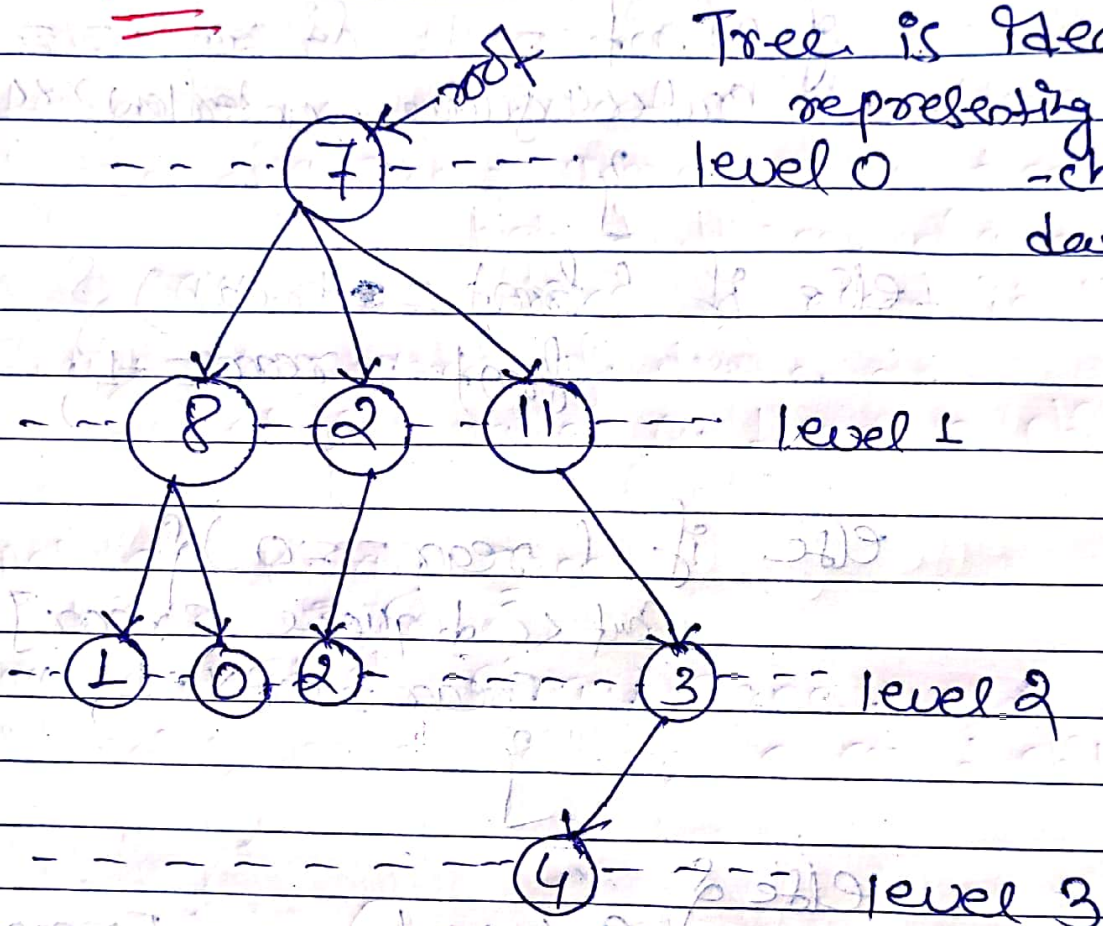
# Tree




Tree is ideal for representing hierarchical data.

