[Advanced]

26th April → Tuesday

first 3 class of Adv ⇒ Yahnvi

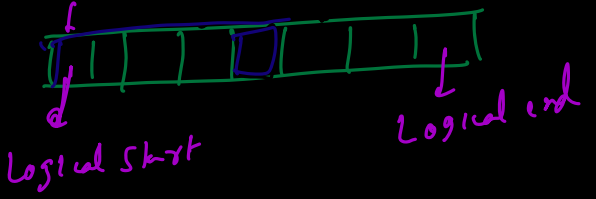
Later ⇒ Ganesh

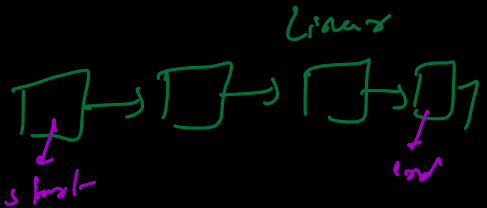
Basics of Trees -1

→ Recursion & structure of Tree

DS — 99%

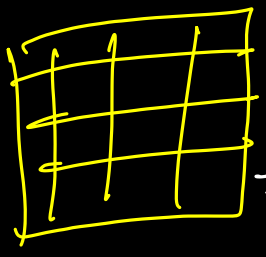
Linear DS — Arrays, Linked Lists, Stacks, queue



