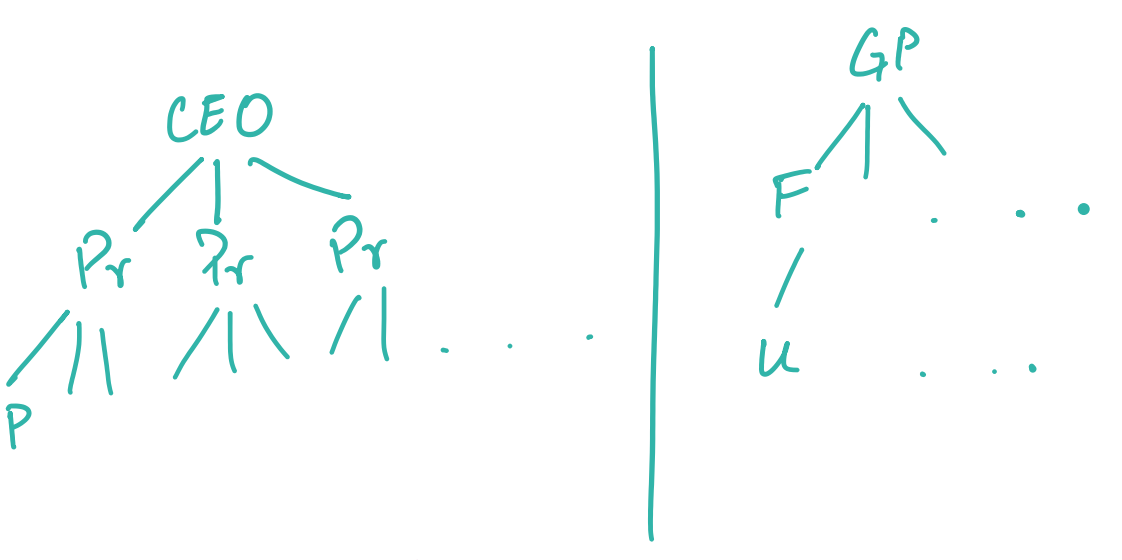
Binary Trees 1

Tuesday, 12 January 2021 4:39 PM

Introduction

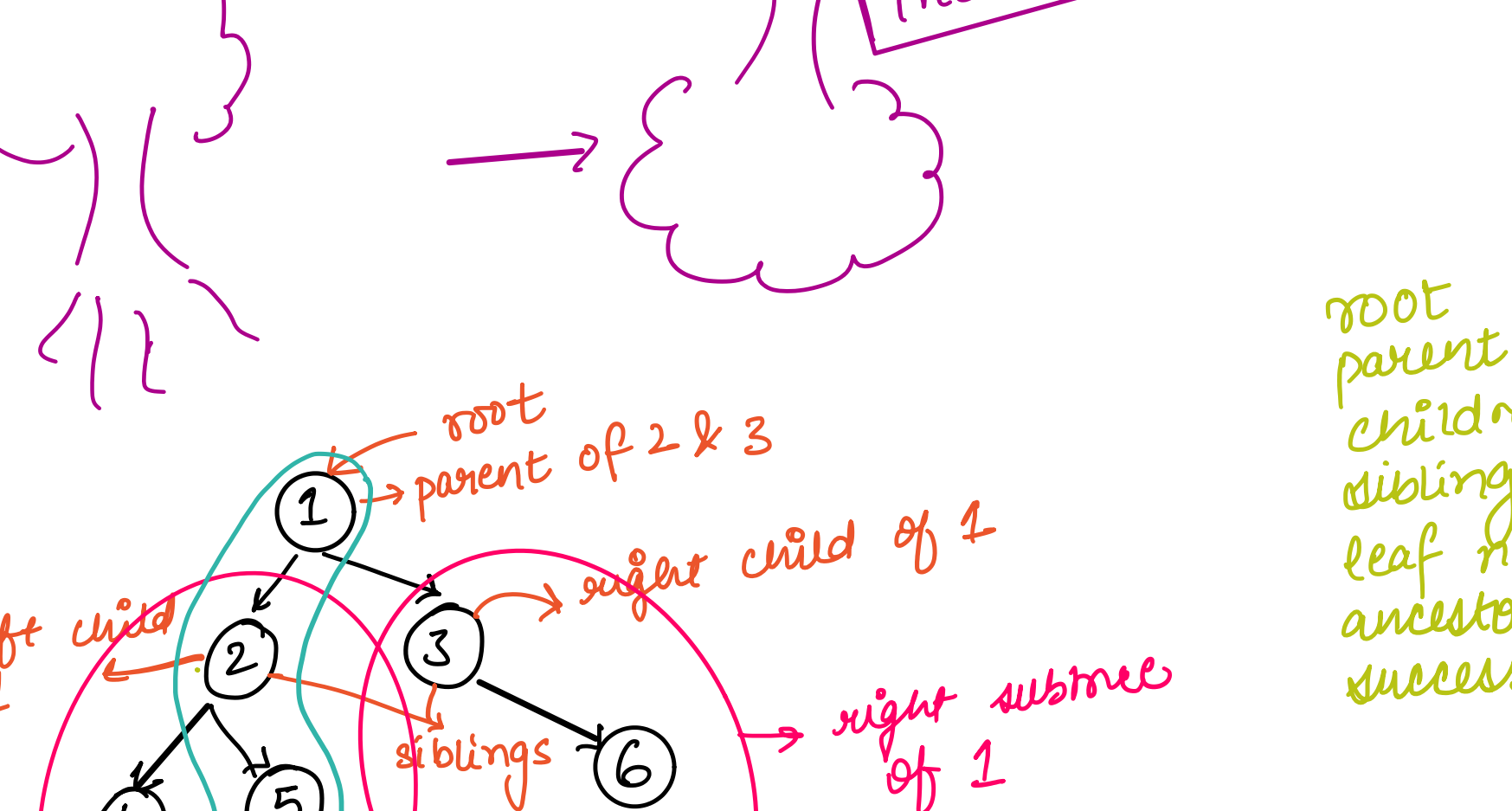
Till now we have studied about linear data structures like arrays, linked list, stacks, queues but for next 5 classes, we're gonna talk about a hierarchical data structure - Trees

