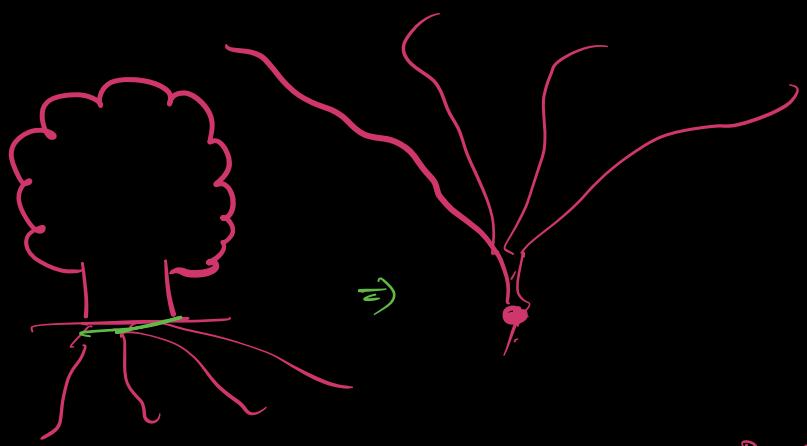
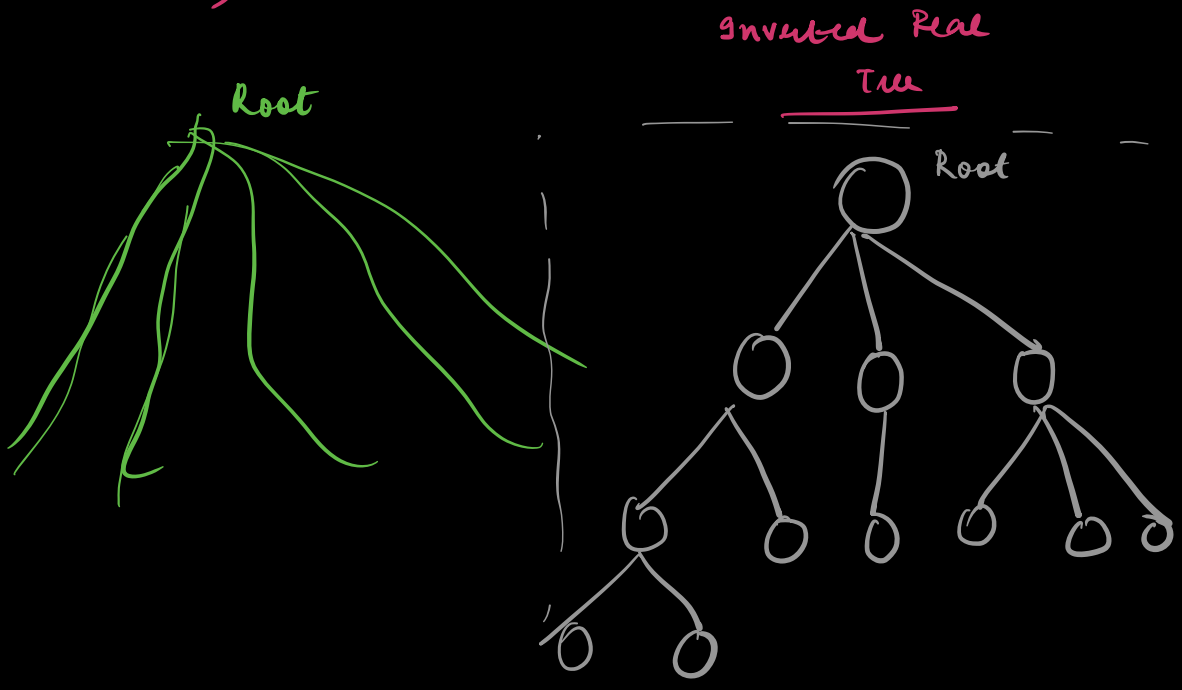
TREES BASICS

- Terminologies
- Traversal
- Couple of problems.
  - Size of Tree
  - Height of tree