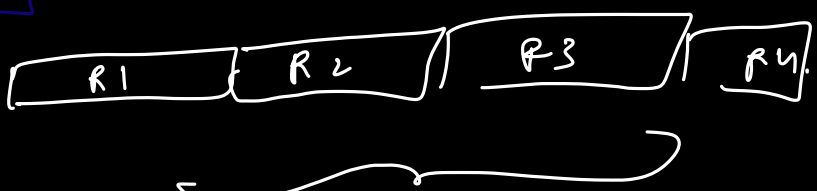


Some sort of Hierarchy

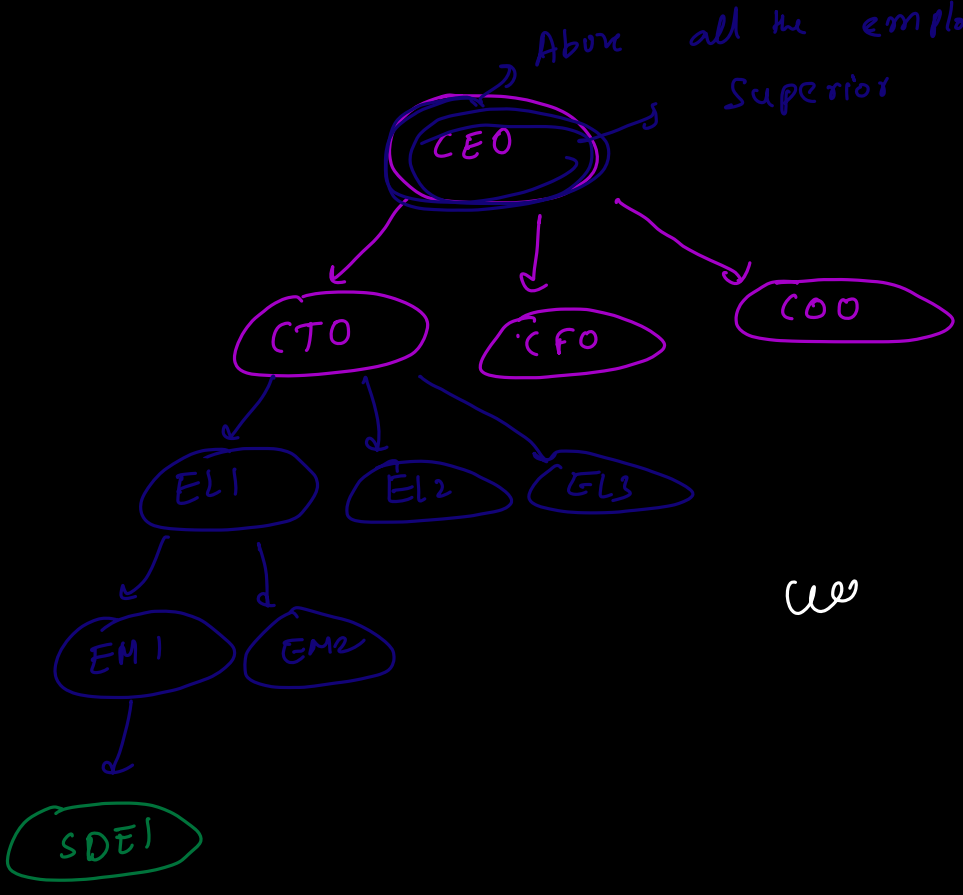
Trees



⇒ Linear DS

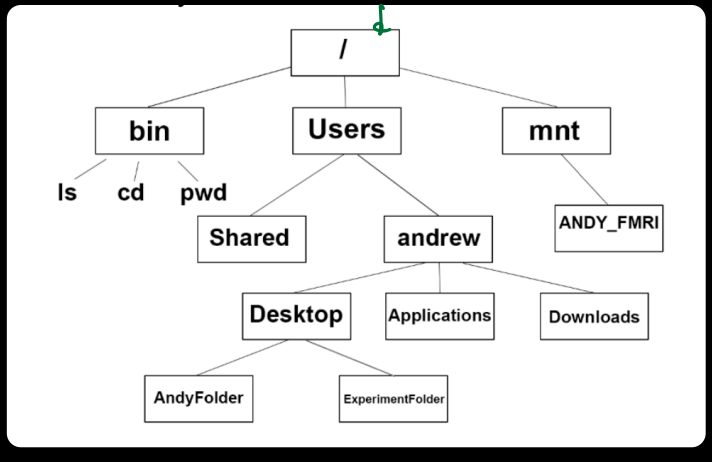


T



→ Family Tree

→ File system



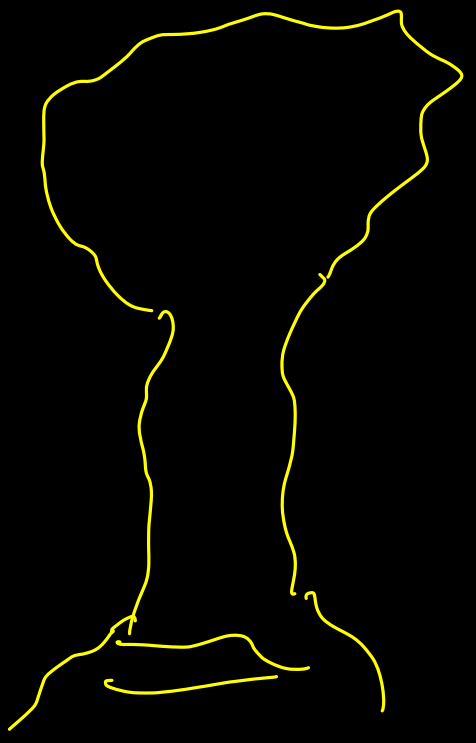
Inverted

Non-linear DS : Trees, Graphs

→ Segment tree

→ Trie

→ B.S.S.T



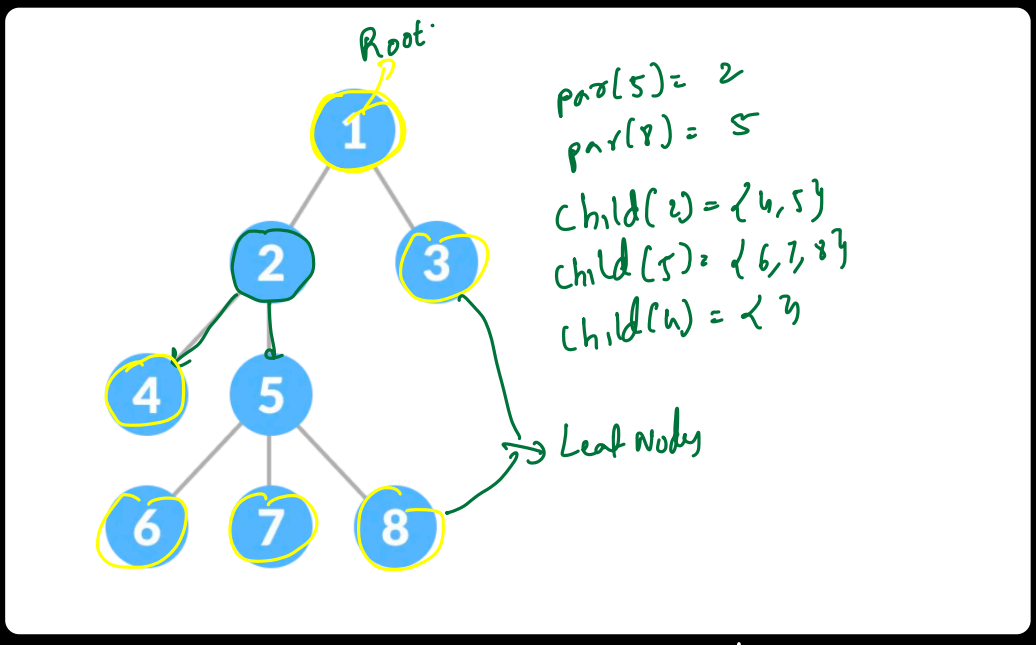
→ Inverted tree with root at the top & leaves at the bottom

Applications of TREES

1) Represent hierarchical data
2) Organizing data so that search, insertion, deletion can done efficiently ⇒ Binary search Tree
   O(log n)
3) Trie Data Structure: Used for auto complete features

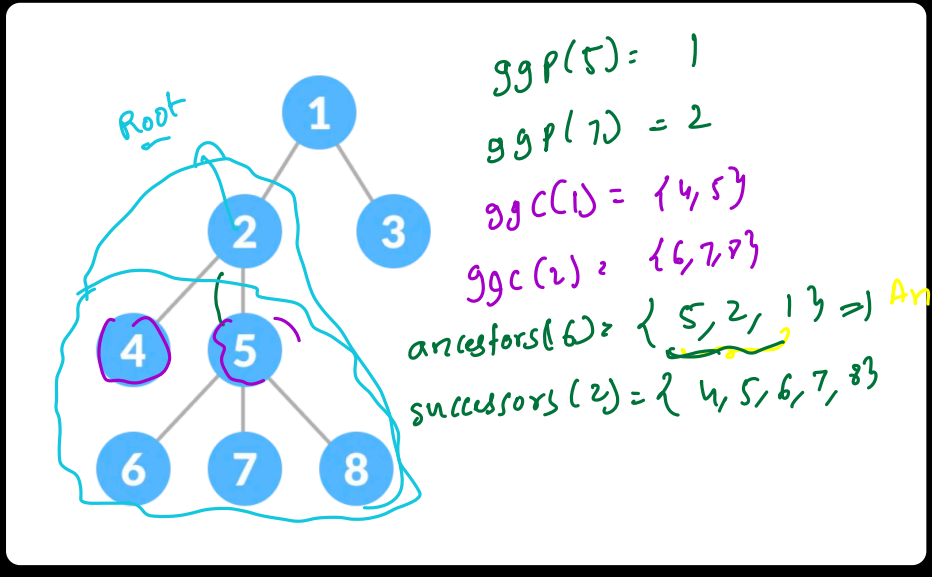
Trie is a tree

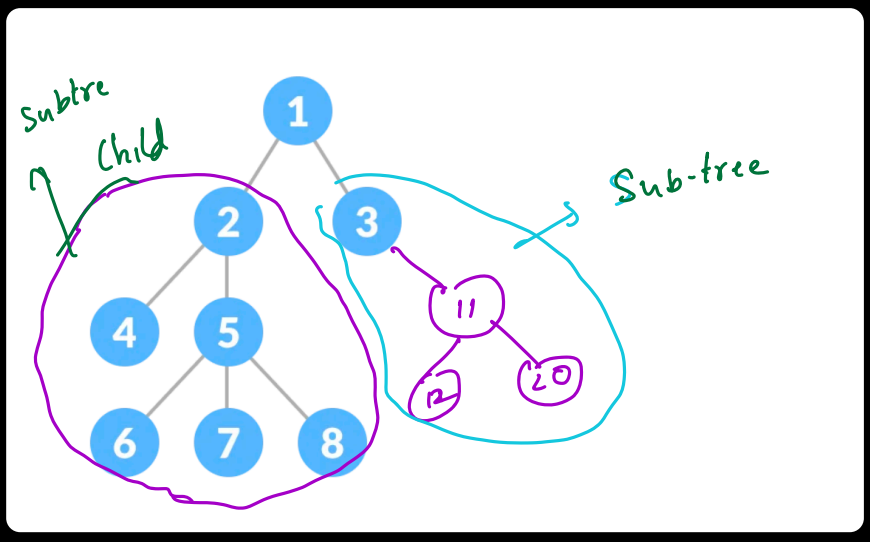
cha ?