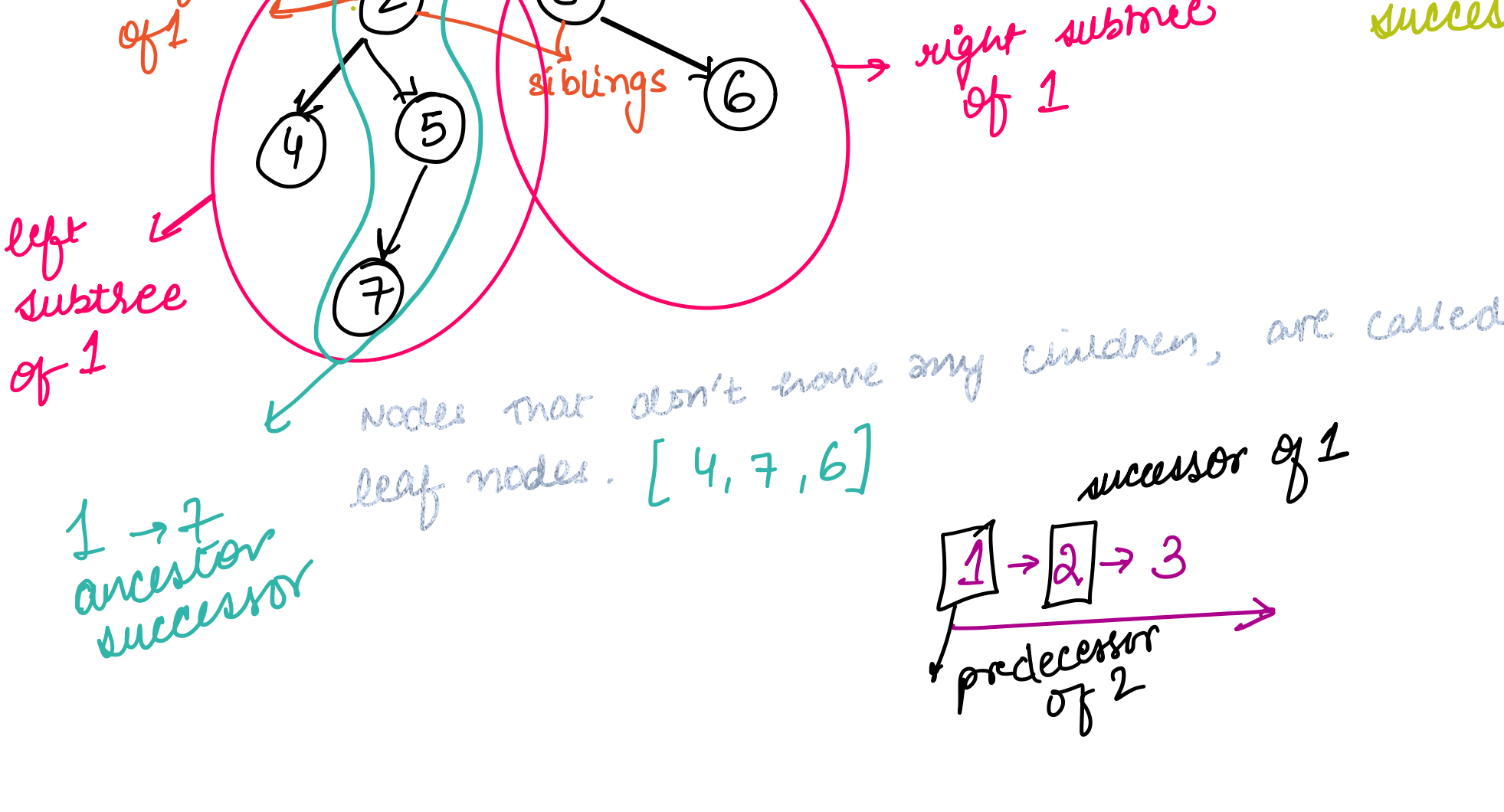
- Family Tree
- Company Tree



Trees ⇒ Graph without cycles

Actual Inverted Tree





root
parent
children
sibling / cousins
leaf nodes
ancestors
successor

Nodes that don't have any children, are called leaf nodes. [4, 7, 6]

1 → 7 ancestor successor

successor of 1

$\boxed{1} \to \boxed{2} \to 3$

predecessor of 2

Depth: Distance from root node to current node

Eg: Depth of 5 is 2 [no. of edges are 2]

Depth of root node → 0 [don't have to travel any edge]

Depth of a node = h
Then depth of it's children will be h+1

Height → Distance from curr node to farthest leaf.

Height of 2 → 2 [There are 2 edges]

" " 3 → 1

" " root → 7 = Height of Tree

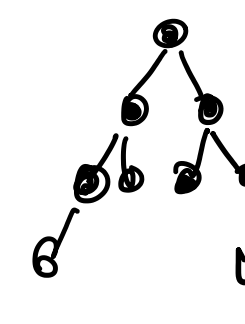
$h(\text{node}) = 1 + \max(\text{height of child nodes})$

If N nodes are there in Tree → N-1 edges

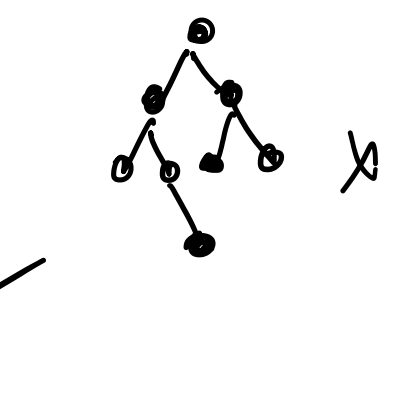
Every node has incoming edge, except the root node.