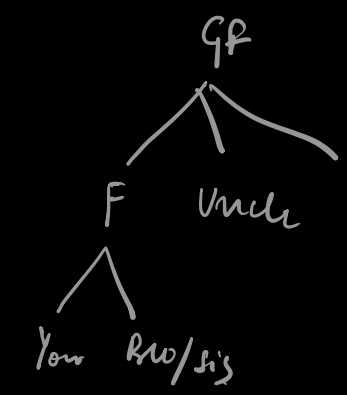


All the DS till now covered

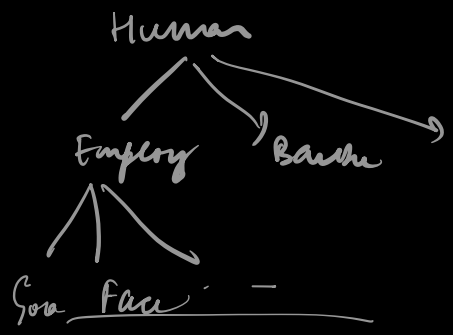
↳ Linear structure (Array, LL, stack, Queue)

Hierarchical structure ↳ spl about trees.

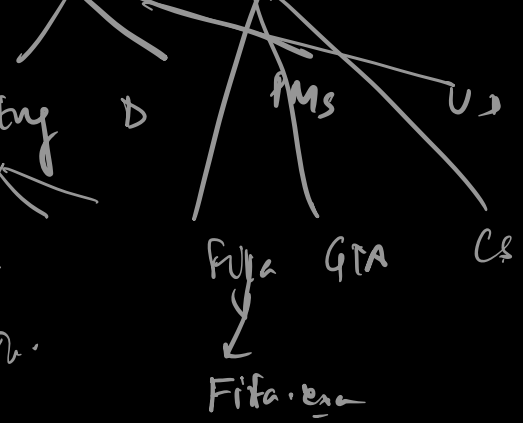
Family tree



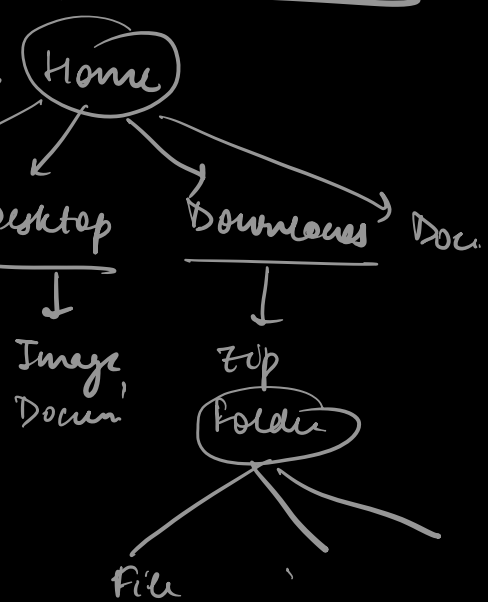
Inheritance



Organisation



File system in OS



tree

HTML DOM

USE CASE

SQL

↳ Indexing

B+ Trees

HTML

↳ Head

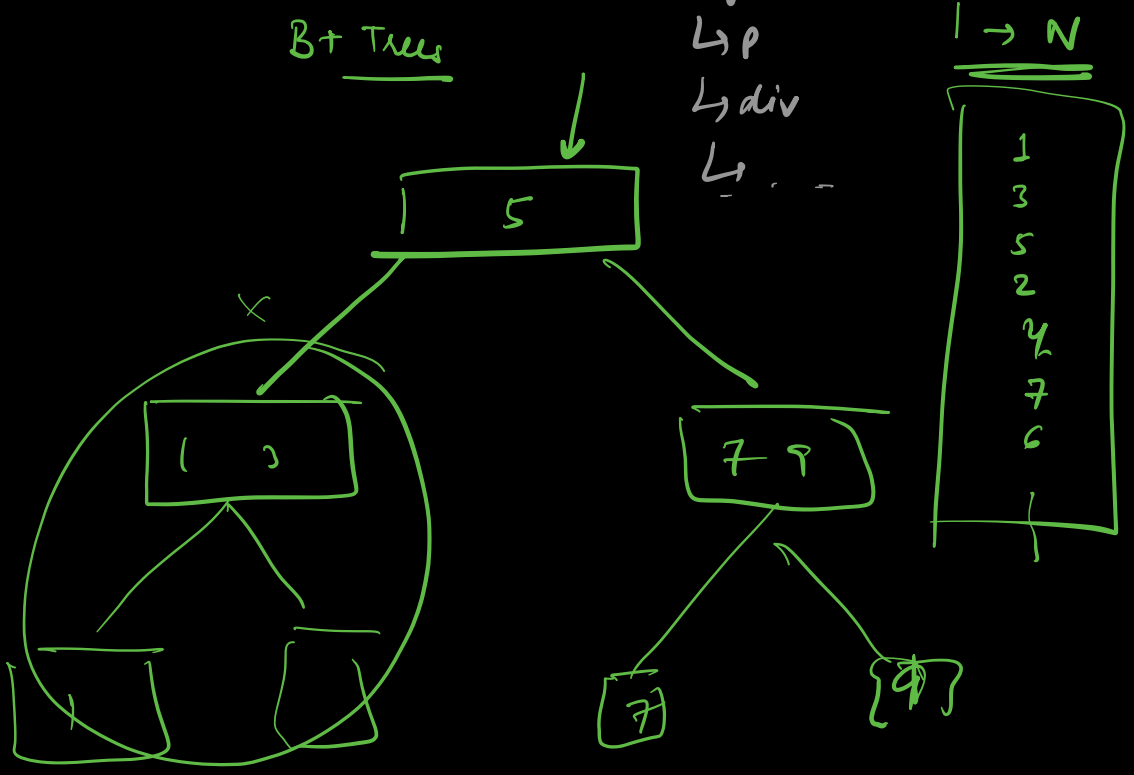