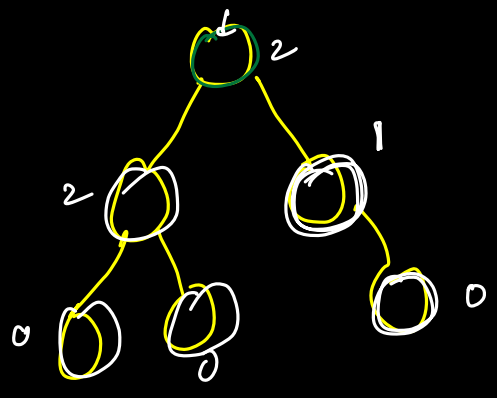
Root: the only node without a parent

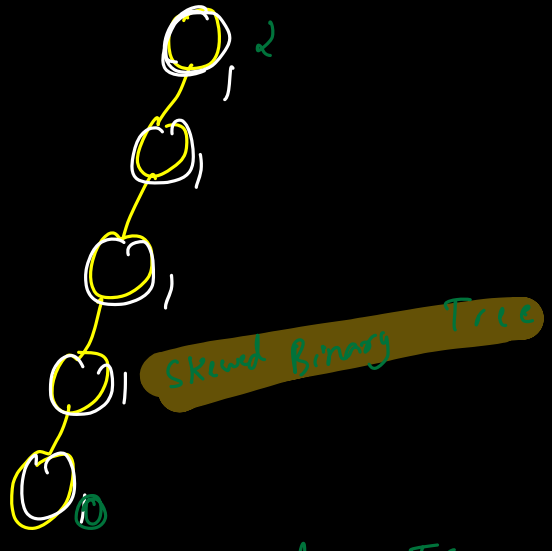
Leaf: the Node with no children





Binary Tree : Any node will have at max 2 children. = {0, 1, 2}





⇒ Linked List is a Skewed Tree

Binary Tree

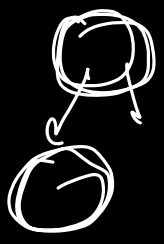
Parent-child

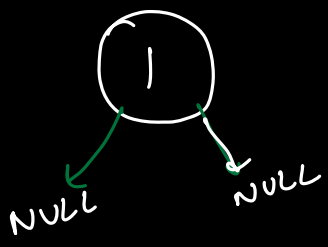
Empty tree is also a Binary Tree

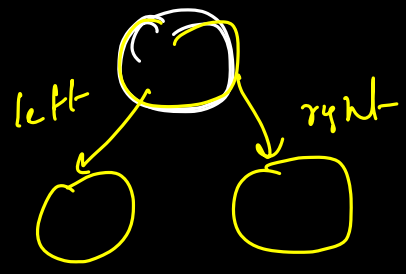
Linked List

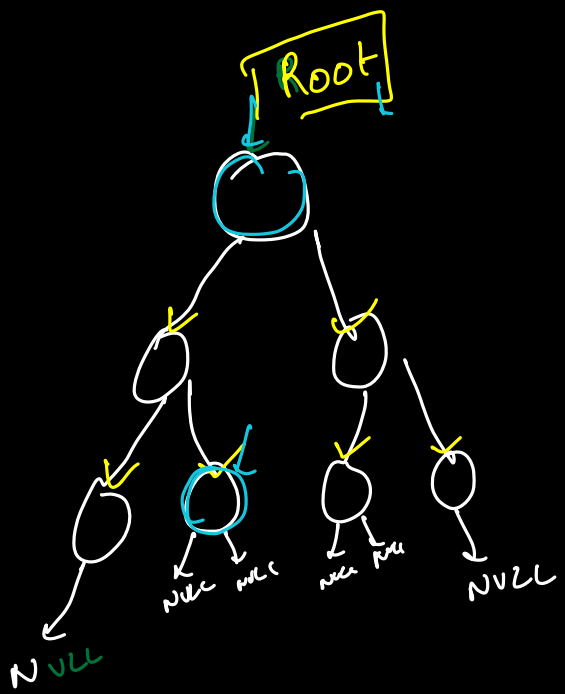
int val
Node next;

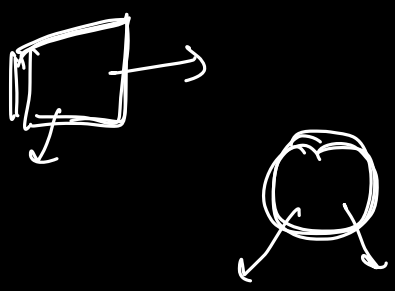
```
Class Node {
    int val;
    Node left;
    Node right;
}
```

