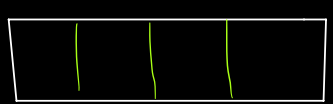
Today's Agenda :-

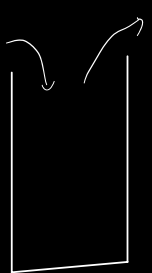
- Tree Basics & Terminologies
- Level, Height, Depth
- Binary Trees
- Tree Traversals
- Basic Problems

Slack ✓

Mob: 9521279429

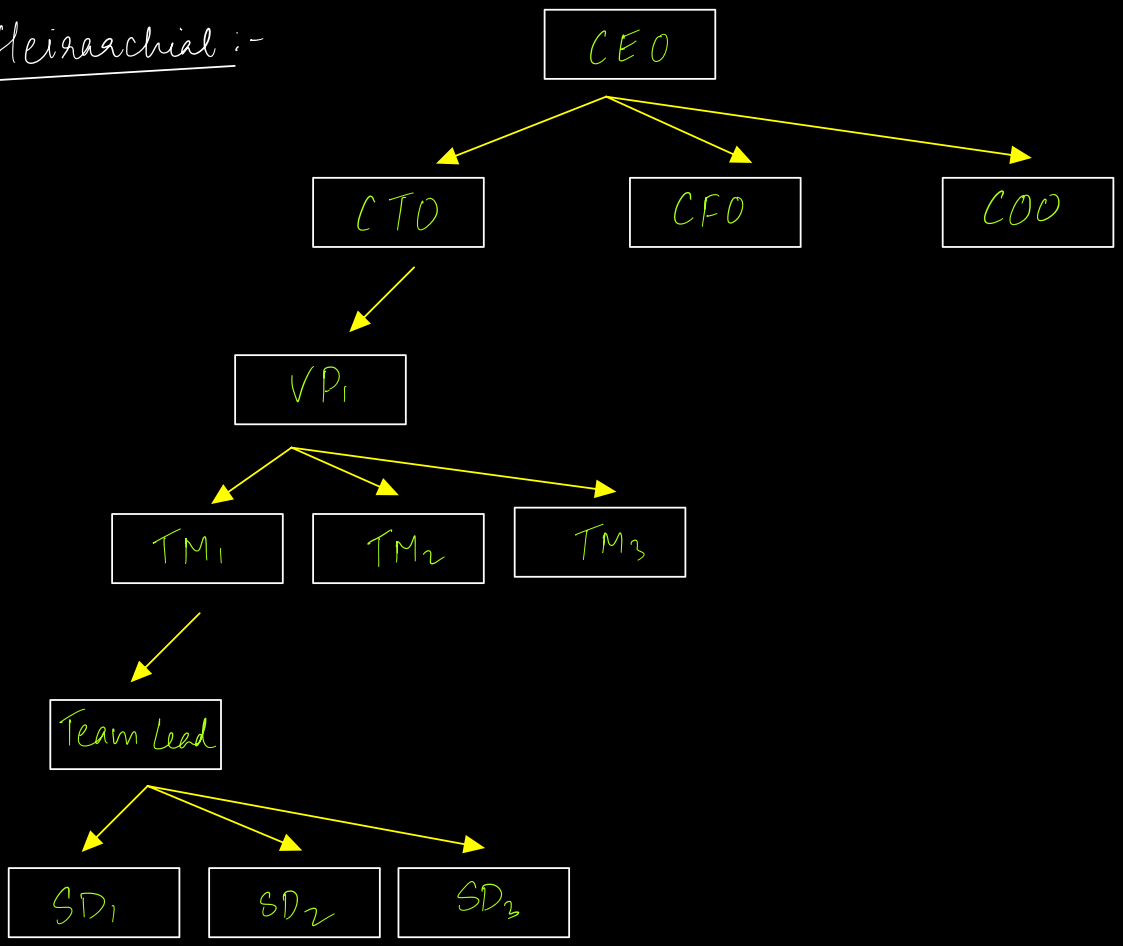
Email: umangcworld@gmail.com





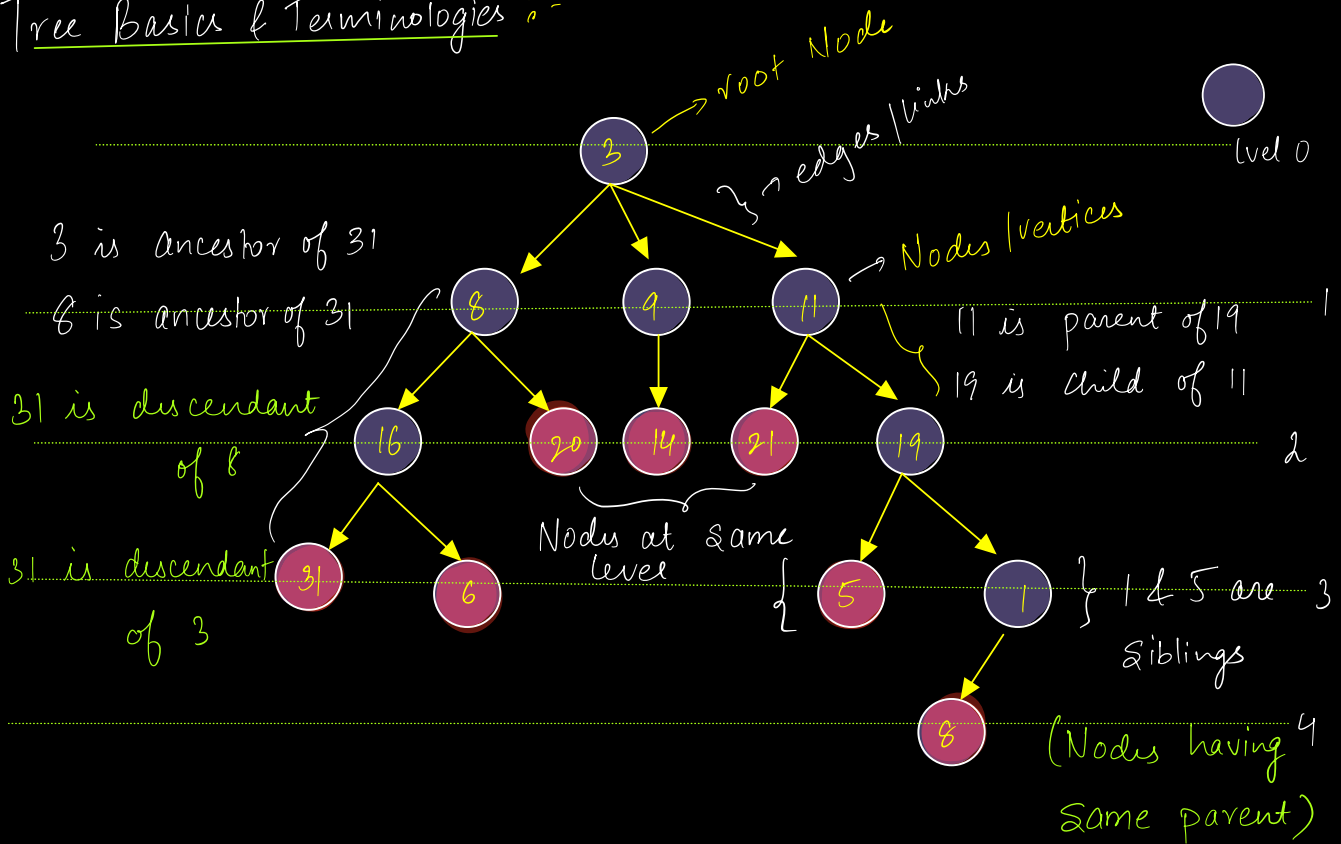
Linear DS

Heirarchial :-



1) Answers in Private chat
2) Questions / in Public chat } Doubts.