↳ Title

↳ Body

↳ p

↳ div

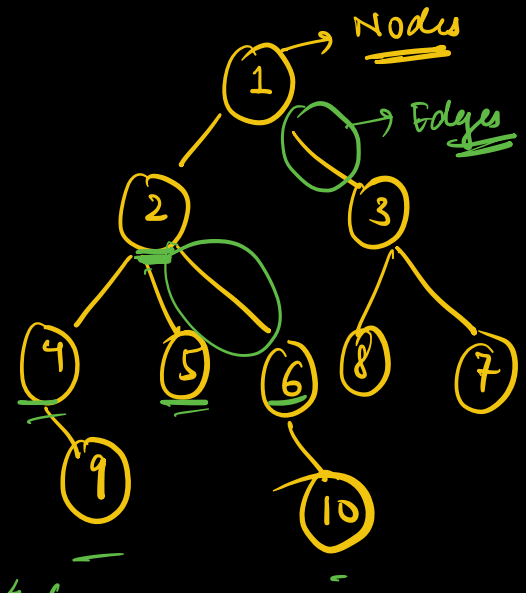
↳ ...



TRIE (Trees)

↳ Auto complete feature.

Terminologies in Trees



Parent of
(2) ⇒ (3)

Sibling of
(8) ⇒ (7)

Siblings of
(5) ⇒ 4, 6

⇒ Root Node
↳ (1)

⇒ children of
(2) ⇒ 4, 5, 6

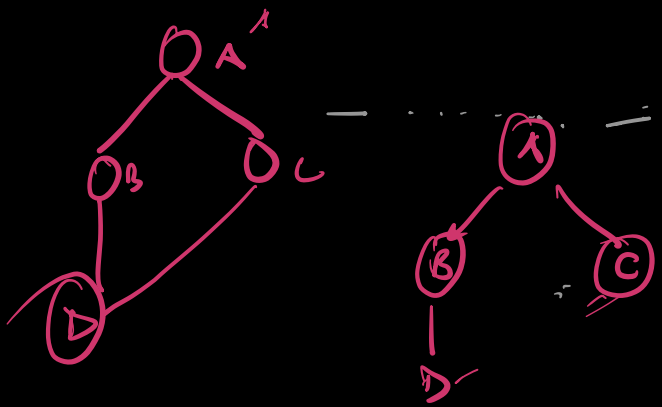
⇒ Successors of
(2) ⇒ 4, 5, 6, 9, 10

⇒ Leaf Nodes in Tree
9, 5, 10, 8, 7

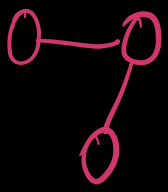
(*) A child can have a single parent.