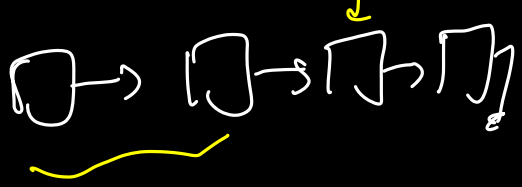


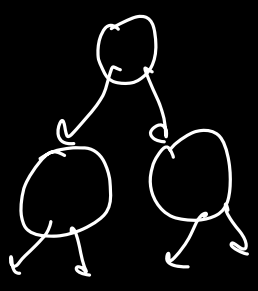


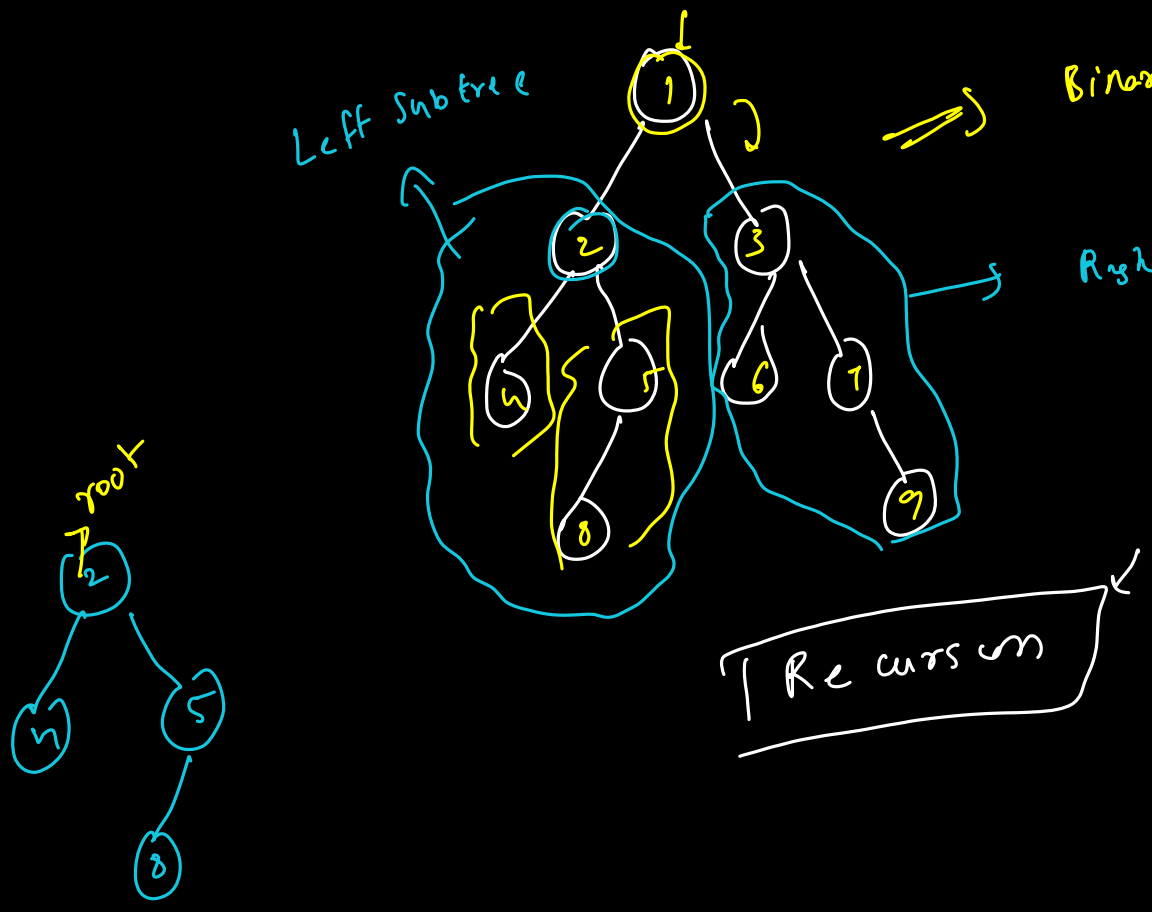










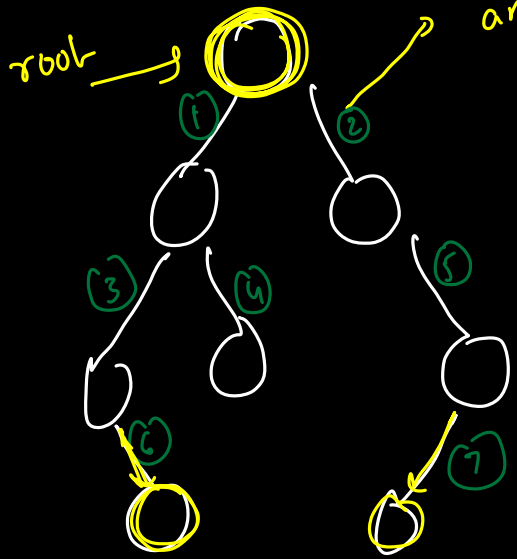


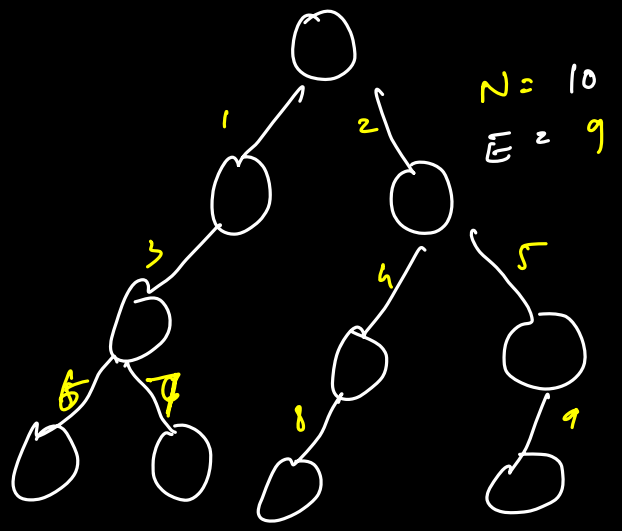
Property

line which connects any 2 nodes

N nodes

$N = 8 \Rightarrow Edge \Rightarrow 7$





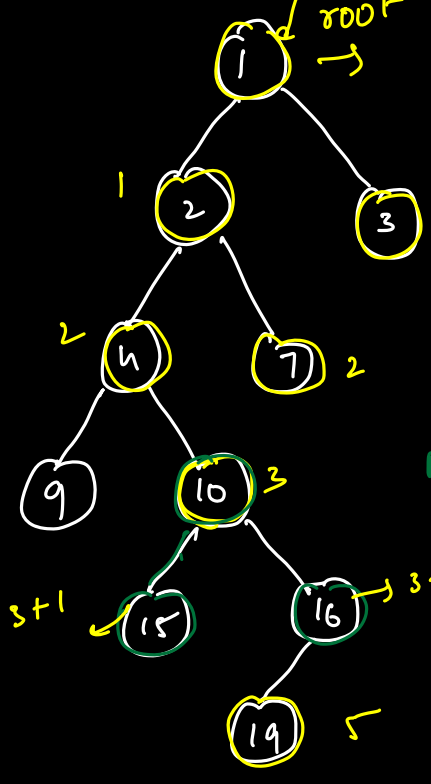
→ All the nodes except root will have exactly 1 incoming Edge.

★ | #Edges = N-1

Depth of a Node (level)

= No. of edges from root of the tree to that node

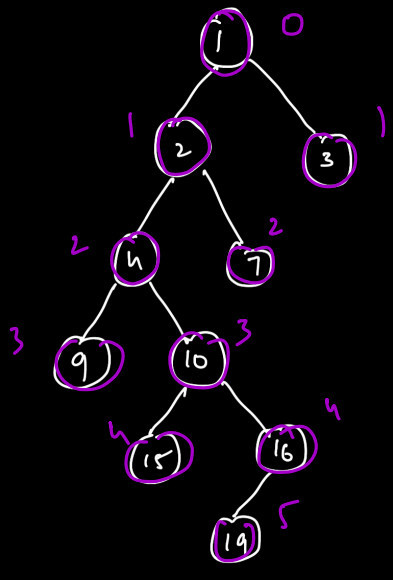
depth(root) = 0



No. of edges from root to 10 = 3

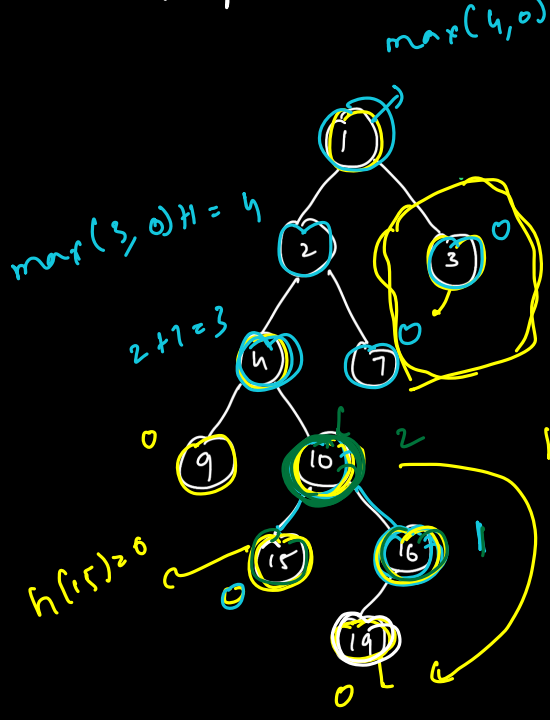
depth(child) = depth(parent) + 1

Depth



Height

No. of edges from the node to the farthest leaf.



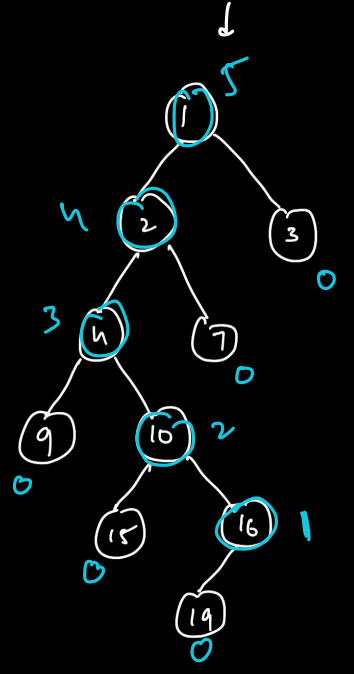
h(1) = 5
h(3) = 0
h(9) =

height(child) + 1

max(left, right) + 1

Height (leaf Node) = 0

height(parent) = 1 + max(height(children)).