## Terminology

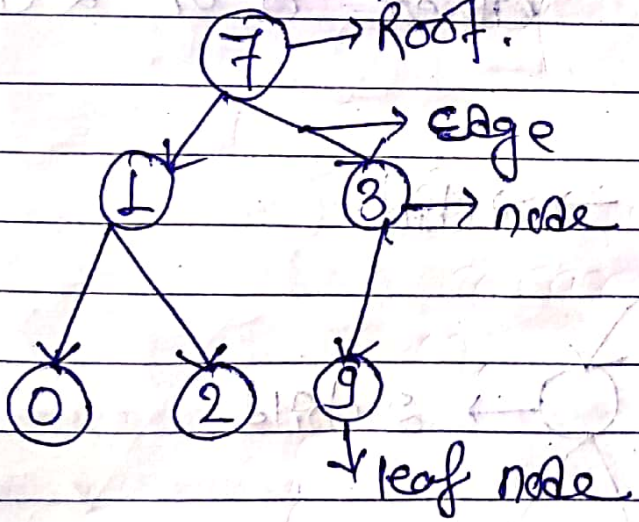
(i) Root :- Topmost node.
(ii) parent :- node which connects with child.
(iii) child :- node which is connected by another node & it is child.
(iv) Leaf/External node :- node with no children.
(v) Internal node :- nodes with atleast one child.
(vi) Depth :- no. of edges from root to the node.
(vii) Height :- no. of edges from node to the deepest leaf.

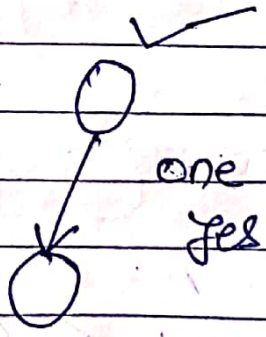
(VIII) Siblings:- nodes belonging to the same parent.
(IX) Ancestors / Descendents:- parent or parent of parent / child or child of child.

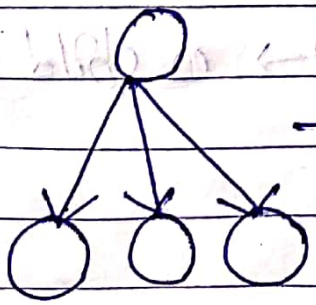
Height of LL is 2.
Depth of LL is 1.

## Binary Tree

Binary tree is a tree which has at most 2 children for all the nodes.



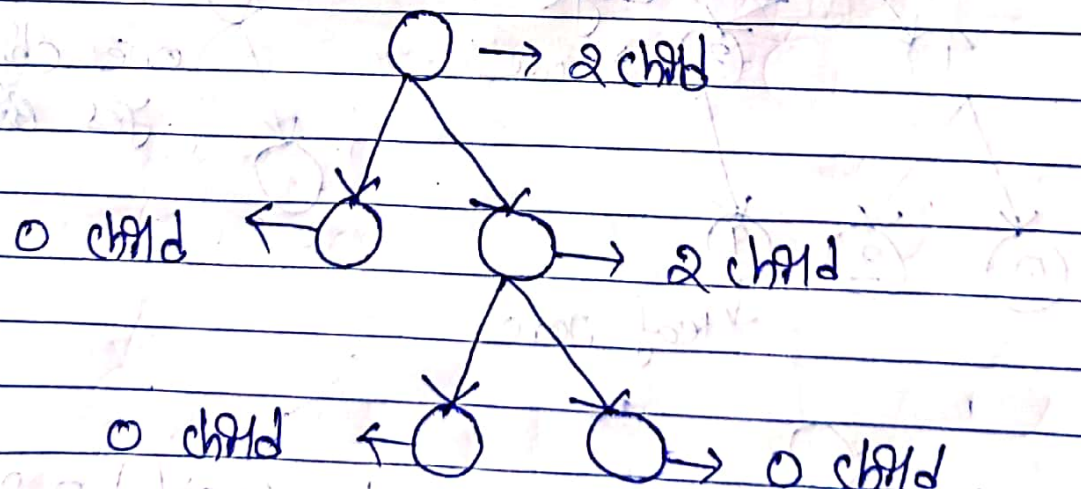



→ NO, deg (root) = 3

(i) Tree is made up of node & Edges.
(ii) nodes → (n-1) edges.
(iii) Degree → no. of direct children. (for a node)
(iv) Degree of a Tree → Degree of a tree is the highest degree of a node among all the nodes present in the tree.
(v) Binary tree = Tree of degree 2.
nodes can have 0, 1 or 2 children.

