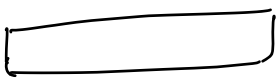
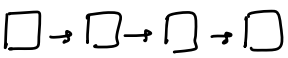
Adv - March 9th

→ Arrays



Linked List

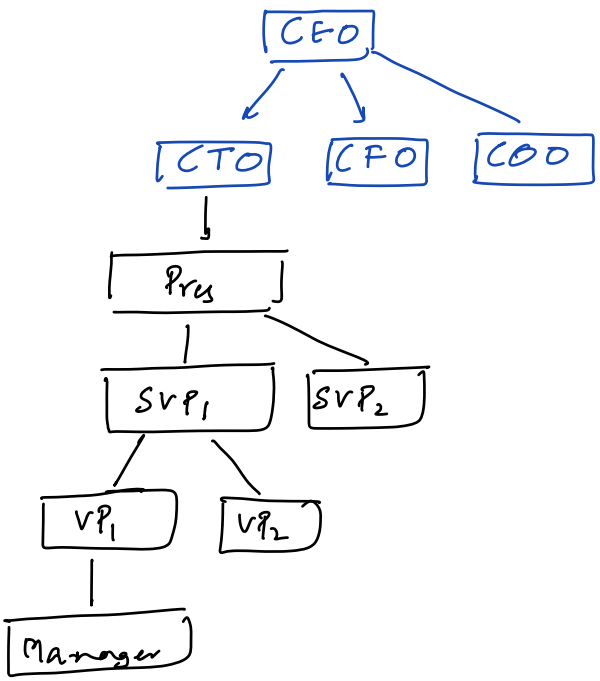


Stacks

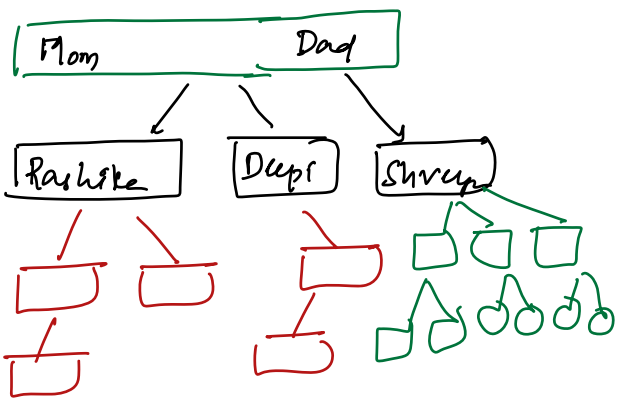
Queue

Hierarchical Data

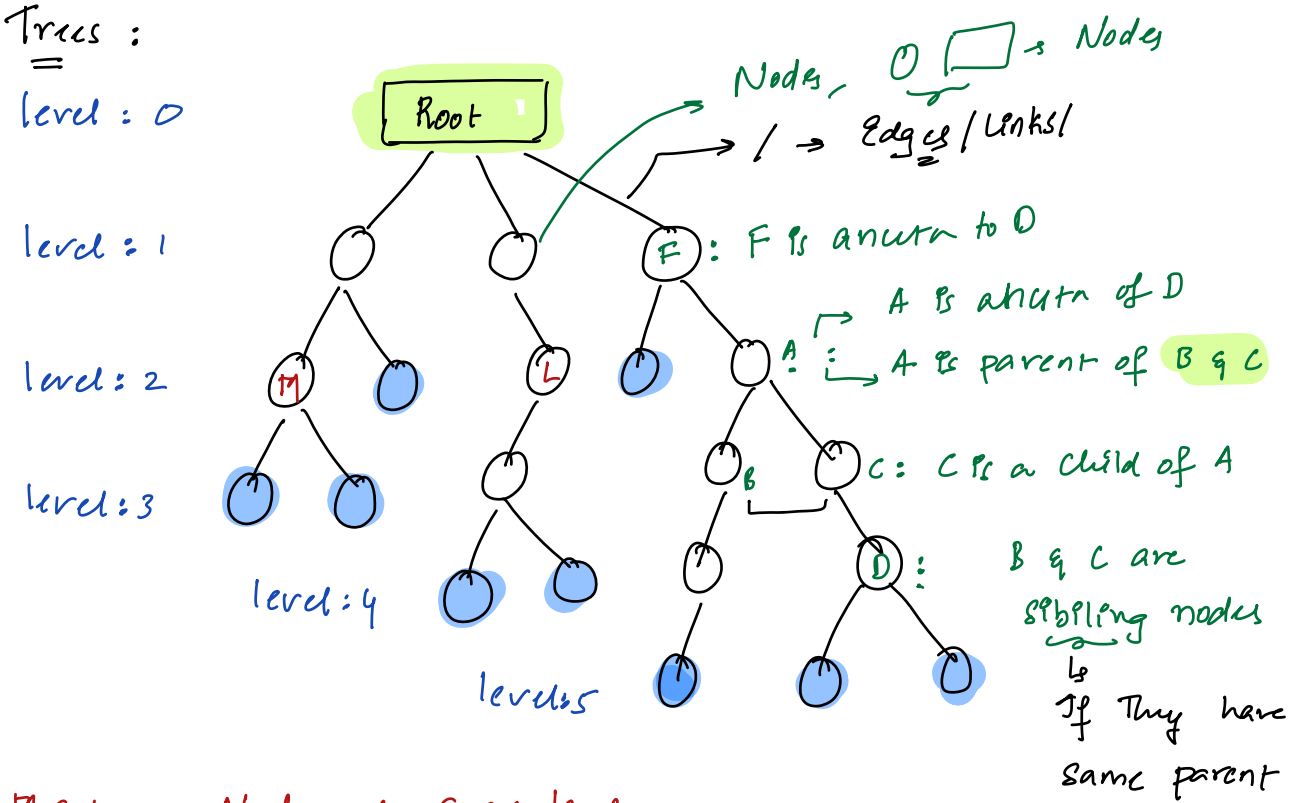