Code

Height of tree => Height of root Node

Naming

→ If a tree can have a maximum of k children ⇒ k-ary Tree

Binary Tree ⇒ 2-ary Tree

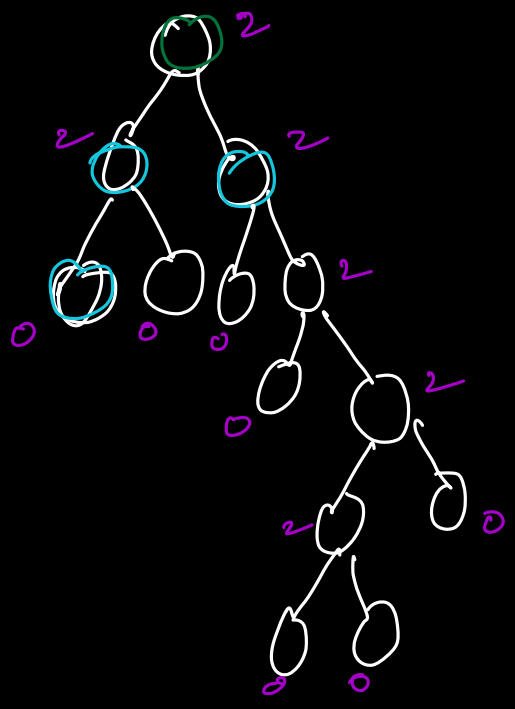
Ternary Tree ⇒ 3-ary Tree

4 children ⇒ 4-ary tree

Types of Binary Trees

1) Proper / Full Binary Tree

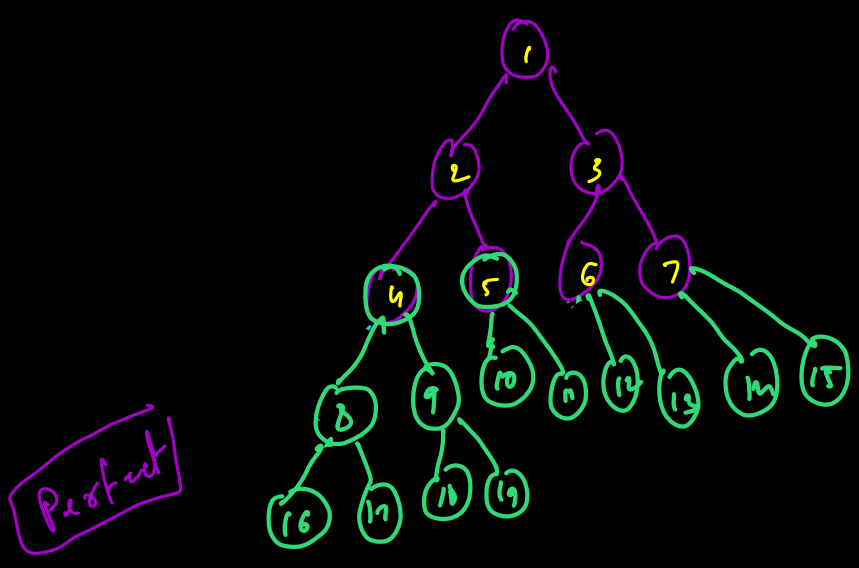
⇒ Every node has 0 / 2 children

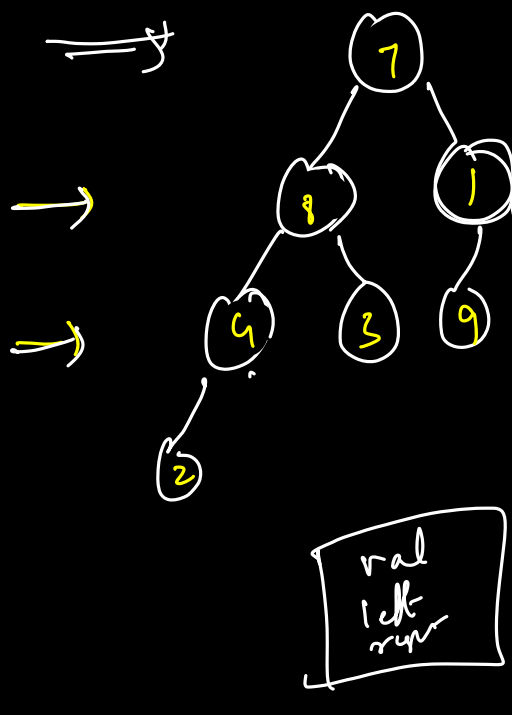


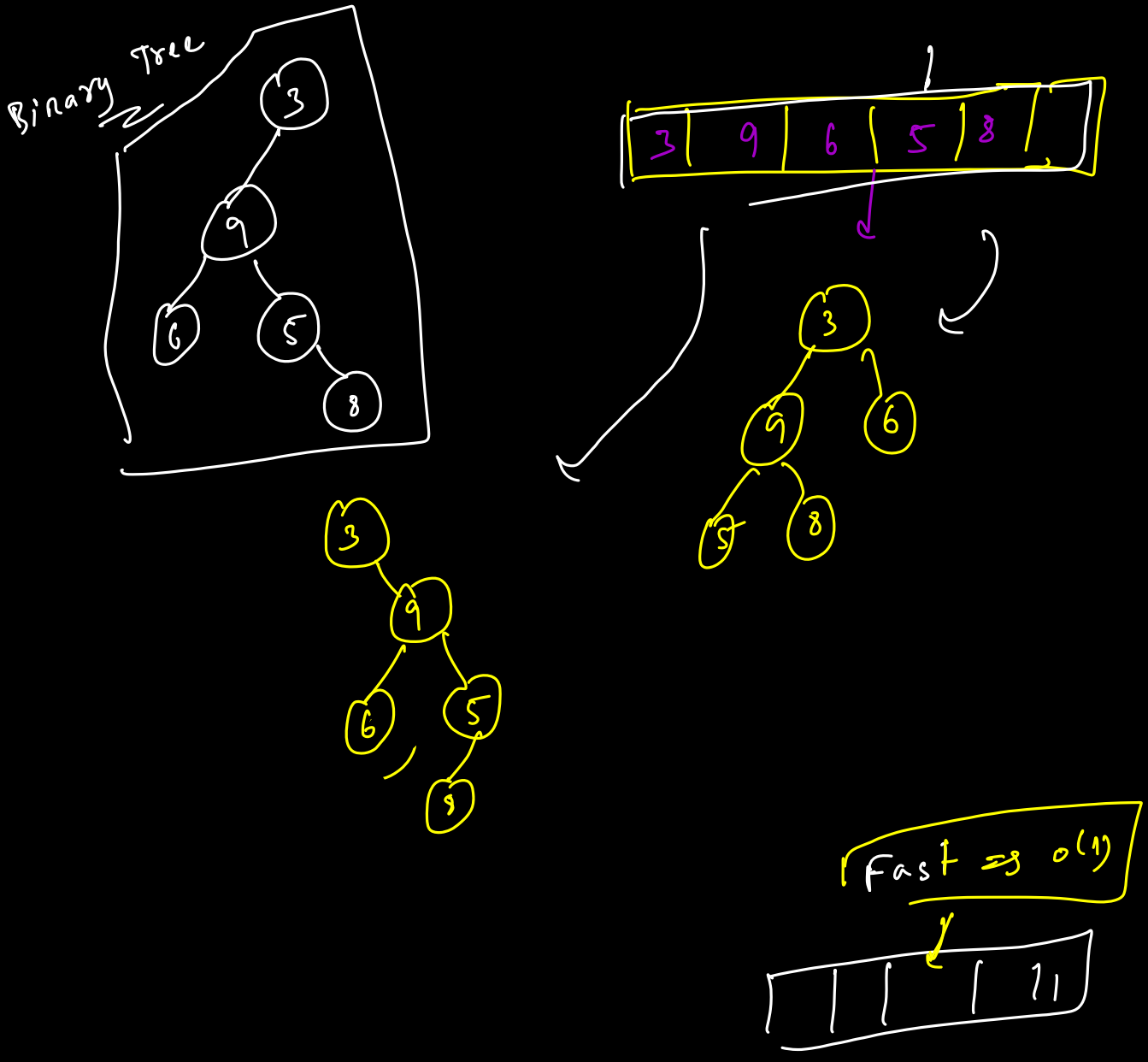
2) Complete B.T

⇒ Every level is completely filled, except possibly the last level.