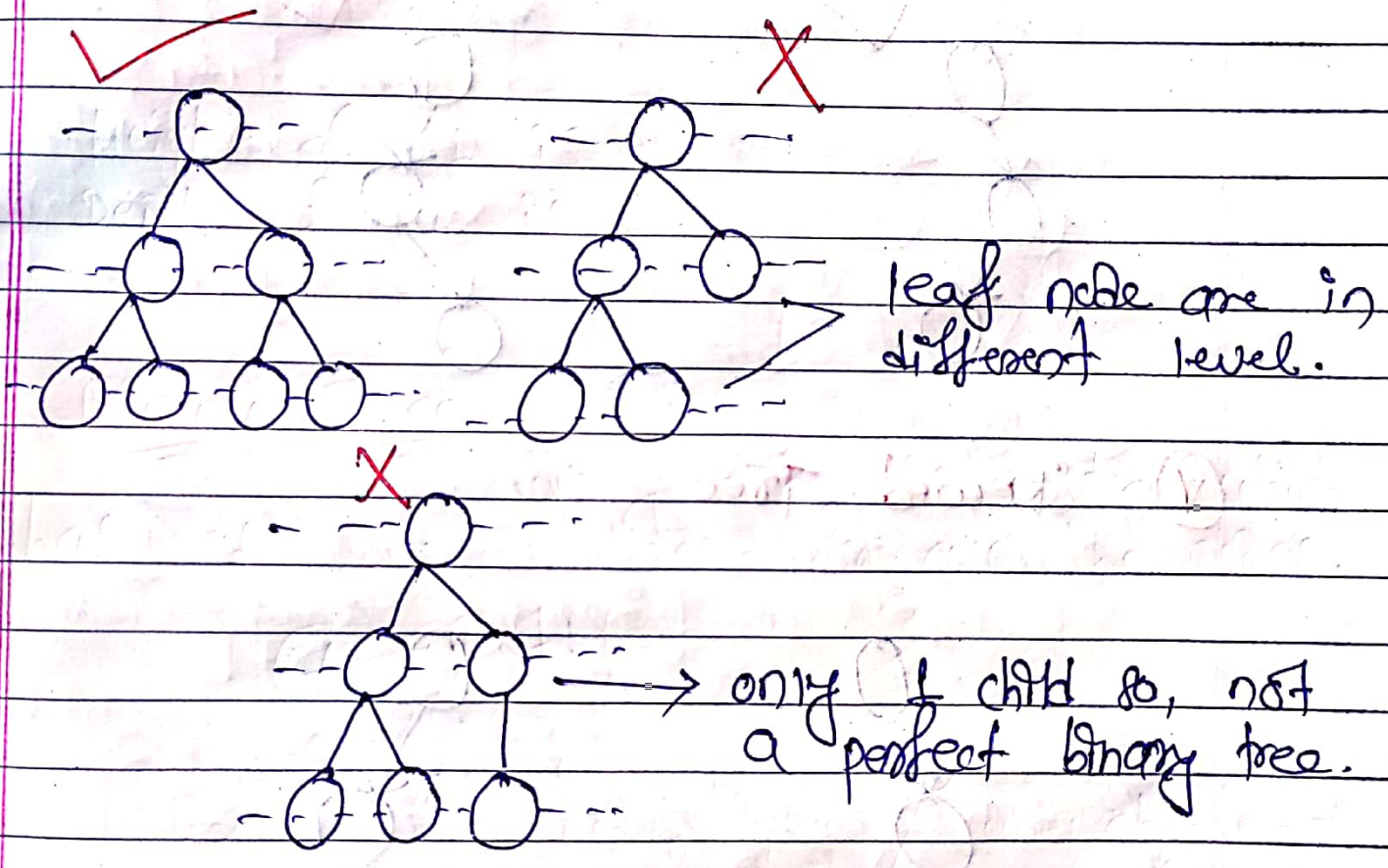
## Types of Binary Tree

(i) Full or Strict Binary Tree.
→ All nodes have either 0 or 2 children.



