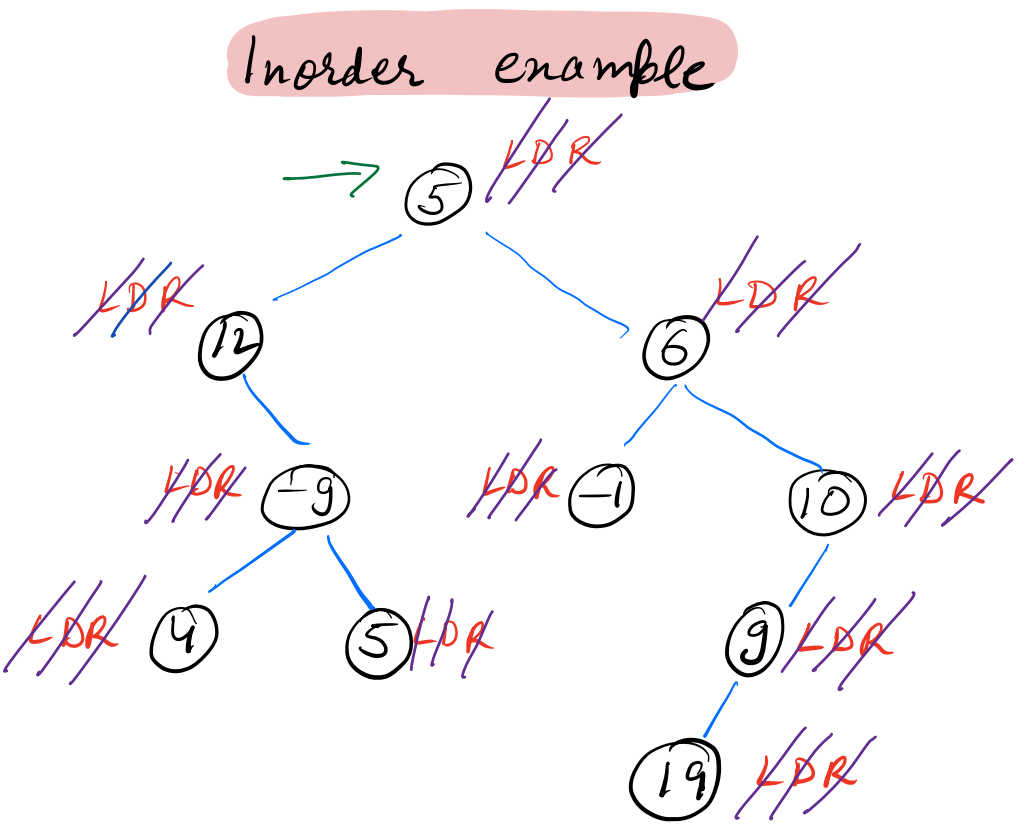
1) **Assumption**: Decide what your function does, and assume it does exactly that
2) **Main Logic**: Solving Assumption with subproblem
3) **Base Condition**: When should code stop.