⇒ If last level is not complete, then all nodes should as left as possible

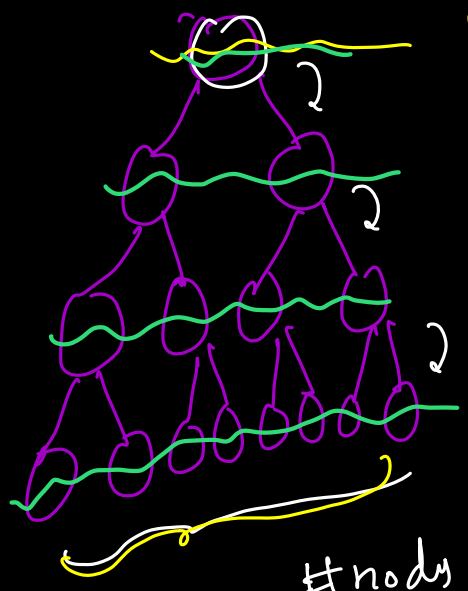





Binary Tree
3
9
6
5
8
3
9
6
5
8
3
9
6
5
8
Fast => o(1)
11

Perfect Binary Tree

→ All the levels are completely full



N nodes

Height (tree) = 3

$1 = 2^0$

$= 2^1$

$= 2^2$

$= 2^3 = 2^h$

→

#nodes in last level = 2^h

$2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \dots\dots + 2^h = N$

$[0, 1, 2, 3, \dots h]$

$[0, h] \Rightarrow h - 0 + 1$

$[l, r] = r - l + 1$

GP

$\frac{a(r^K - 1)}{r - 1} \Rightarrow$ $a = 1$, $r = 2$, $K = h + 1$

$\frac{1(2^{h+1} - 1)}{2 - 1} = N$

$2^{h+1} - 1 = N$

$2^{h+1} = N + 1$

Apply log on both sides

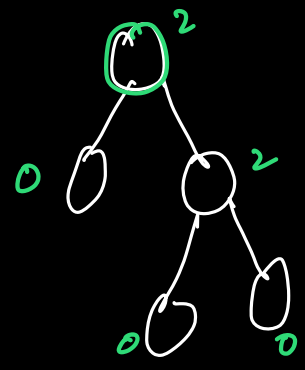
$\log_a^b = b \log_a 1$

$\log_2^{(2^{h+1})} = \log_2^{(N+1)}$

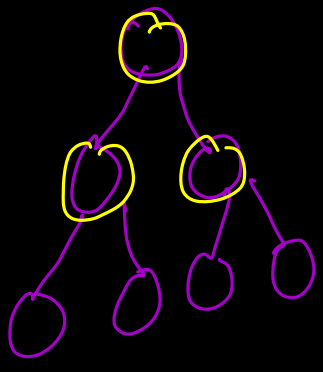
$h+1 = \log_2^{N+1}$

$$h = \log_2^{N+1} - 1$$

Quiz 1



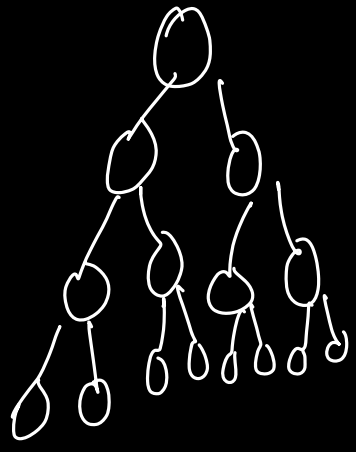
⇒ Full Binary



} 2 child

y → 0 child

Quiz 2



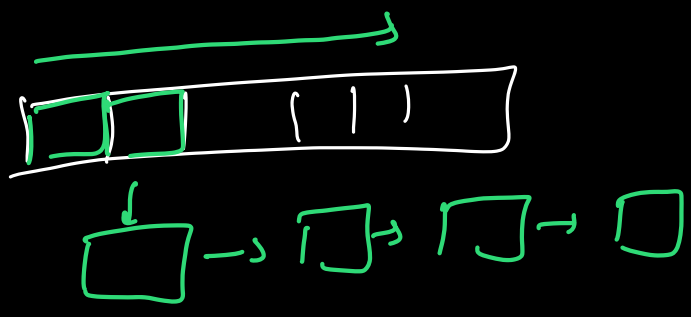
Perfect ⇒ Complete / Full

Complete

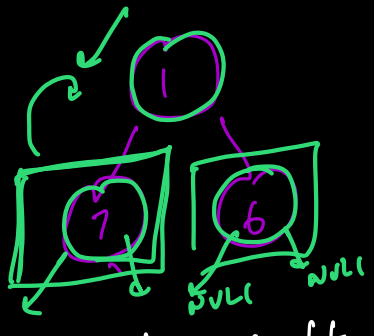
Complete ⇒ Heap

Traversals

Process of visiting every node of the tree exactly once



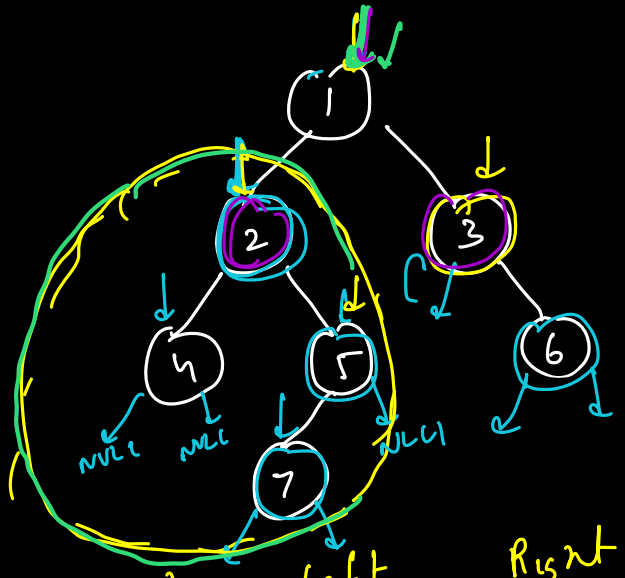
- Depth First Search
  - Preorder Traversal
  - Inorder Traversal
  - Postorder Traversal
- Breadth First Search



Preorder Traversal: Root Left Right

1 7 6

root




```
{ print(1)
  preordu (root.left)
  preordu (root.right)
```

Preordu : Root Left Right

1	2	4	5	7	3	6

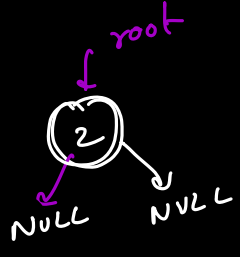
1) Assumption: `preordu (root)` will print the preordu traversal of the tree

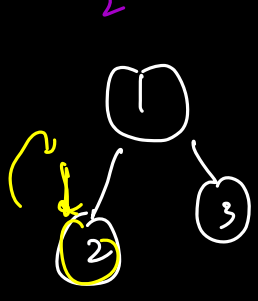
2) Main Logic :

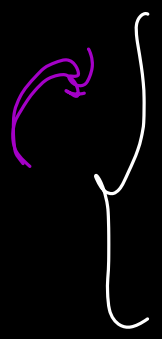
```
print (root.val)
preordu (root.left)
preordu (root.right)
```

3) Base Case :

```
if(root == NULL) {
    return;
}
```





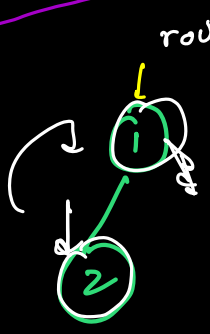


```
void preorder (Node root) {
    if(root == NULL) {
        return;
    }
    print(root.val);
    preordir(root.left);
    preorder(root.right);
}
```

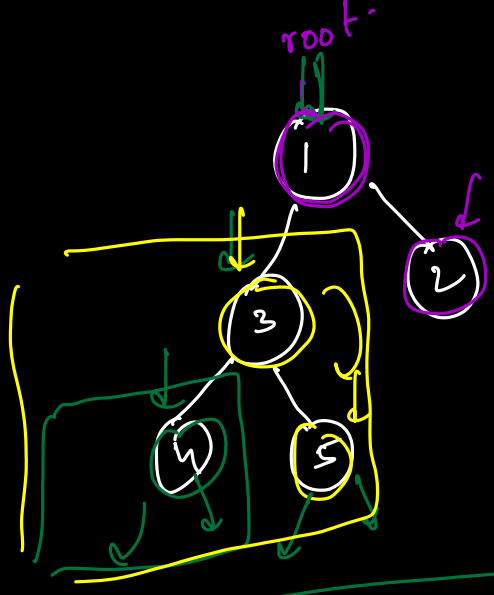
Another Base Condition:

```
if(root == NULL) return;
if( root.left == NULL && root.right == NULL){
    print(root.val);
    return;
}
```

preorder(NULL)



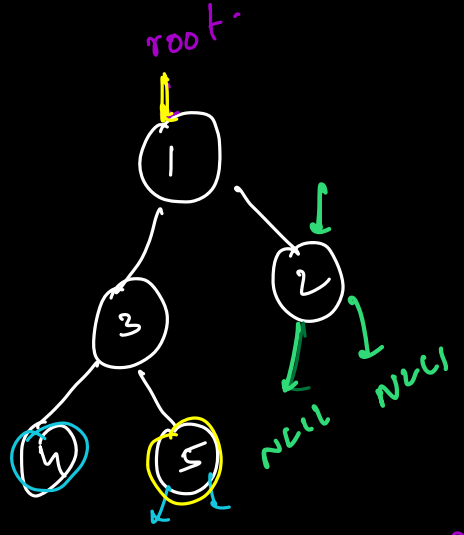
Output: 1 2

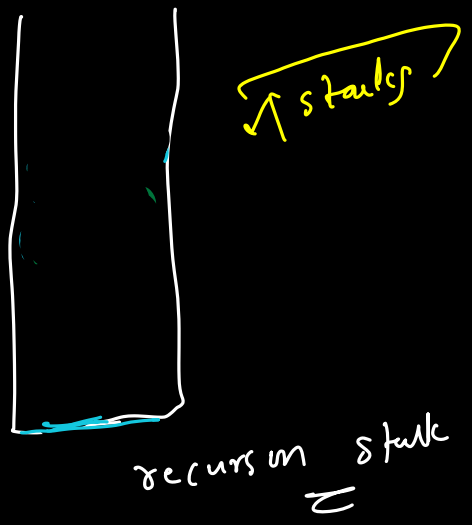


Pre: 1 3 4 5 2

Preorder :