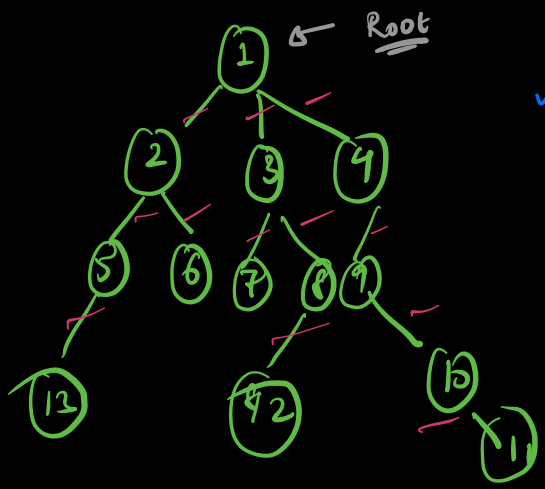
(*) A tree won't have any cycles.

What is a cycle.



If you start from a node and come back to the same node without repeating any edges that is called a cycle





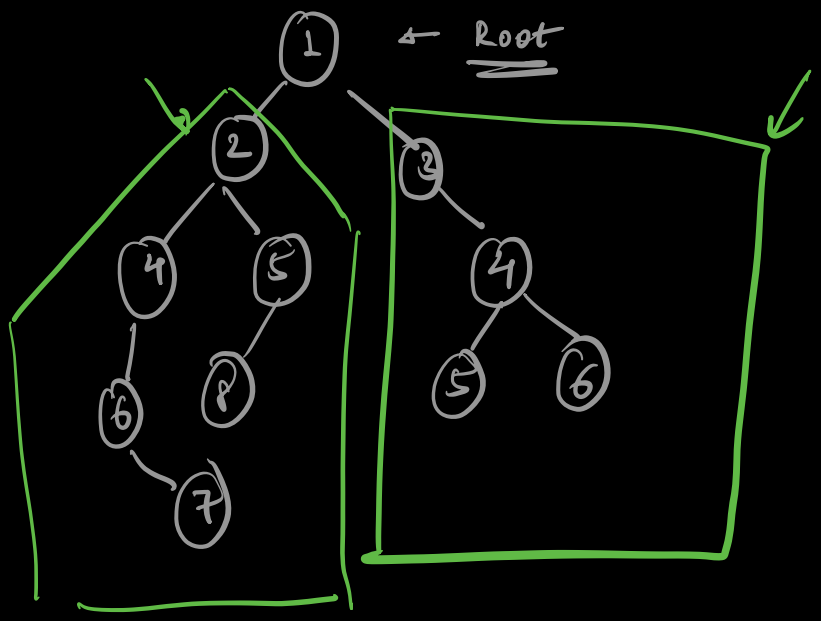
count of nodes = 13

count the no of edges??

12

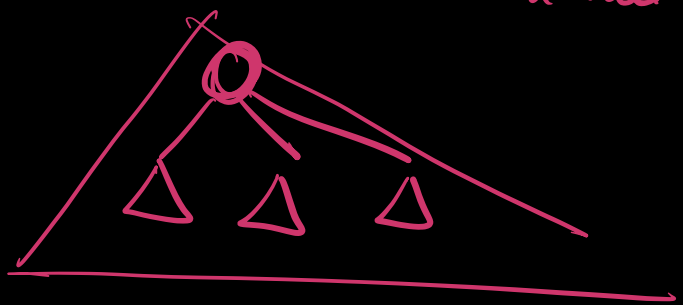
For a tree, n nodes → n-1 edges

Since every node (except the root node) has a parent; there is an edge connecting node to the parent, and hence n-1 edges.



subtrees of the tree

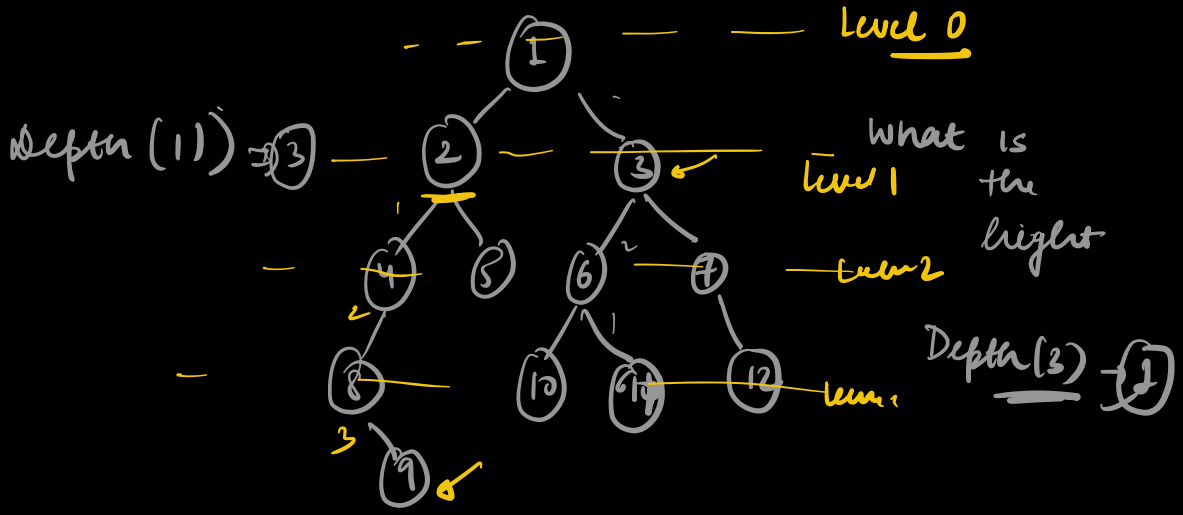
For a root node, every child can act as the root node of its own tree