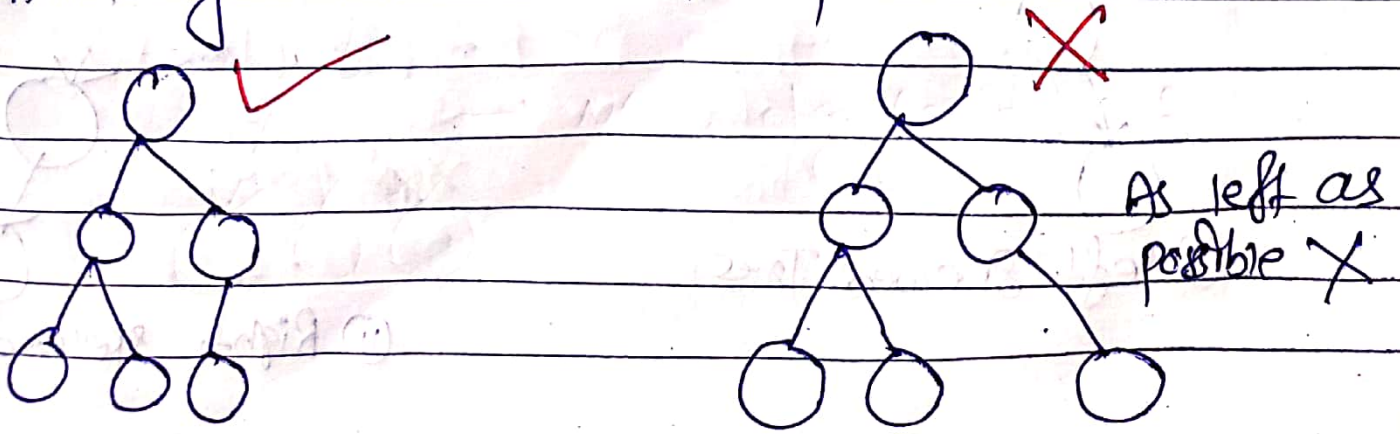
(ii) Perfect Binary Tree

→ Internal nodes have exactly 2 children and all leaf nodes are on same level.



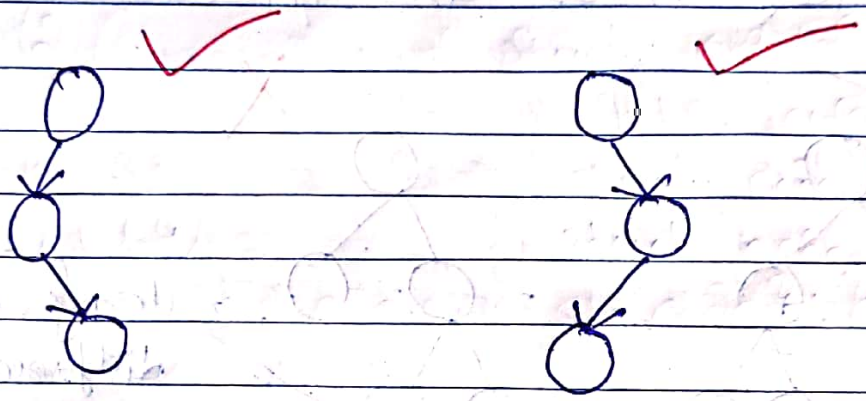

(iii) Complete Binary Tree

→ All levels are completely filled except possibly the last level and last level must have its key as left as possible.