Output : [1 3 4 5 2]

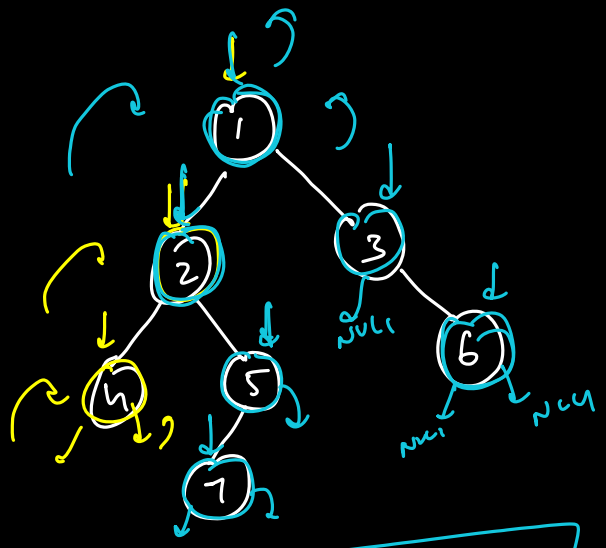





```
void preorder (Node root) {
    if(root == NULL) {
        return;
    }
    print(root.val);
    preorder(root.left);
    preorder(root.right);
    return;
}

preorder(root);
```

2) Inordu : Left Root Right



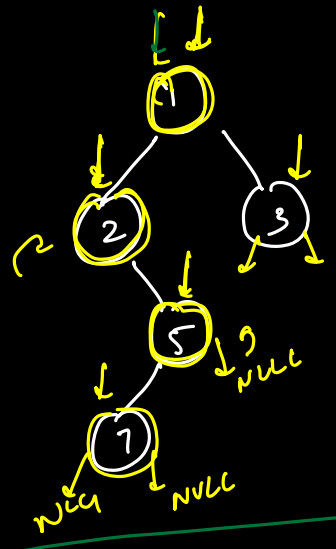
Output : 4 2 7 5 1 3 6

Inordu →

```
global count= 0;
void inordu (Node root){
    if (root == NULL){
        return;
    }
    inordu (root.left);
    print (root.val);
    inordu (root.right);
    count++;
}
```

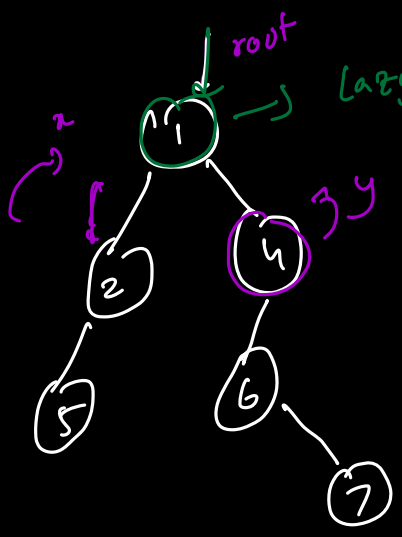
postordu (root.left)
postordu (root.right)
print (root.val)

3) Postorder: Left, Right Root .



Postorder: 7 5 2 3 1.

Question: Given a binary tree, count no. of nodes



#nodes = 6

count = 0;

Assumption: numNodes(root) returns the no. of nodes in that tree

Main Logic:

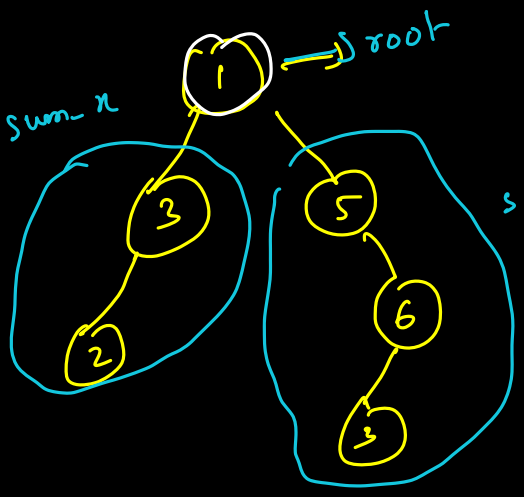
```
left = numNodes(root.left)
right = numNodes(root.right)
return 1 + left + right
```

Base condition:

```
if(root == NULL)
    return 0;
```

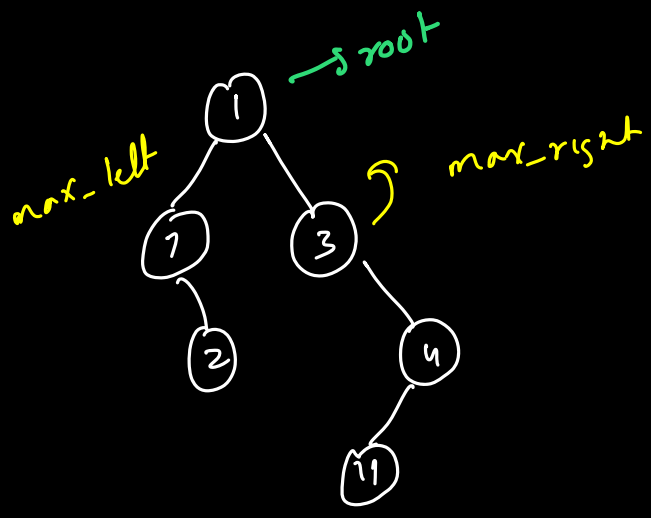
root

Question: Sum of nodes in a tree



sum_x + sum_y + root.val

Question: Max Element of Binary tree

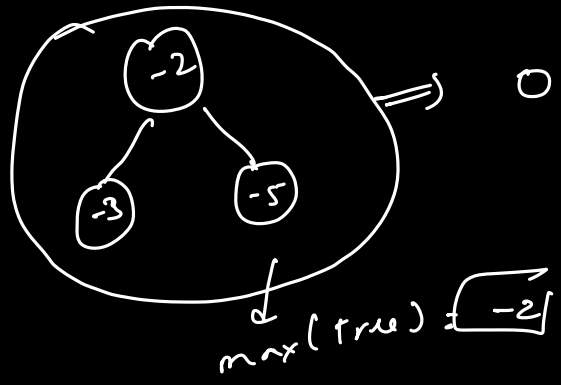


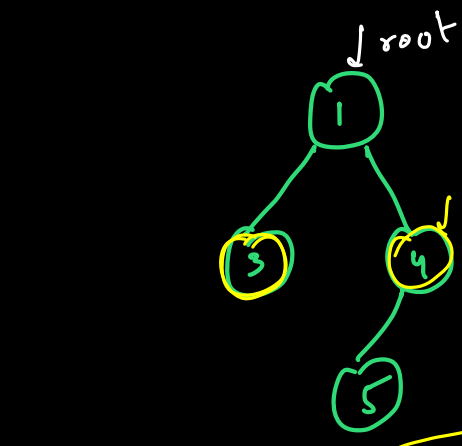
max(max_left, max_right, root.val)

Base Case:

```
if (root == NULL) {
    return 0;
}
```

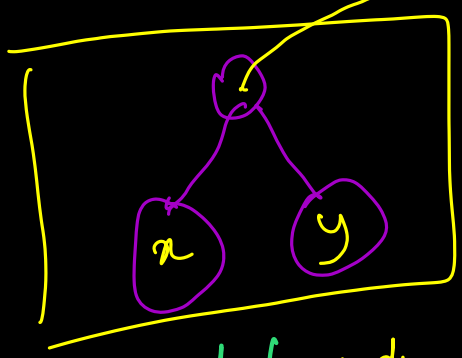
INT_MIN





1
2
3
...

