Todays Content

a. Intro to Trees

b. Terminologies

c. Tree Traversals

Recursive Code

1. Assumption = Decide what your function does

2. Main Logic = Solving Problems using SubProblems

   Note: When we call subproblem
   It works according to assumption.

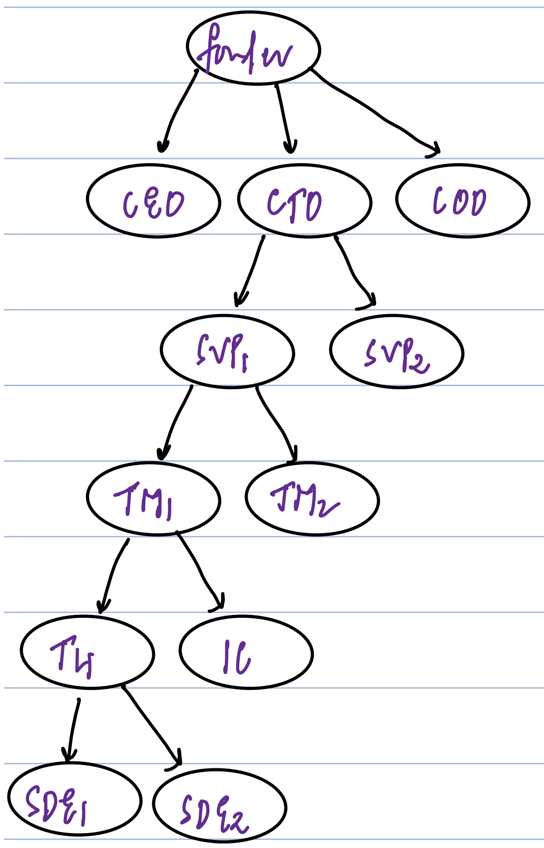
3. Base Condition = Input for which code stops.

Linear Data Structure:

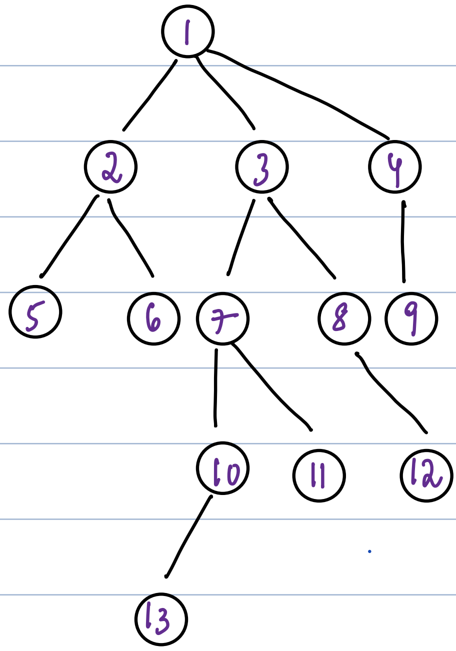
a. Arrays b. Linked List c. Stacks / Queue

Hierarchical Data Structure: Tree

a. Company positions



Level:



Naming Convention:

Node —— Edge

Parent: If Node has child, It's a parent node
Ex: 1 2 3 4 ..

Child: If Node has parent It's a child node
Ex: 2 3 4 ..

Root: Node without any parent.
Ex: 1

Leaf: Node without any child.
Ex: 5 6 9

Height(Node): length of longest path from node to one of it's leaf node.

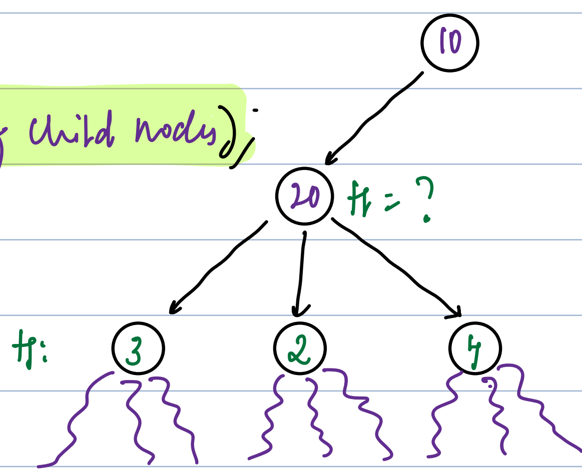
H(3) : 3 Edges, 4 Nodes
H(4) : 1 Edge, 2 Node
H(1) : 4 Edges, 5 Node
H(12) : 0 Edges, 1 Node