Can we apply recursion to solve problems on trees

Height of tree

Distance of farthest leaf node from the root node



Depth of any node ⇒ distance from root

Height of any node ⇒ distance from farthest leaf.

Height (3) = 2

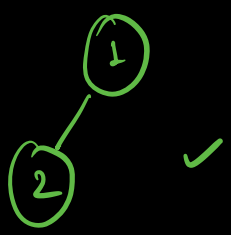
Height (2) = 3

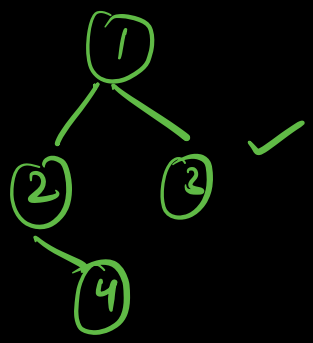
⇒ length of longest path to the leaf node from any

Height of tree = ? 4

Binary Tree

↳ A tree in which every node can have a maximum of 2 children. → 0, 1, 2

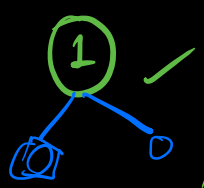


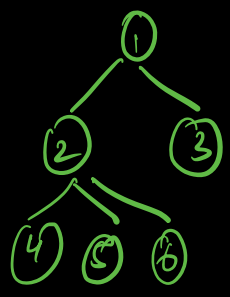


Every node has at most 2 children

↳ left child

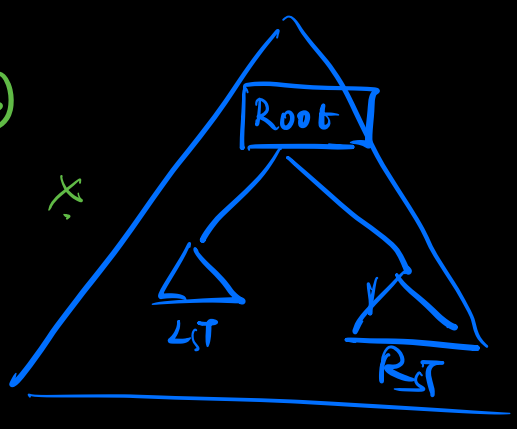
↳ right child.





↳ left subtree

↳ right subtree.



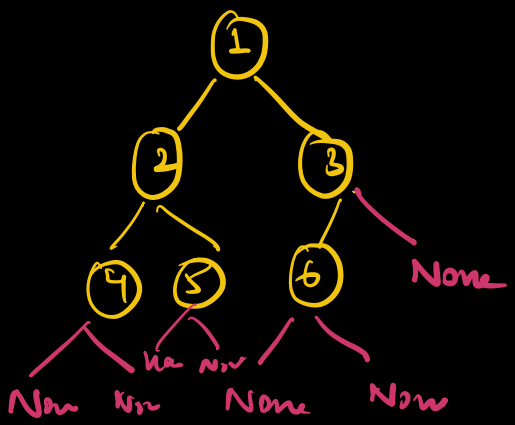
Ternary (3 children at most)

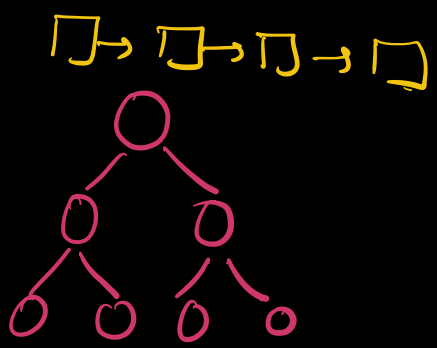
K - any tree.