Naming:

max no. of children for any node is K
Then the tree is known as K ary tree

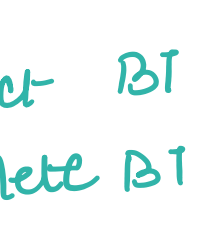


1) Proper BT | Full BT ⇒ Every node has 0 or 2 children



2) Complete BT ⇒ Every level is completely filled, except the last level (possibly)
If last level is not filled then nodes in that level are as left as possible

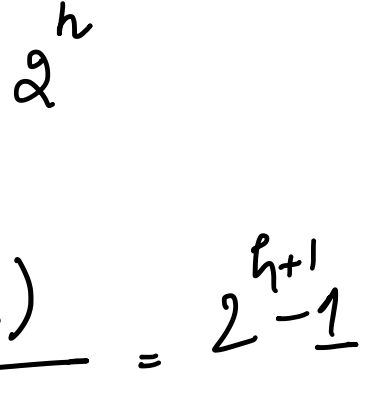
3) Perfect BT - All levels are completely filled



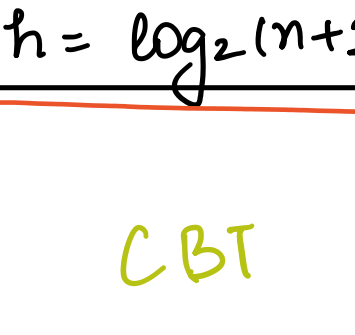
All complete BT are perfect BT X
All perfect BT are complete BT ✓

All Perfect BT are proper BT ✓
All proper BT are perfect BT X

Eg: Tree → Complete
Proper
But not perfect

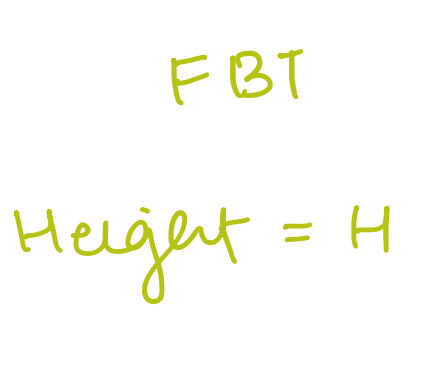


Que.1 Perfect Binary tree with n nodes is there. Tell the Height of that tree.



⑦

1 2^0
2 2^1
4 2^2
.
.
.