#obs

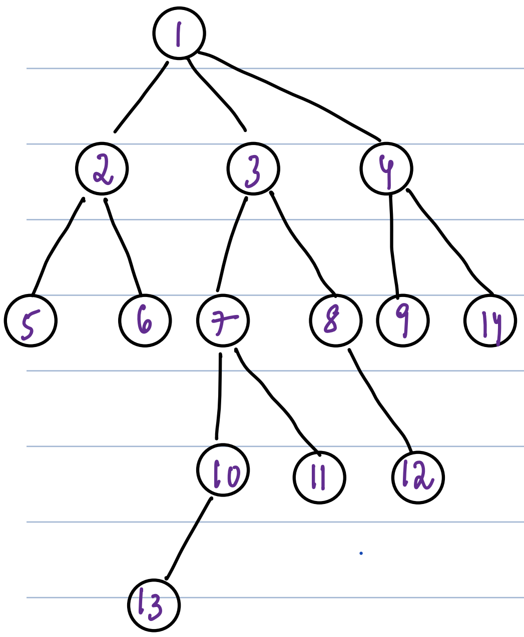
1. H(leaf): 0 Edge, 1 Node

2. H(Node): 1+max(height of child nodes);

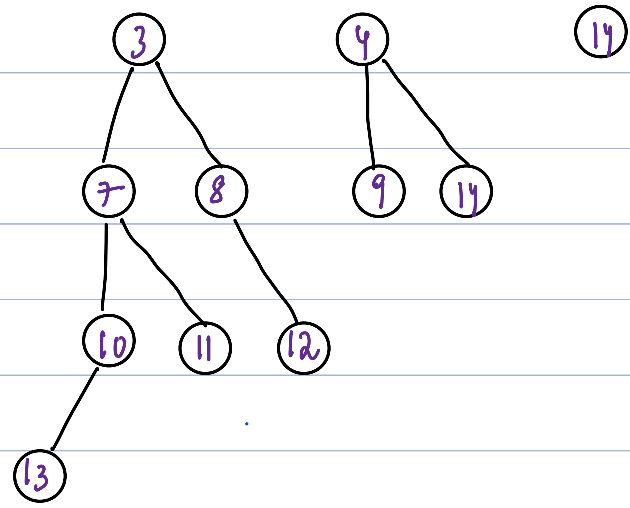
3. H(Tree): H(root)



#Subtree: A subtree is a subset of a Tree structure that starts at a node & contain all it's descendants.



Eg: subtree 3 subtree 4 subtree 14



#obs: No. of subtree is = # No. of nodes

Siblings: Nodes with same parents

Ancestor: All nodes along path from node to root.