Eg: Company organisation



Eg: Family Tree



Trees :



M & L are Nodes at same level.

○ → leaf Nodes

No children

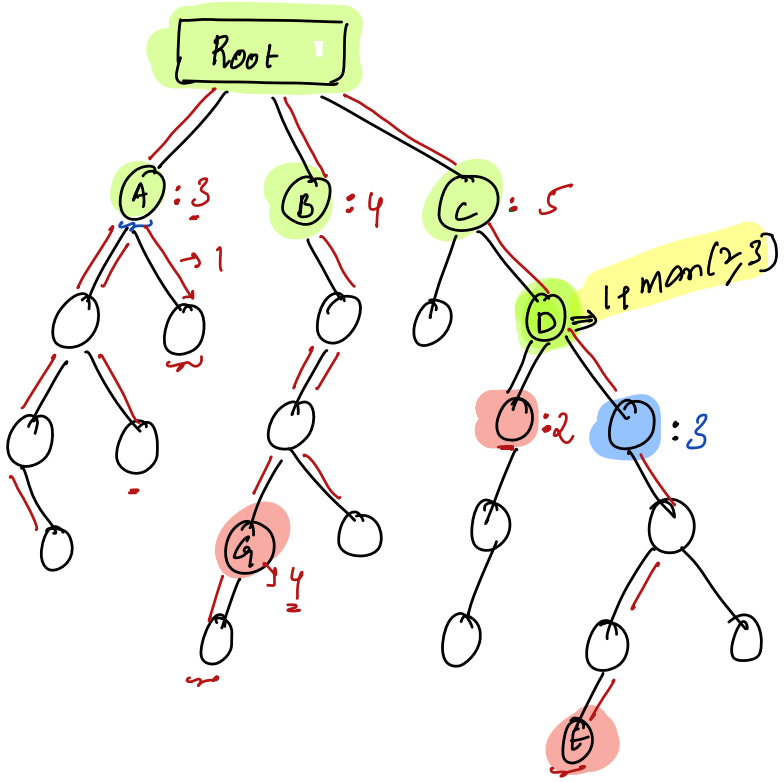
Naming Conventions

- → Parent / Child
- → Ancestor / Descendent
- → Sibling Nodes / Nodes at same level
- → leaf Nodes
- → (Root Node)