$$n = 2^0 + 2^1 + 2^2 + \dots 2^h$$

GP

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

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

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

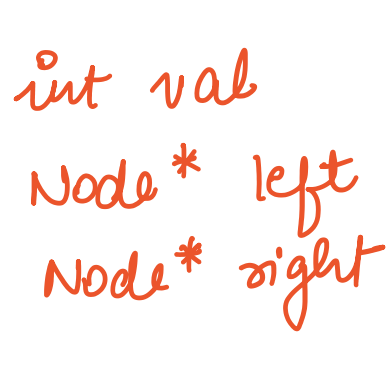
$h + 1 = \log_2(n+1)$

$\boxed{h = \log_2(n+1) - 1}$

CBT

Height = H

H=2



$[4, 7]$

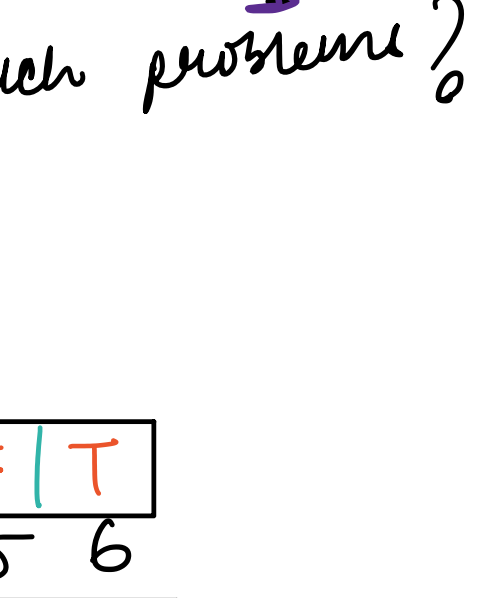
$2^h, 2^{h+1} - 1$

FBT

Height = H

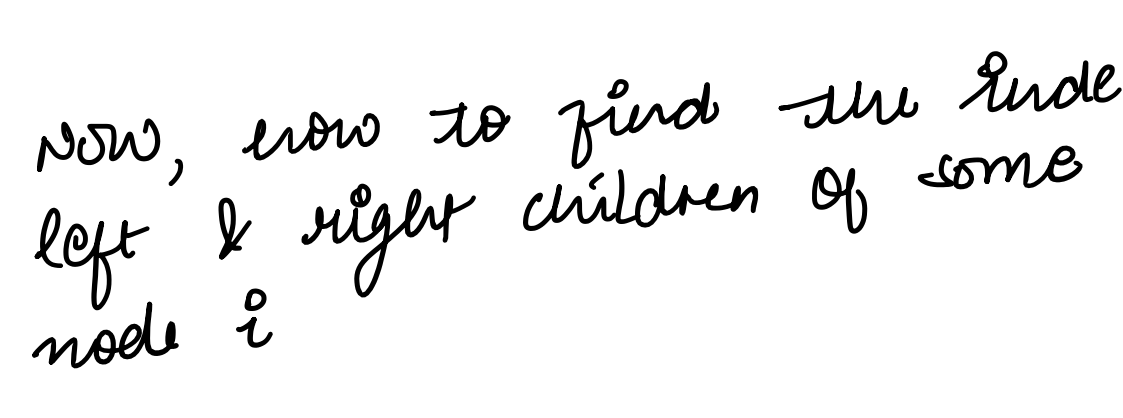
Min height 7

$\boxed{2h+1}$



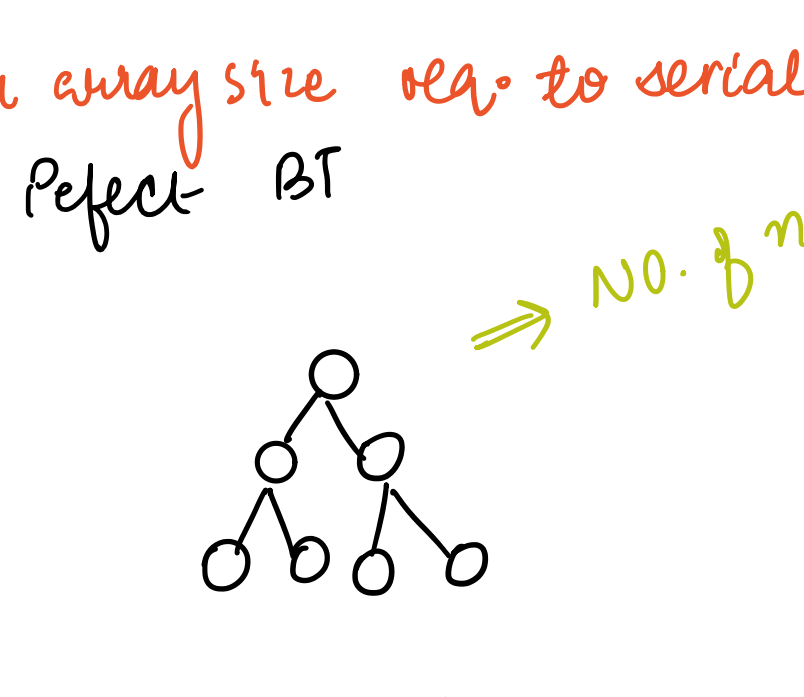
Structure of a tree node

```
class Node{
    int val
    Node* left
    Node* right
}
```