Eg: 13 : 13 10 7 3 1

↳ Node can be ancestor to itself in few problems

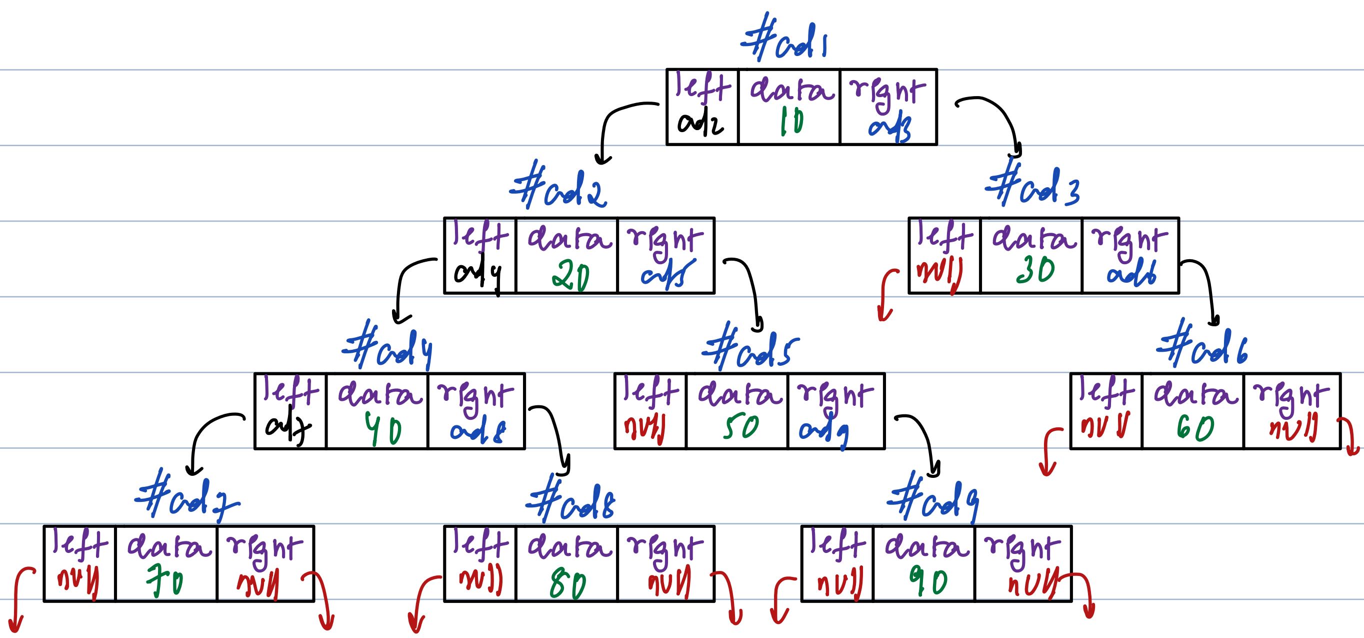
$\boxed{\text{\#Node can have only 1 parent, except root without a parent}}$

Binary Tree: A Tree in which every node can at max have 2 children

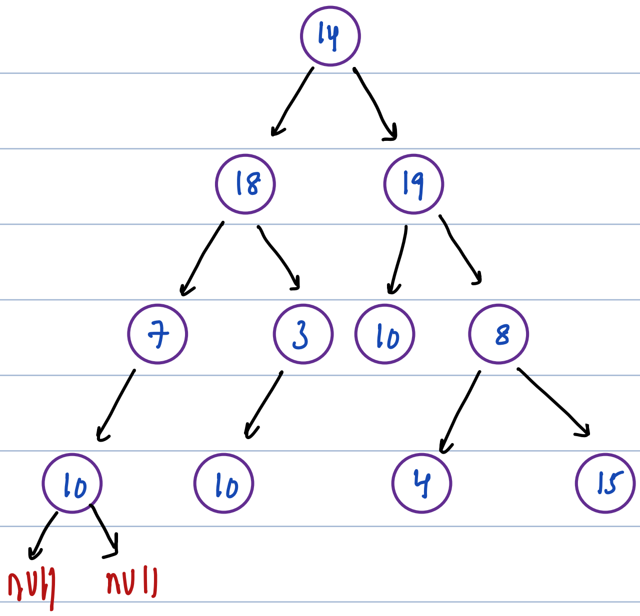
0, 1, 2,

```
class Node{
    int data;
    Node *left, *right;
    Node(int x){
        data = x;
        left = nullptr;
        right = nullptr;
    }
};
```

#obs: Only root node is given to access entire BT



Ex:



1. Root : 14
2. LST of 14
   Root of lst : 18
3. RST of 14
   Root of rst : 19

#Note: LST & RST concept exist for all nodes

#Note: Most of tree problems can be done with recursion.

Ex: sum(BT) = sum(LST) + sum(RST) + root

size(BT) = size(LST) + size(RST) + 1; → root will contribute 1.