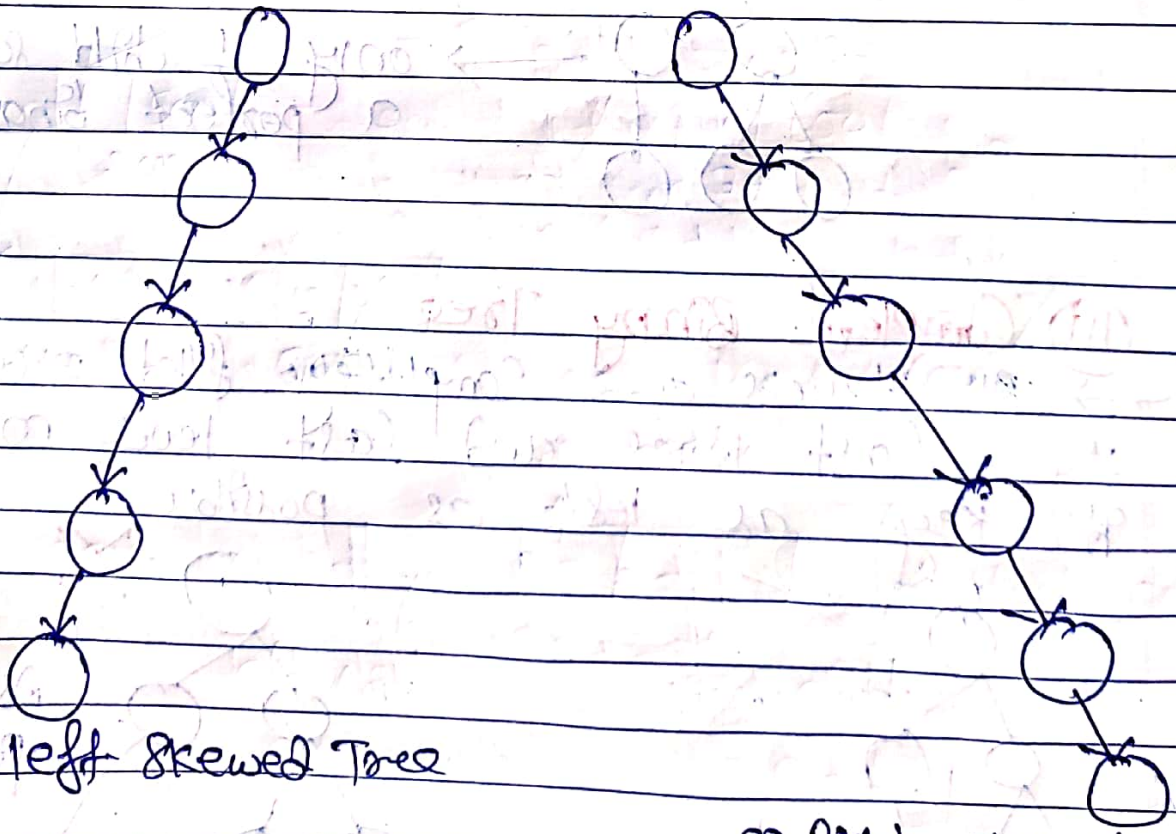


(iv) Degenerate Tree

→ parent node has exactly one child.



(v) Skewed Tree

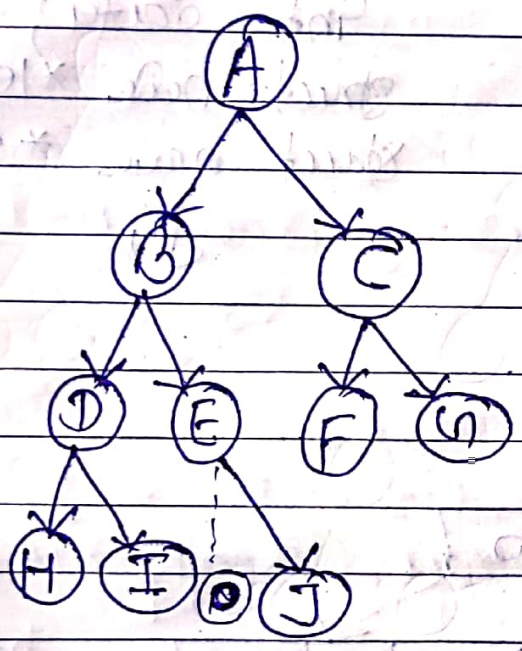


(i) left skewed Tree

(ii) Right skewed Tree

# Representation of Binary Tree

① Array Representation / Sequential Representation.




if a node is at $i^{th}$ index:-
→ left child would be at $[(2*i)+1]$
→ right child would be at $[2*i+2]$
→ parent would be $\left[\frac{i-1}{2}\right]$

| A | B | C | D | E | F | G | H | I | null | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