Suitable paradigm to solve such problems?
— Recursion

Serialization & Deserialization [storing Tree in arrays]



bool array

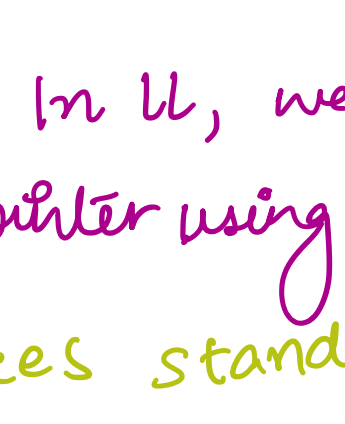
T	T	T	T	T	F	T
0	1	2	3	4	5	6
10	4	3	1	9	-1	5

Now, how to find the index of left & right children of some node i

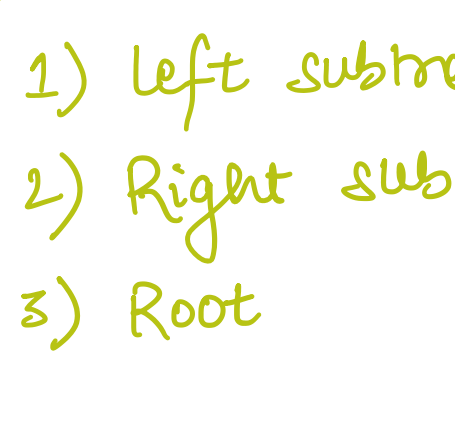
Left → $2i+1$
Right → $2i+2$
Parent → $\left\lfloor \frac{i-1}{2} \right\rfloor$

Max array size req. to serialize

1) Perfect BT ⇒ No. of nodes



2) Complete BT ⇒ No. of nodes



3) FBT → Space wastage

1 Based indexing
Left → $2i$
Right → $2i+1$
Parent → $i/2$

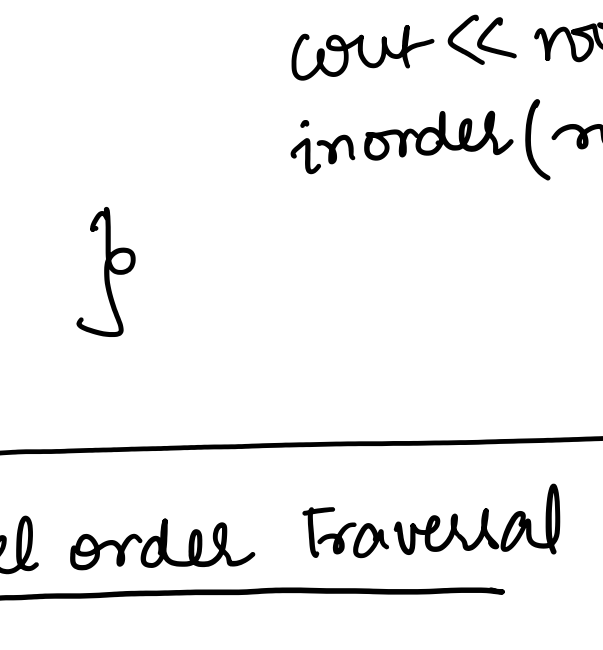
Tree Traversal

Like In LL, we're given head ptr, in case of BT we have a root pointer using which we can traverse the whole tree.