Height (Node):

length of longest path from Node to any of its descendant leaf Nodes

Path: No: of Edges



Height(A): 3

Height(B): 4

Height(C): 5

Height(root) = 1 + max(Height of child)

= 1 + 5 = 6

Height(Node) = 1 + max of (Height of all child Nodes)

Note: Height(LeafNode) = 0

Depth(Node) = length of path from Root Node → Given Node

Depth(G) = 4

Depth(E) = 6

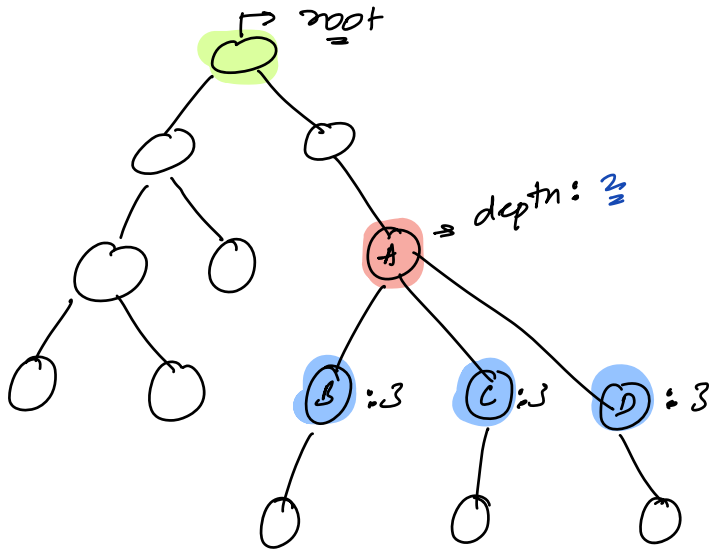
Fn:

obs: Node

depth (Node) = k

depth of all Pts

child Nodes = (1 to k)



Q)

Height of Tree = Height (Root) = max depth (Leaf Node)

Note: Depth (Root) = 0

obs: → (Take a tree / try them our working or that)

Binary Tree

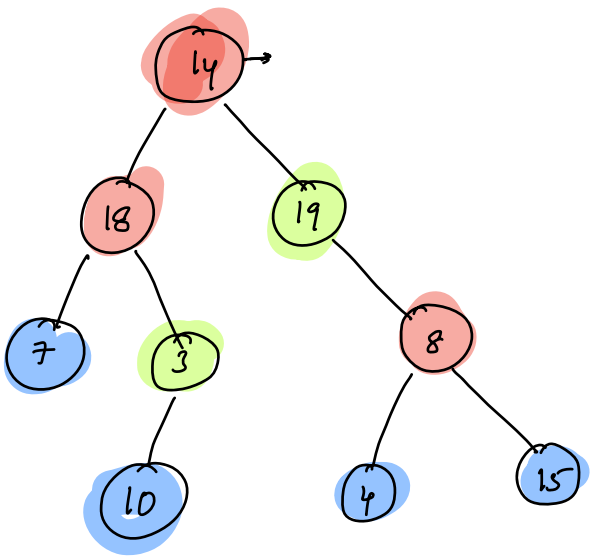
Every Node can at max have 2 child Nodes