A node can have at max K children

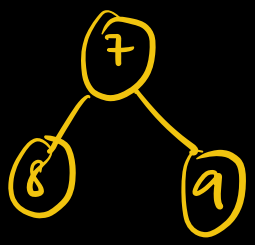
max

Implementation of binary Trees



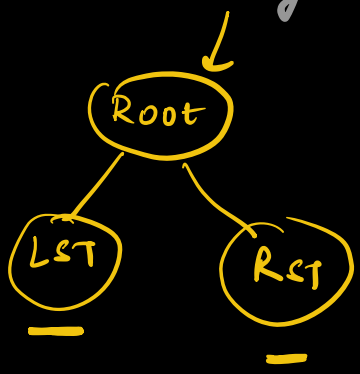


```
class TreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
```




```
root = TreeNode(7)
root.left = TreeNode(8)
root.right = TreeNode(9)
```

(1) count the no. of nodes in a binary tree.



Assumption:

Returns the no. of Node in the tree rooted at root

count(None)

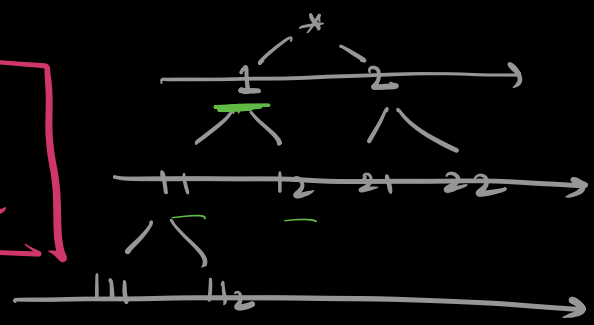
count(2)

Recursive technique

count of Nodes in the

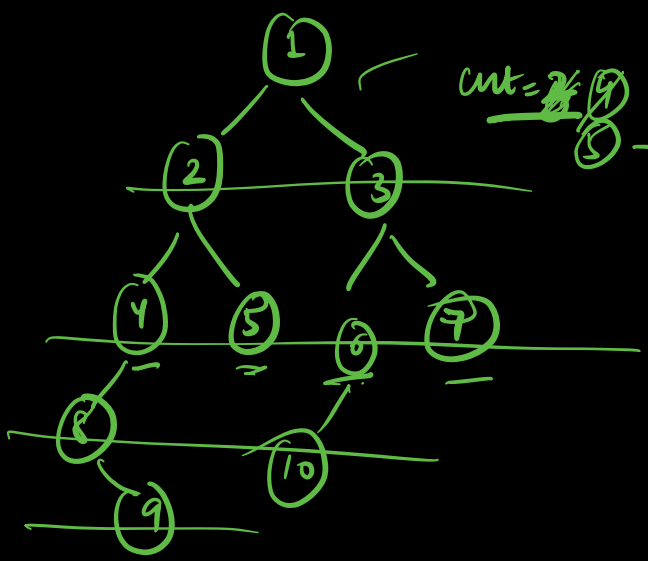
```
def count(root):
    if root == None:
        return 0
    return 1 +
        count(root.left)
        + count(
            root.right)
```