Tree Traversals

1) Preorder [data Left Right] DLR

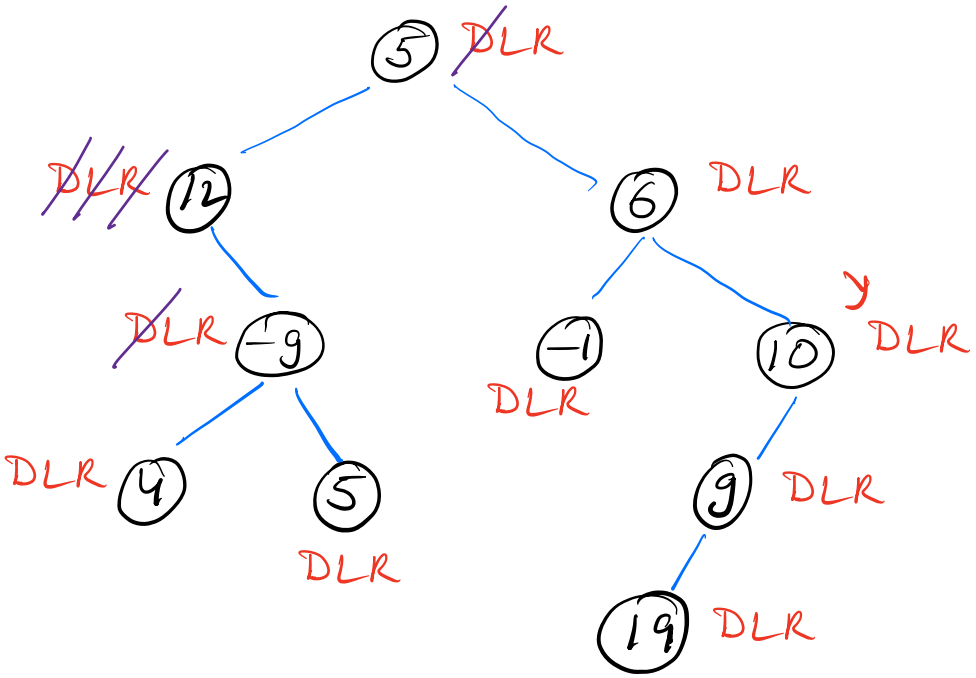
2) Inorder [Left data Right] LDR

3) Postorder [Left Right data] LRD



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

Postorder LRD

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

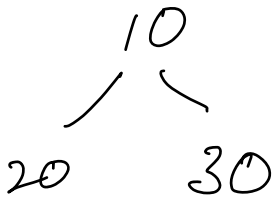
Q1 INORDER TRAVERSAL Code

Assumption: Print all node values in inorder fashion

Base Case: if root == null return

Logic: LDR
- left subtree
- right subtree

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



20 10 30

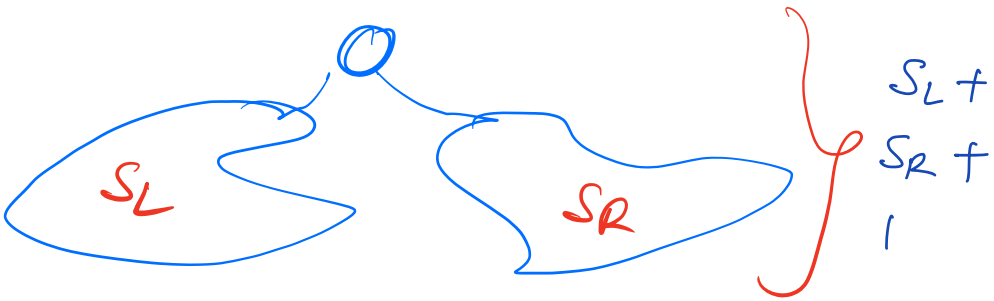
Going to parent ⟹ return statement in recursive function

Q2 Given Binary Tree, calc size of tree

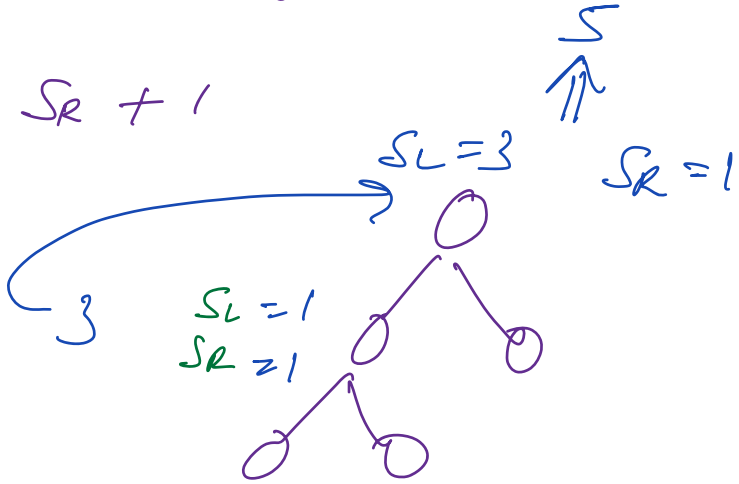
Assumption: Returns no of nodes in tree

Base Case: if root is NULL, ans = 0

Logic:



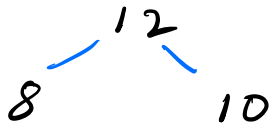

```
int size(Node root){
    if(root==NULL)
        return 0
    SL = size(root.left)
    SR = size(root.right)

    return SL + SR + 1
}
```



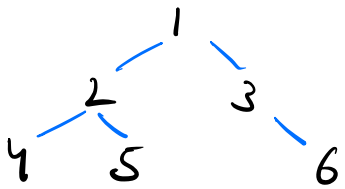
TC: O(N)
SC: O(N)

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

Eg:



ans = 30

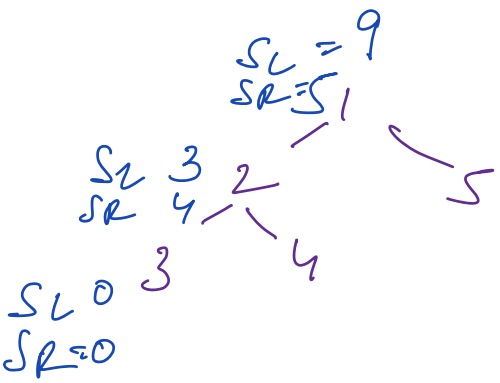


ans = 21

Assumption: return sum of all nodes

Base Case: if root is NULL, ans = 0

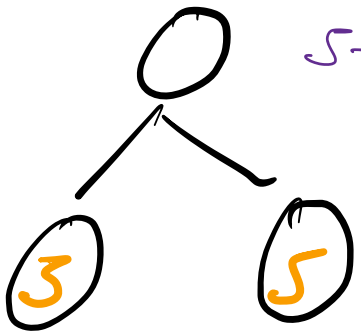
Logic:




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

    sum_l = sum(root.left)
    sum_r = sum(root.right)

    return sum_l + sum_r + root.data
}
```

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

04) Height of a tree




```
int height (root) {
  if (root == null)
      return 0
  hl = height (root.left)
  hr = height (root.right)

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

<done>