Choices standing at root
1) Left subtree
2) Right subtree
3) Root

+AB
A+B
AB+

NLR	Root	Left	right	⇒ Preorder
LNR	left	Root	Right	⇒ Inorder
LRN	left	right	Root	⇒ Postorder



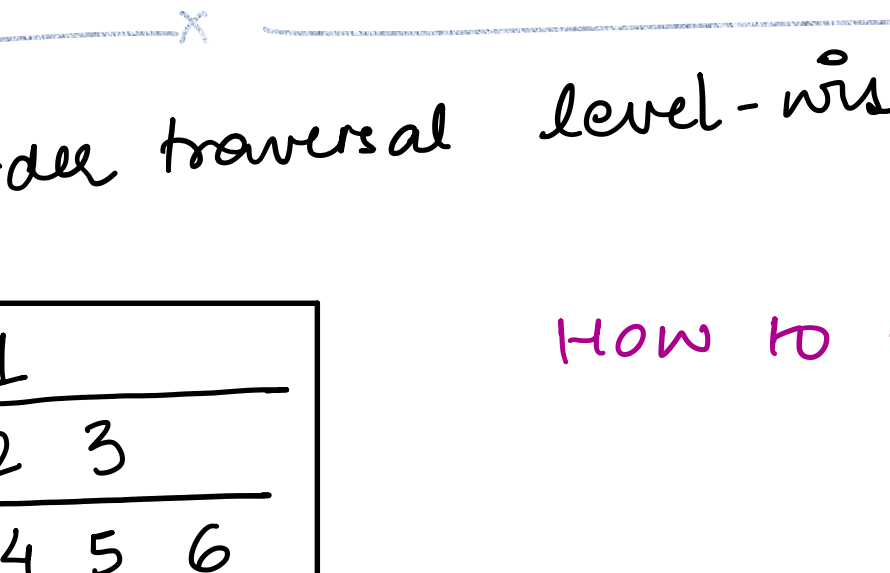
Dry run for
→ Inorder
→ Preorder
→ Postorder

Code

```
void inorder(root){
    if(root == NULL) return;
    inorder(root->left);
    cout << root->val;
    inorder(root->right);
}
```

Write yourself for preorder & postorder.

Level order Traversal



1 2 3 4 5 6 7

~~1 2 3 4 5 6 7~~

o/p

$\boxed{1\ 2\ 3\ 4\ 5\ 6\ 7}$

---

level order traversal level-wise

1
2 3
4 5 6
7

How to move to a next level?

Good delimiter for nodes → NULL

① Approach 1 NULL can be inserted after each level.

Pseudocode

```
level_order_null(root){
    queue<Node*> q;
    q.push(root)
    q.push(null)
    while( q.size()>1 ){
        Node* t = q.front()
        q.dequeue()
        if ( t != NULL){
            cout << t->val
            if(t->left) q.push(t->left)
            if(t->right) q.push(t->right)
        }
        else{
            q.push(NULL)
        }
    }
}
```

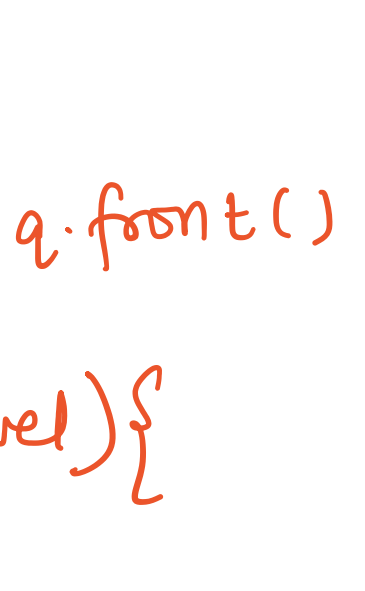
② Approach 2

Push <Node, level> into Queue.

initially <4,0>

$\boxed{\text{cur\_level} = 0}$

~~<4,0>~~	<1,1>	<2,1>



```
level_order_pair(root){
    queue< pair<Node*, int>> q
    cur_level = 0
    q.push({root, 0})
    while (!q.empty()){
        pair<Node*, int> t = q.front()
        q.dequeue()
        if(t.second != cur_level){
            cout << "\n";
            cur_level++
        }
        cout << t->val
        if (t.first->left) q.push({t.left, t.second+1})
        if (t.first->right) q.push({t.right, t.second+1})
    }
}
```