Level Order Traversal

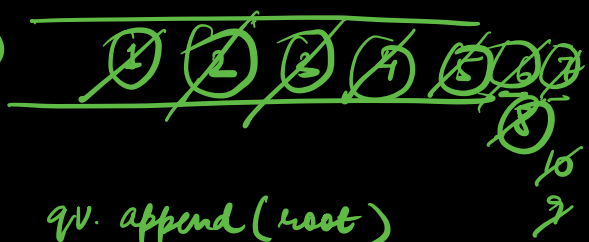


⇐ Tree DS

1, 2, 11, 12, 21, 22



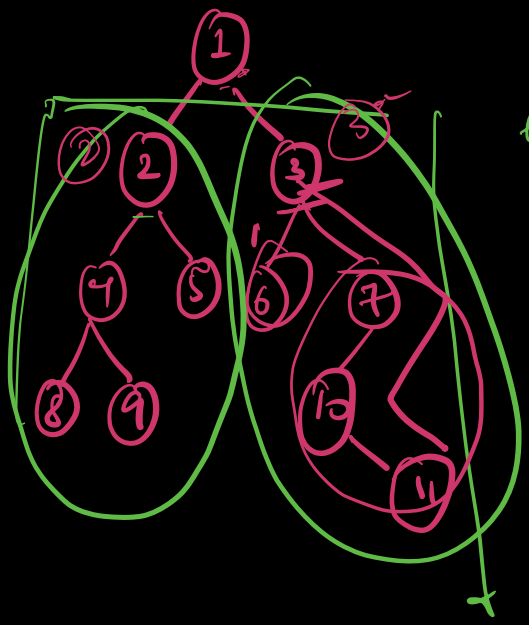
cnt = 0



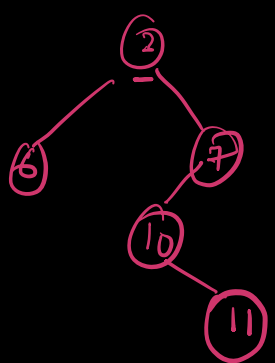

```
qu.append(root)

x = qu.popleft()
cnt += 1
if x.left:
    qu.append(x.left)
if x.right:
    qu.append(x.right)
```

Q. Calculate height of the tree



$$= 1 + \max(\text{height}(\text{root.left}), \text{height}(\text{root.right}))$$

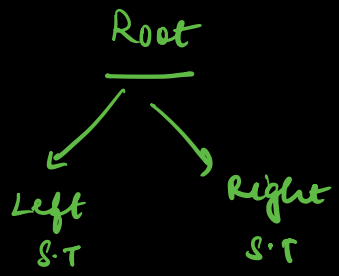



```
def height(root):
    if root == None
        return -1
    return 1 + max(height(root.left), height(root.right))
```

if root == Null
return 0

Traversals

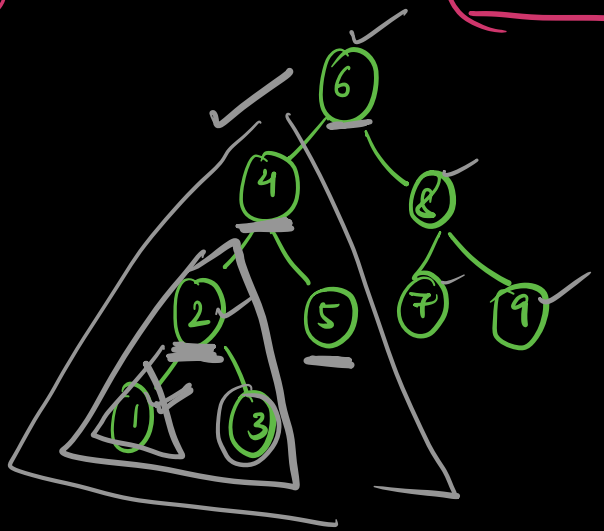
→ Recursive Traversals



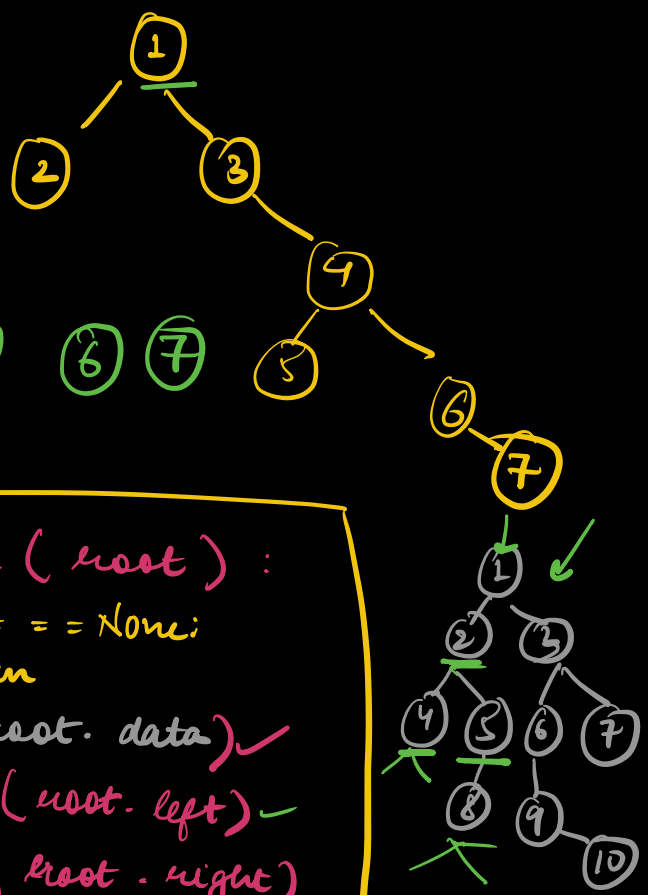
1. Left Part | Root | Right Part
2. Left | Right | Root
3. Root | left | right
4. Right Left Root
5. Right Root left
6. Root Right left