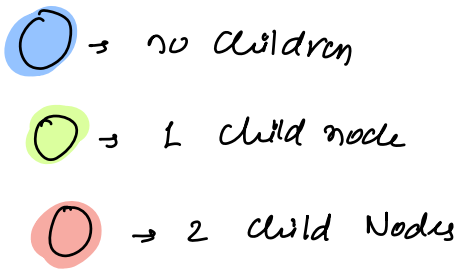
N - Array Tree → {Adv}

At max every node can have N child Nodes

$x = N \{0, 1, 2 \cdots N\}$

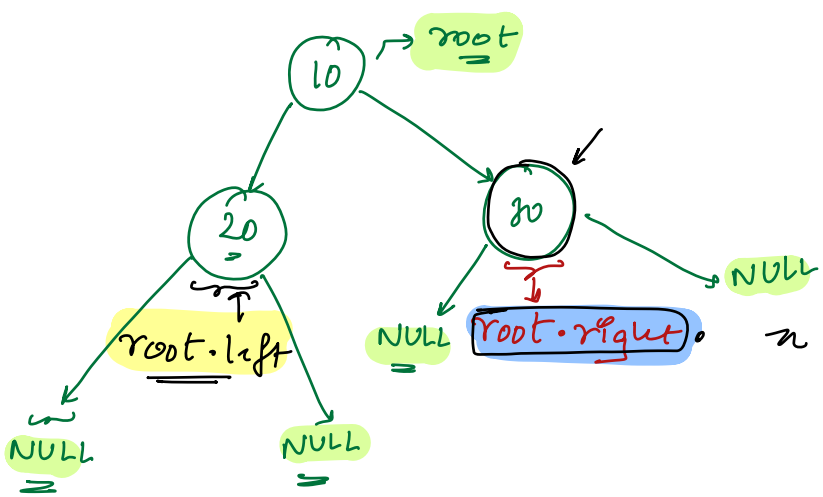
Eg:





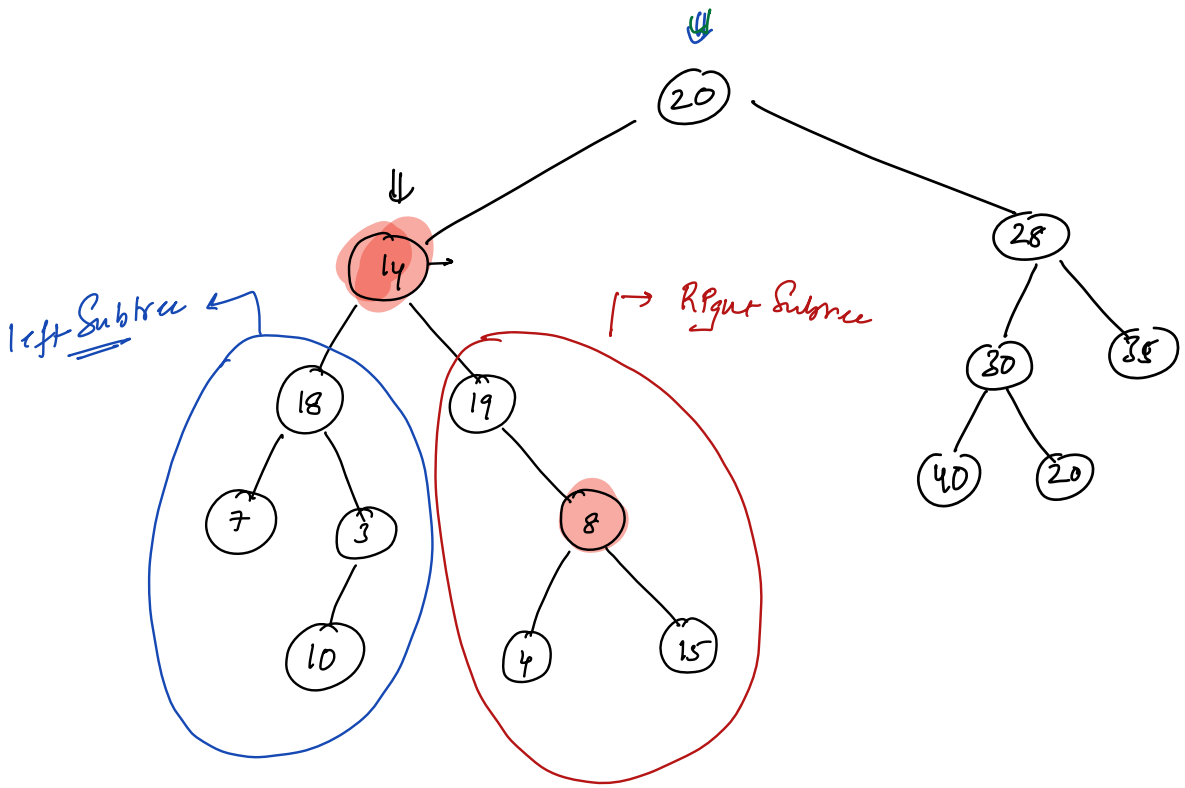

```
class Node {
    int data;
    Node left
    Node right
    Node(int n) {
        data = n
        left = NULL
        right = NULL
    }
}
```

```
Node root = new Node(10);
root.left = new Node(20);
root.right = new Node(30);
```