Tree Traversal

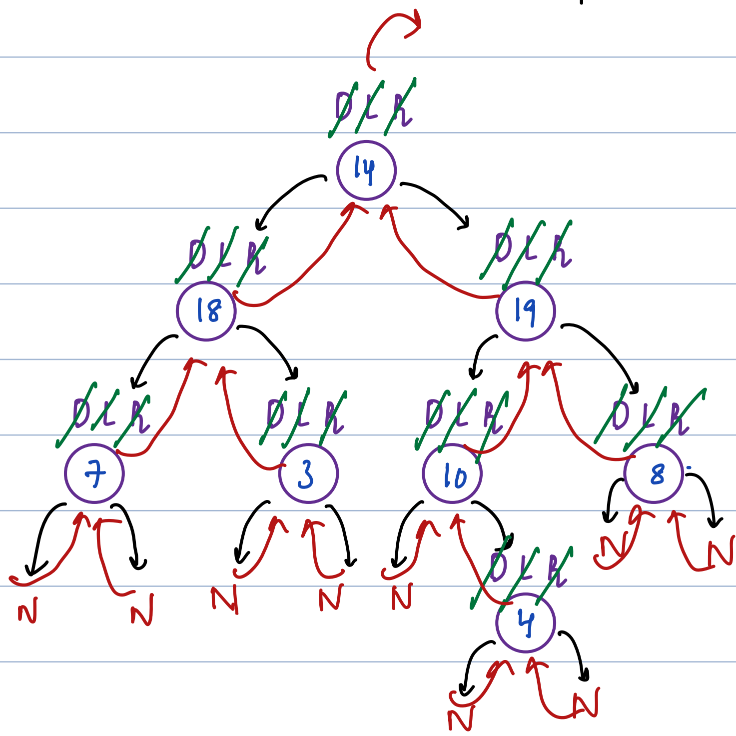
a. PreOrder
b. InOrder
c. PostOrder

PreOrder: DLR Data Left Right

Data: Print data

Left: Goto leftsubtree & print entire leftsubtree in preOrder

Right: Goto rightsubtree & print entire rightsubtree in preOrder



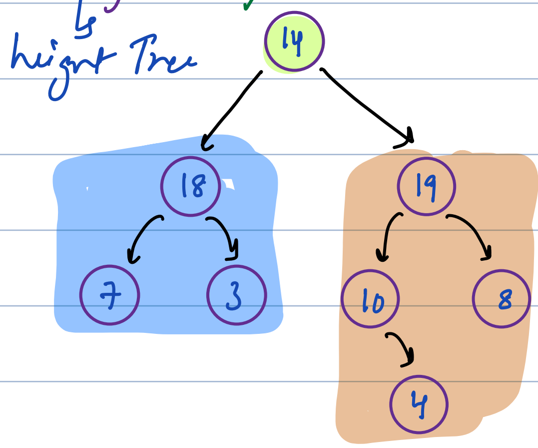
PreTree: 14 | 18 7 3 | 19 10 4 8

root | preLST | preRST

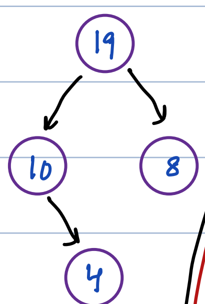
Ass: Given a root node of BT, Print entire BT in PreOrder & return nothing.

```
void preOrder(Node root){ TC:O(N) SC:O(H)
    if(root==null){return;}
    print(root->data);
    preOrder(root->left); #root of LST
    preOrder(root->right); #root of RST
}
```

$O(H)$ → height Tree



Ex: Tree



p(null): return : 3

p(null): return : 3

p(8) : ~~D~~ ~~L~~:null ~~R~~:null : 2

p(null): return : 4

p(null): return : 4

p(4) : ~~D~~ ~~L~~:null ~~R~~ : 3

p(null): return : 3

p(10) : ~~D~~ ~~L~~:null ~~R~~ : 2

p(19) : ~~D~~ ~~L~~:10 ~~R~~ : 1

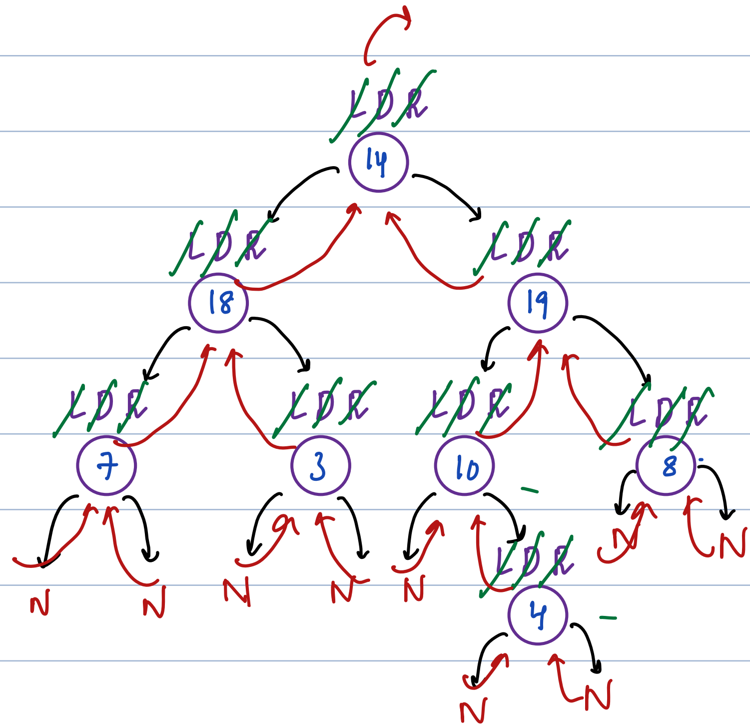
main(): pre(19)