Left should be visited before right

(1) Pre Order (Root Left Right)



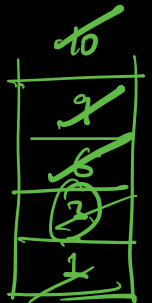
6 4 2 1 3 5 8 7 9



1 2 3 4 5 6 7

```
def preorder(root):
    if root == None:
        return
    print(root.data)
    preorder(root.left)
    preorder(root.right)
```


1, 2, 4, 5, 8, 3, 6, 9, 7



N is no. of nodes

H is height of

TC → O(N)

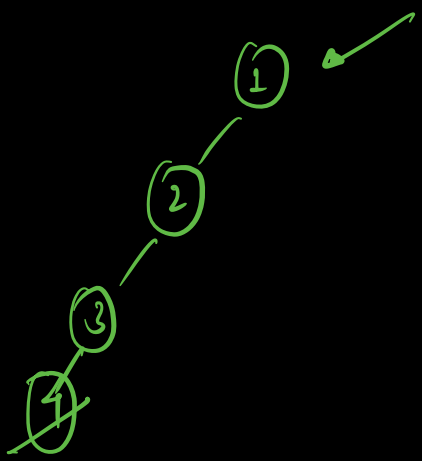
SC =

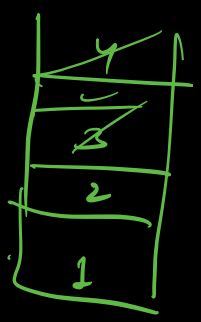
Hmm

$$T(N) = T(L) + T(R) + O(1)$$

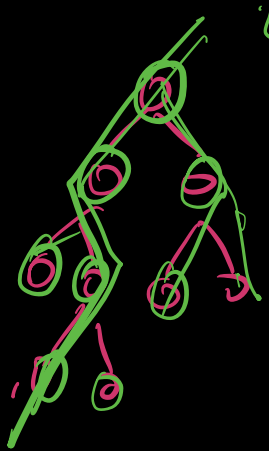
$$T(N) = 2\,T(N/2) + O(1)$$

$$= O(N)$$

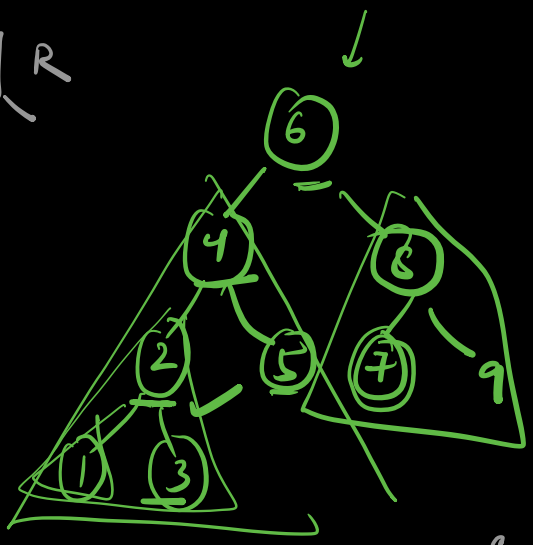




Height



Inorder Traversal



Binary Search Tree

Inorder Traversal

1 2 3 4 5 6 7 8 9

Sorted

Inorder Traversal of a BST is sorted!

if root = Null

```
inorder
inorder (root.left)
print (root.data)
inorder (root.right)
```

Postorder

L R D

root, val, 0

```
def depth ( root, value, cur)
    if ( root.val = value )
        return cur
    depth ( root.left  cur+1 )
    depth ( root.right cur+1)
```