// Problems:

- You are given BT, { we will do problems }
- We cannot change Node structure, (Node structure fixed)



// Because of above subtree property → Recursive plays Huge

Recursion:

Ass: { Decide, what your function should do, assume it does }

Main logic: { Solving problems with subproblems }

Base Condition: { When code stops }

→ (C#)

↳ TreeMap / TreeSet in C#

→ TreeMap / TreeSet in Java

→ Map / Set in C++

TreeMap, TreeSet are built by BBST Trees

Balanced Binary Search Tree ?

10:30 PM

↳ 9 Marks

Start: Ramesh has 2 children

Ramesh has 2 children

Ramesh has 2 children

Ramesh has 2 children

Content..

↳ Moving advanced Batch

Poor's women Joke

→ Paradhi :- godes of PG's : 8am - 10:30pm

→ Yahnit :

M/W/F // @ support

11

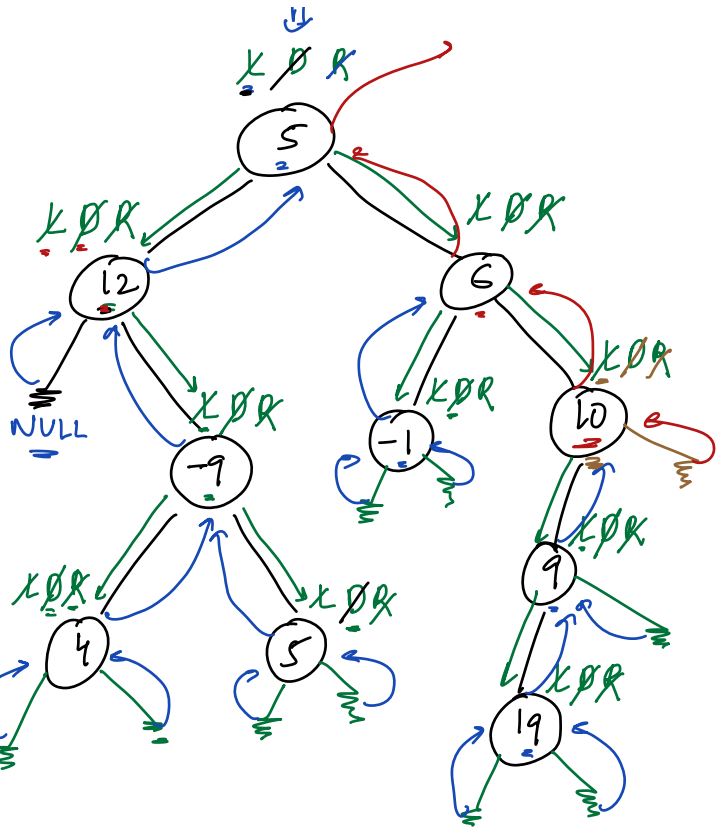
Problems:

Tree Traversals

1) pre order : [D L R] → {TODO}

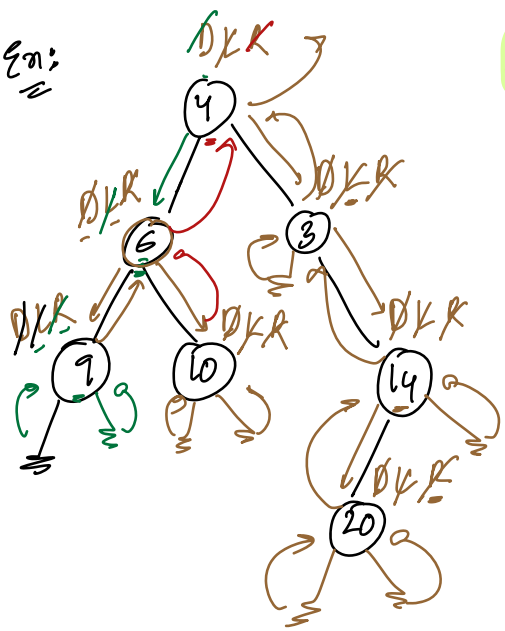
2) In order: [L D R] → {

3) post order : [L R D] → {TODO}



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

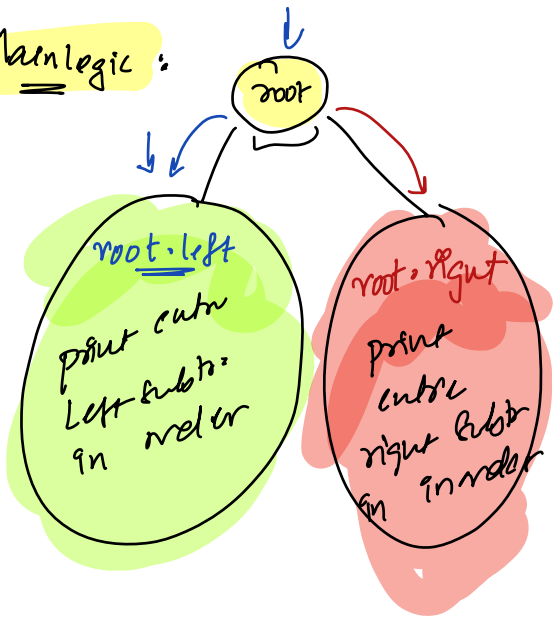
Postorder: 9 10 6 20 14 3 4

// Inorder traversal - (Recursive)

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

Ass: Given root node, it will print entire tree in Inorder.

Main logic:



print entire left subtree in inorder

print root.data

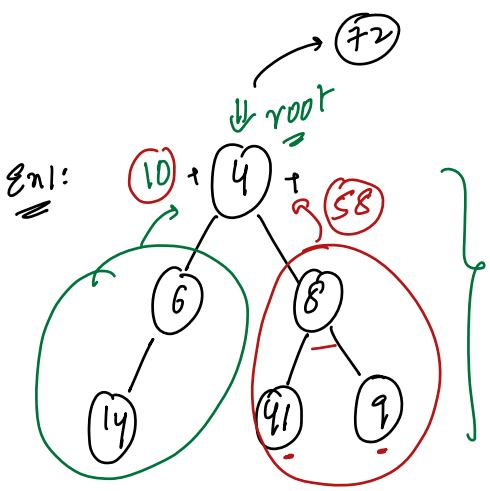
print entire right subtree in inorder

Inorder : 1 2 3 4

preorder : 1 3 2 4

postorder : 1 2 4 3

Base Condition: if root == NULL return



size = 6

- size(): given Node, it will return size
- sum(): given Node, it will return sum of all Nodes

probl :

int size (Node root) {

(Nodes)