Tree Basics & Terminologies :-



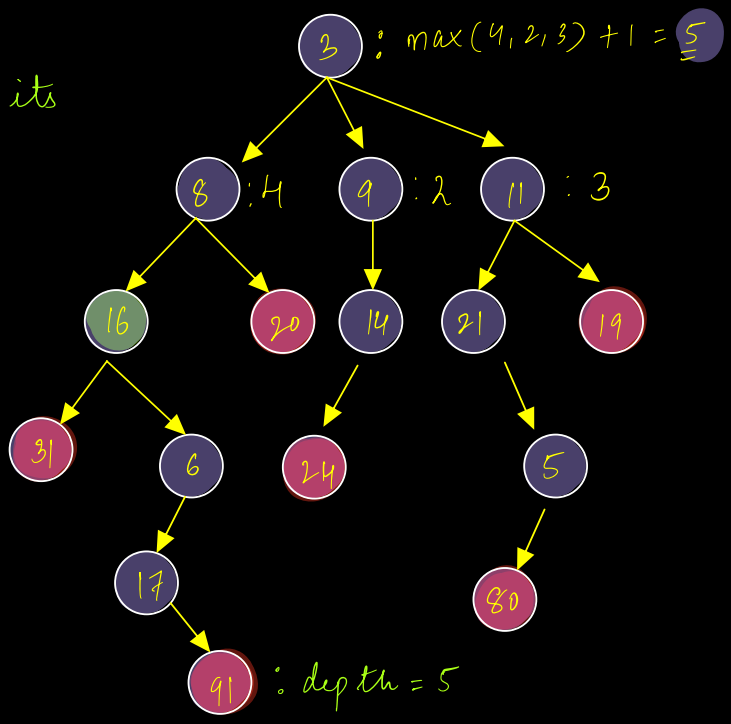
Root : is node with no parent

Leaf Nodes : Node with no child Nodes

Height of Node :-

Distance from the Node to its farthest leaf Node

(Distance : No of edges)



Height (8) = 4

Height (16) = 3

Height (21) = 2

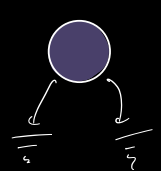
Height (5) = 1

Height (24) = 0

Obs1 :

Height (Node) = 1 + max (child Node heights)

Edge Case :-



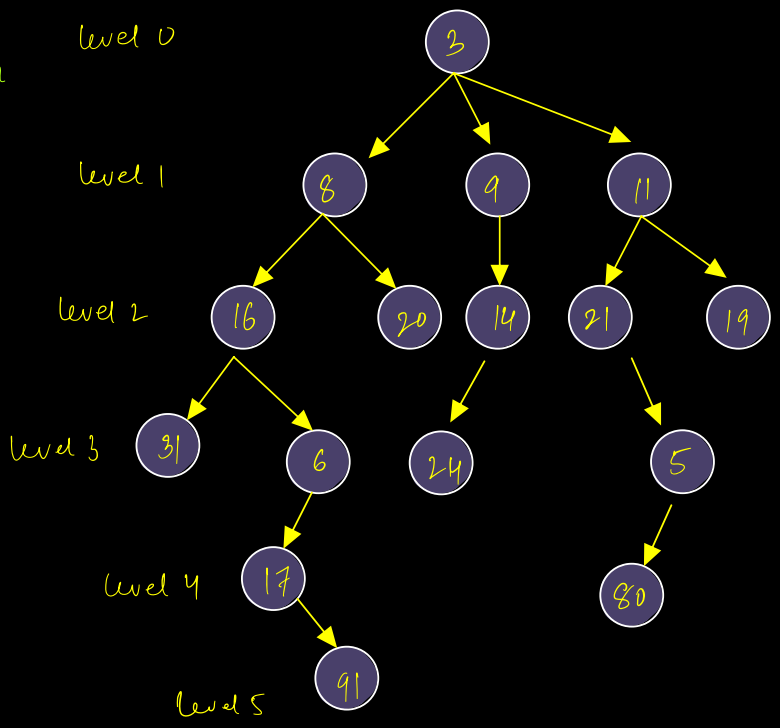
→ No child Nodes

= 1 + max (child Node heights)

Height of Tree = Height (root Node) = max (depth of any leaf Nodes)

Depth of a Node :-

Distance of a Node from root Node.



Depth (9) = 1

Depth (16) = 2

Depth (24) = 3

Depth (17) = 4

Depth (3) = 0

depth = level.

depth (Node) = d

↳ depth of all its child Nodes = d+1

depth (root Node) = 0

{ 8 Min break }

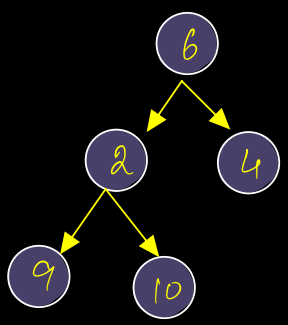
{ 2 = 2:30 }

by 9:30 AM

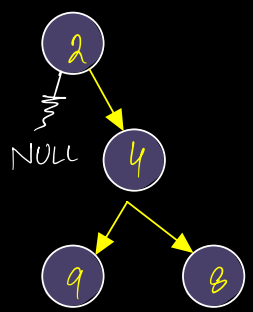
Binary Tree :- For every node, No of child Nodes ≤ 2