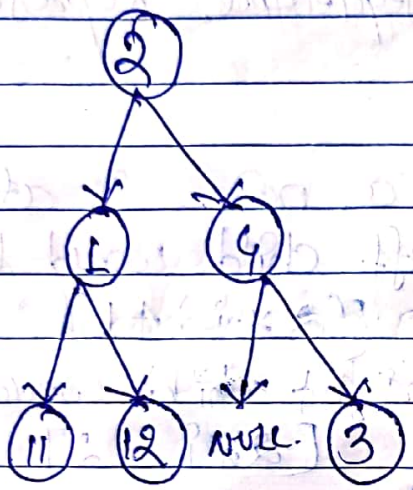
Case 'I'.

| A | B | C | D | E | F | G | H | I | null | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

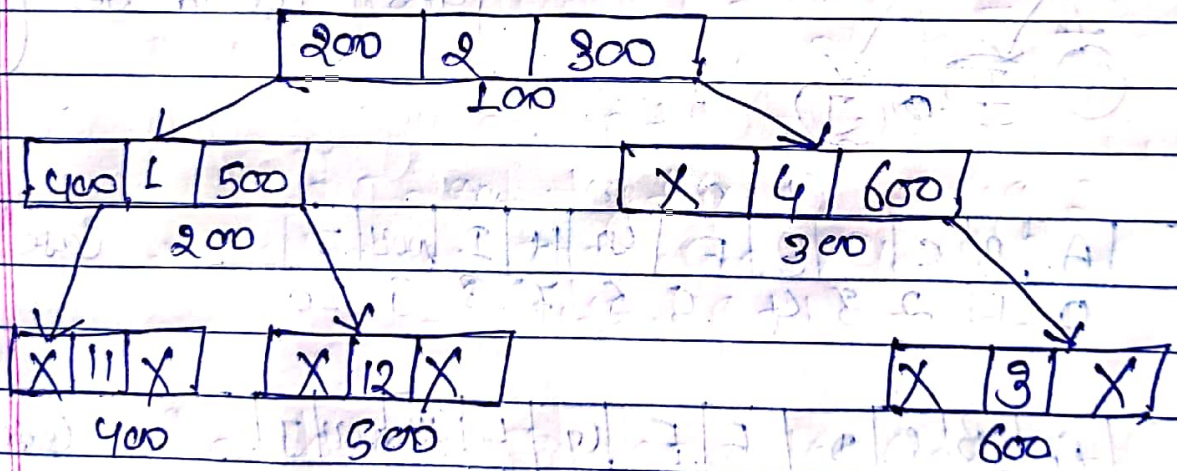
Case 'II'

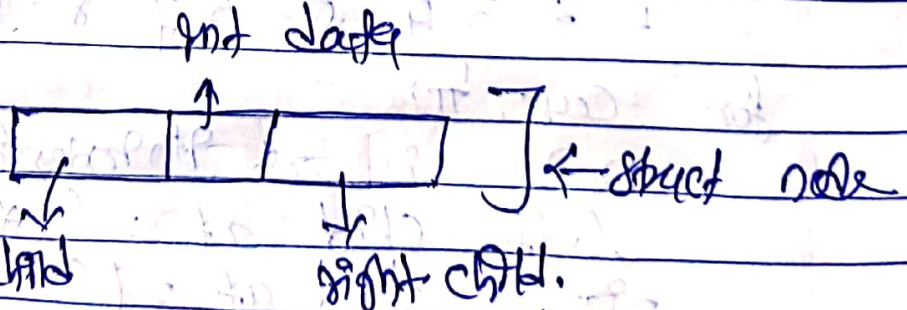
for Case 'II'
node is at $i^{th}$ index:-
left child at = $(2*i)$
right child at = $[(2*i)+1]$
parent at = $\lceil i/2 \rceil$