Max Stack Size ≈ Height of Tree

Output:

19 10 4 8

<u>In Order</u> LDR Left Data Right



Left: Goto leftsubtree & print entire leftsubtree in InOrder

Data: Print data

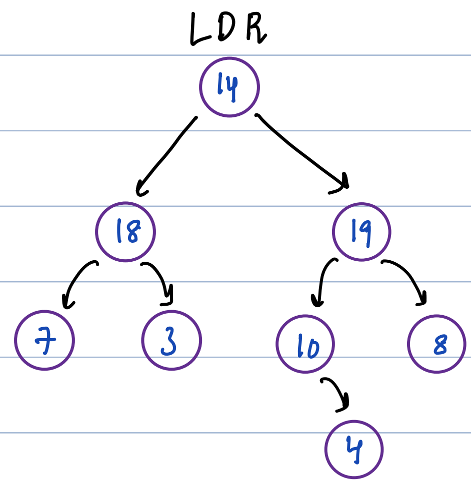
Right: Goto rightsubtree & print entire rightsubtree in InOrder

InTree: 7 18 3 14 10 4 19 8

InLST root InRST

Ass: Given a root node of BT, Print entire BT in Inorder

```
void InOrder(Node root){
    if(root==nullptr){return;}
    InOrder(root.left);
    print(root.data);
    InOrder(root.right);
}
```

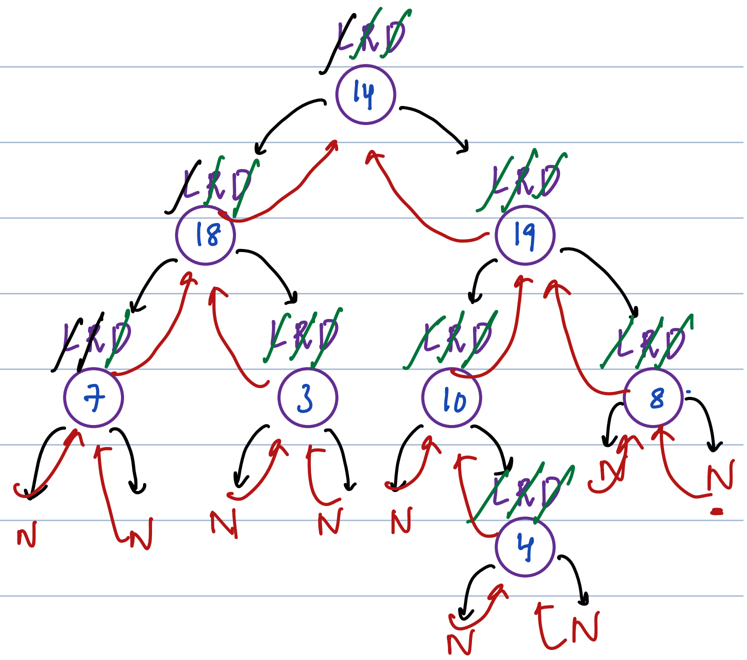


PostOrder   LRD Left Right Data

Left: Goto leftsubtree & print entire leftsubtree in PostOrder

Right: Goto rightsubtree & print entire rightsubtree in PostOrder

Data: Print data



PostTree: 7 3 18   4 10 8 19   14

PostLST   PostRST   root

Ass: Given a root node of BT, Print entire BT in PostOrder

```
void PostOrder(Node root){
    if(root==nullptr){return;}
    PostOrder(root.left);
    PostOrder(root.right);
    print(root.data);
}
```