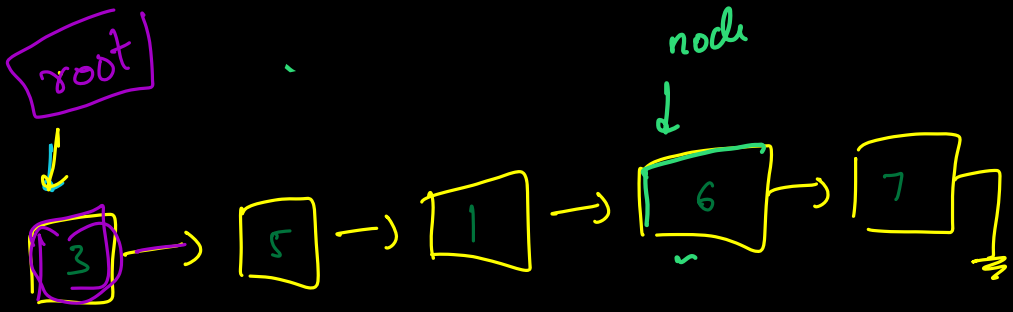
```
int numNodes (root) {
  if(root == NULL){
    return 0;
  }
  left = numNodes(root.left)
  right = numNodes (root.right)
  return 1 + left + right;
```



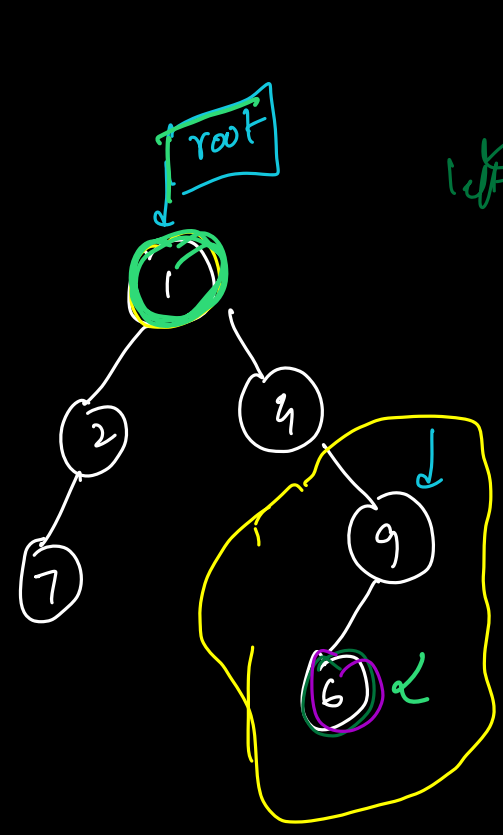

```
int maxNode ( Node root ) {
  if(root == NULL) {
    return -INF;
  }
  left_max = maxNode ( root.left )
  right_max = maxNode (root.right)
  return max( left_max, right_max, root.val);
```

6, 7

```
}
```




```
class Node {
  int data;
  Node next;
}
```



1. left = 2

1. right = 4

root.right.right.left

root.left

root

Preorder

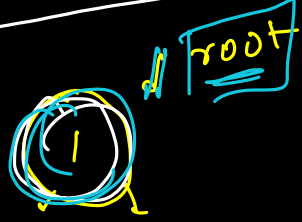
Post

Inorder

root == NULL

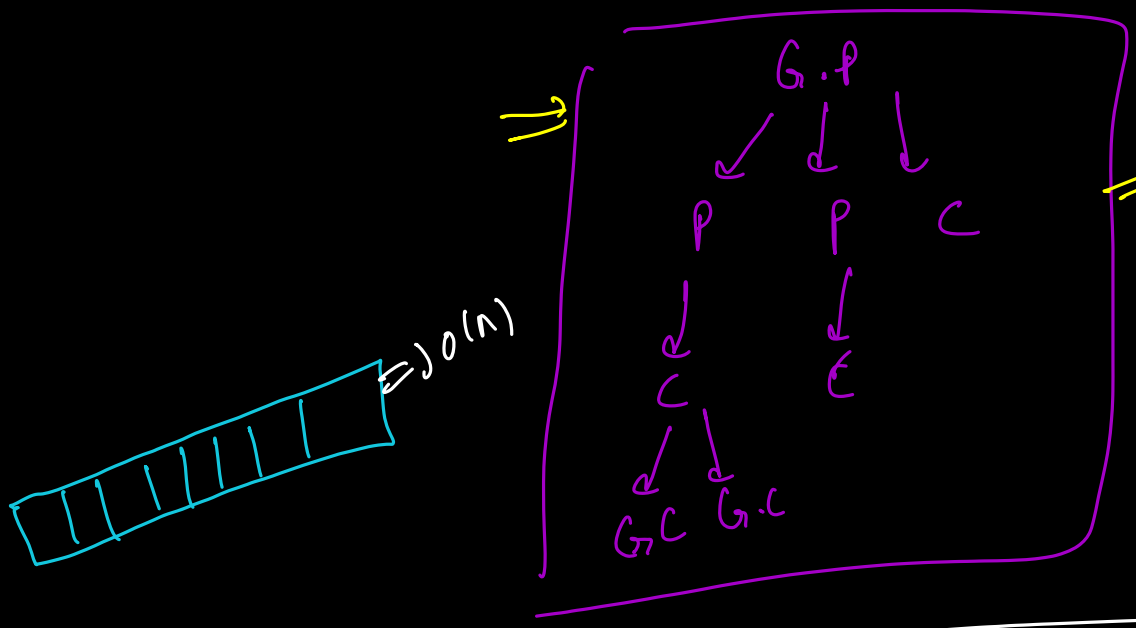
root.left == NULL && root.right == NULL

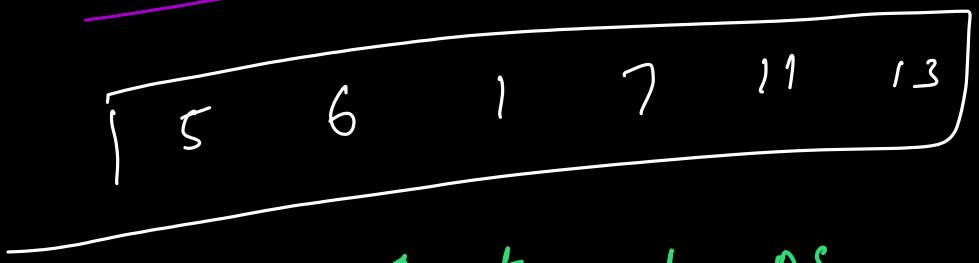
Leaf

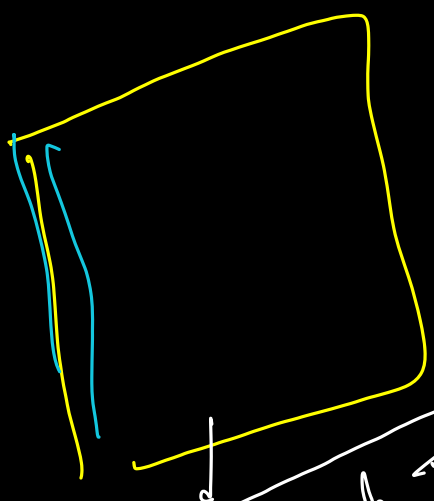


root.left = NULL

root.right = NULL







Binary search Tree

=) ✓ Insert into DS

=) search - num =) O(logn)

=) ✓ Delete Numb

O(logn)

Graph

M C