0, 1, 2 childs

Ex1



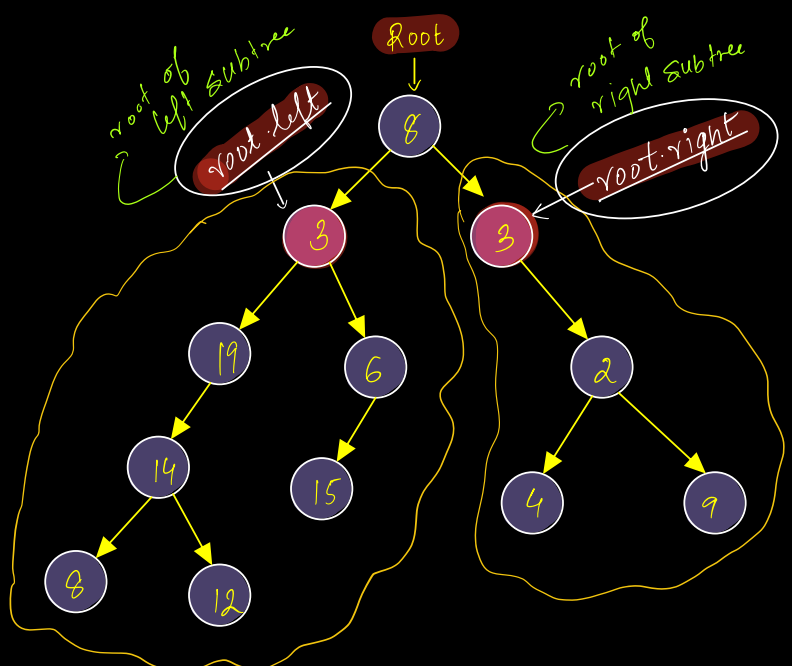
Ex2



(Node Structure)

```
Class Node {
    int data ;
    Node left ;
    Node right ;
    Node(x) {
        data = x
        left = NULL
        right = NULL
    }
}
```

Constructor



Recursion Basics :-

→ **Assumption** : Decide what your function does & assume that it works.

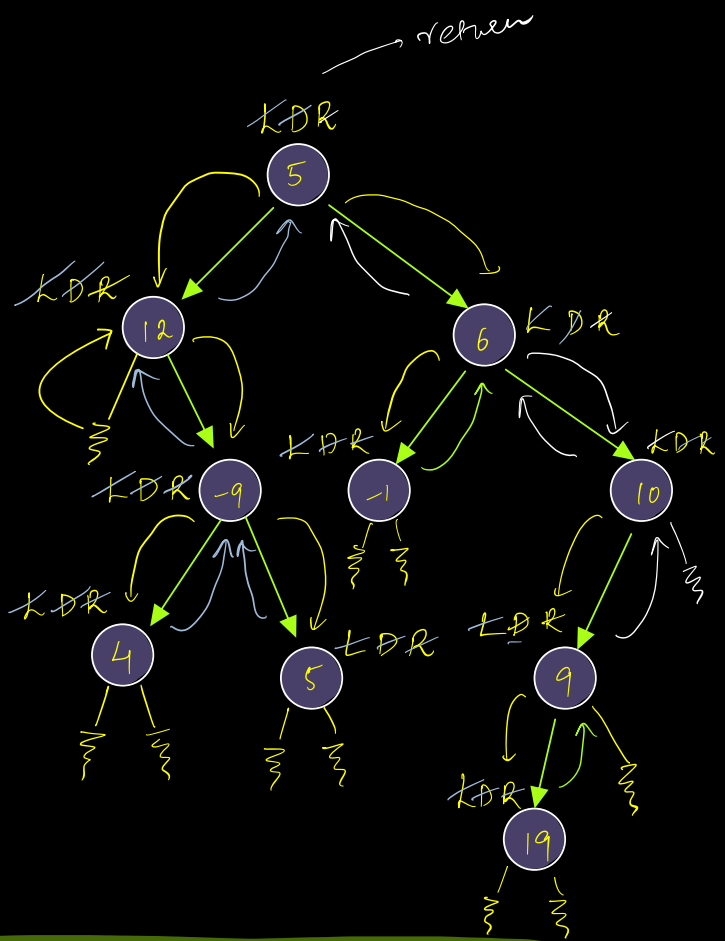
→ **Main Logic** : Solving problem using subproblem

→ **Base Condition** when to stop.