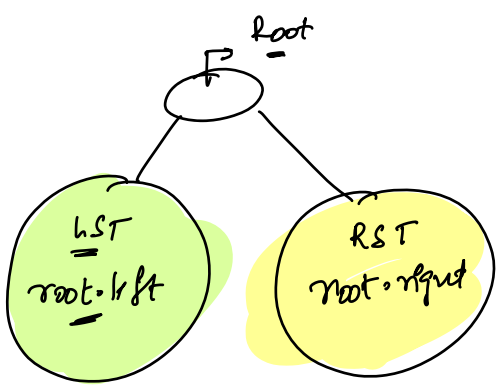
```
if (root == NULL) {return 0}
l = size(root.left)
r = size(root.right)
return l + r + 1
}
```

int sum (Node root) {

sum of all Nodes

```
if (root == NULL) {return 0}
l = sum(root.left)
r = sum(root.right)
return [l + r + root.data]
}
```

LST RST



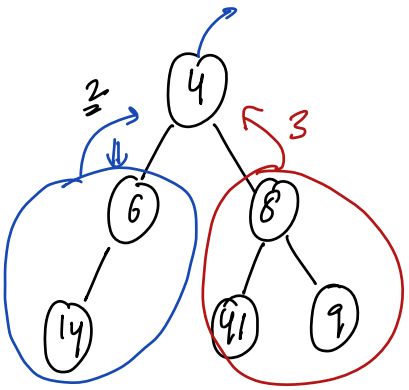
Height: Given Node, it will return height of tree

```
height (Node root) {
if (root == NULL) {return 0}
hl = height(root.left)
hr = height(root.right)
return (max(hl, hr) + 1)
}
```

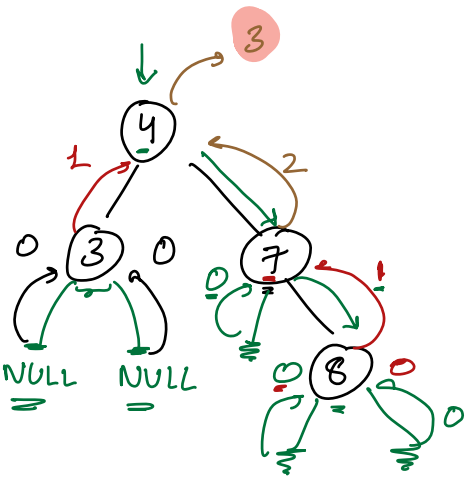
Height(Node) = 1 + max of (Height of all child Nodes)

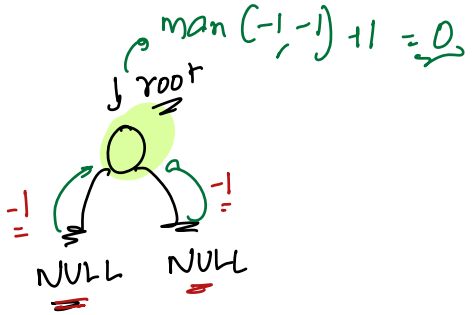
Trace

Trace




```
height (Node root) {
    if (root == NULL) { return -1 }
    hl = height (root.left)
    hr = height (root.right)
    return (max (hl, hr) + 1)
}
```





In Assignment height is different to what we discussed

→ height : No: of Edges

→ height : No: of Nodes

→ Tree Traversals : (Iterative)

→ Arrange submit code recursive manner

Doubts

```
int i = 0;

void inorder(Node root, ar[])
{
  ① if(root == NULL){return}
  ② inorder(root.left, ar)
  ③ {ar[i] = root.data, i = i+1}
  ④ inorder(root.right, ar)
}
```

⇒

```
solve() {
  int n = size(Root)
  int ar[] = new int[N]
  inorder(Root, ar[])
  return ar[]
}
```

Oops → Class Object Object Ref } Oops :

Recursion →

⇒ Tracing Recursion

(Nodes)

```
Int size(Node root){
    if(root == NULL){return 0}
    l = size(root.left)
    r = size(root.right)
    return l+r+1
}
```