Tree Traversals :-

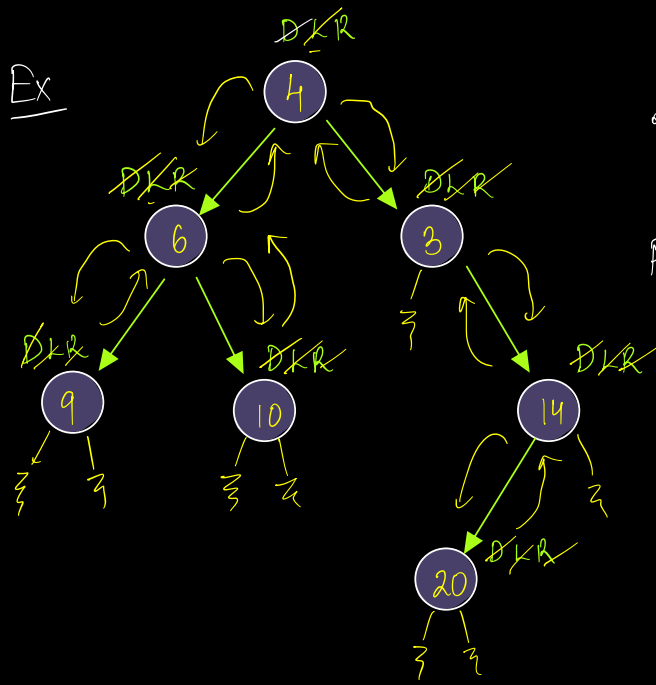
1) Preorder D L R
2) Inorder L D R
3) Postorder L R D

L → visit left subtree
D → current data
R → visit right subtree



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

Ex



inorder : 9 6 10 4 3 20 14

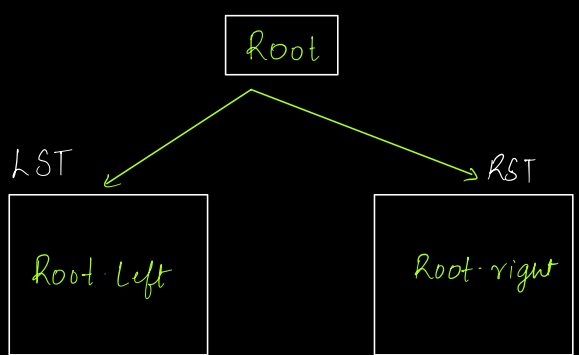
Preorder : 4 6 9 10 3 14 20

Postorder : TODO

LDR

Ass : (Print entire tree inorder)

```
void inorder (Node root) {
    1 if (root == NULL) { return }
    2 inorder (root.left)
    3 print (root.data)
    4 inorder (root.right)
}
```



//idea (LDR)

- print entire LST inorder
- print (root.data)
- print entire RST inorder

DLR

Preorder : 1 3 2 4

Postorder : 1 2 4 3

Inorder
Infix 2+5 3+4

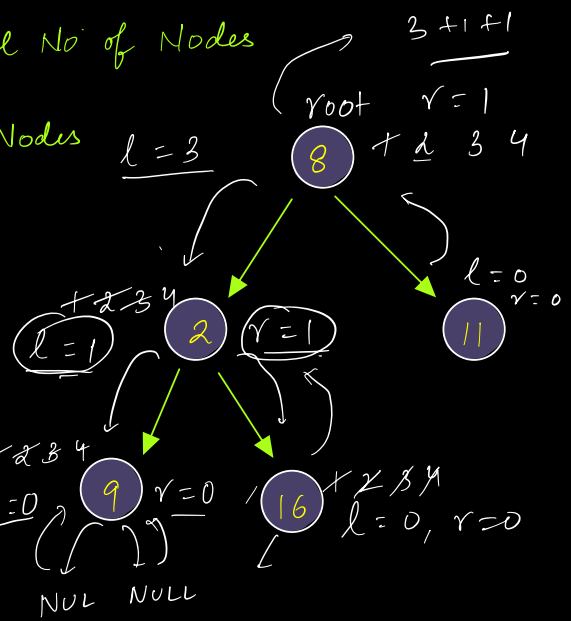
Preord
Prefix +25 +34

Postor-
Postfix 25+ 34+

Q) Calculate the size of the tree : Total No of Nodes

Ass: Given root node, return no of Nodes

```
int Size (Node root) {
1 if (root == NULL) { return 0 }
2 int l = Size (root.left);
3 int r = Size (root.right);
4 return l + r + 1 ;
}
```



Height (Node) = 1 + max (child Node heights)

2) Calculate height of a tree

Ass: Given root Node, get height of the tree

```
int height(Node root){
    if(root == NULL){ return -1; }
    int lh = height(root.left)
    int rh = height(root.right)
    return max(lh, rh) + 1;
}
```