(ii) Linked Representation



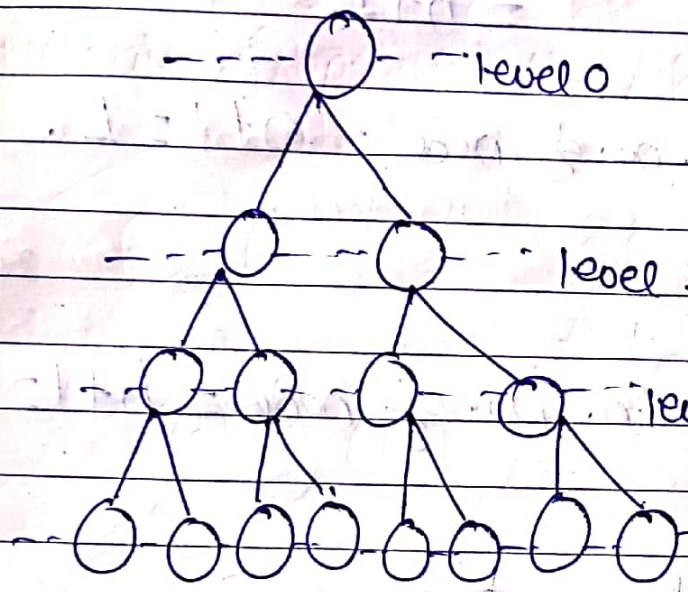

```
struct node {
    int data;
    struct node *left;
    struct node *right;
};
```



predecessor = previous

successor = next node.




max. node = 1 ($2^0$)

max. node = 2 ($2^1$)

max. node = 4 ($2^2$)

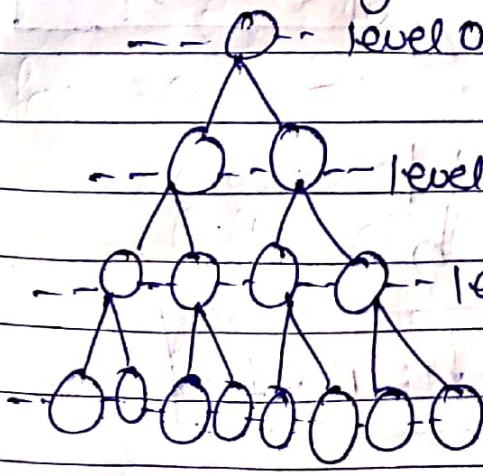
max. node = 8 ($2^3$)

# max. no. of nodes possible at any level $i$ is $2^i$.

$i \rightarrow$ no. of level.

Eg: max. node possible in 4th level $= 2^i = 2^4 = 16$

# max. no. of nodes of height $h$.




$= 1 + 2 + 4 + 8$

$= 2^0 + 2^1 + 2^2 + 2^3$

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

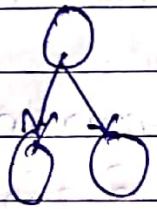
$= (2^{3+1} - 1) = 2^4 - 1$

$= 16 - 1$

$= 15$

# min. no of nodes of height h.
$= h+1$

$h=0$ ○ min no. of node $= 0+1 = 1$.

$h=1$  min. no. of node $= h+1 = 1+1 = 2$.

min. height of a tree,

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

$$\text{or, } n+1 = 2^{h+1}$$

$$\text{or, } \log(n+1) = \log_2 2^{h+1}$$

$$\text{or, } \log(n+1) = h+1$$

min. height $\Rightarrow \boxed{h = \log(n+1) - 1}$

max. height, $n = h+1$

$$n-1 = h$$

max. height $\Rightarrow \boxed{h = n-1}$

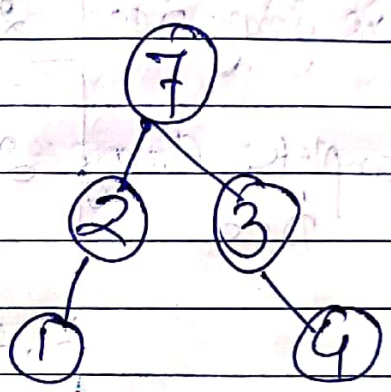
| | max. nodes | min. nodes |
|---|---|---|
| Binary tree | $2^{h+1}-1$ | $h+1$ |
| full Binary tree | $2^{h+1}-1$ | $2h+1$ |
| Complete Binary tree | $2^{h+1}-1$ | $2^h$ |

| | min. height | max. height |
|---|---|---|
| Binary tree | $\lceil \log_2(n+1) \rceil - 1$ | $n-1$ |
| full Binary | $\lceil \log_2(n+1) \rceil - 1$ | $\left(\frac{n-1}{2}\right)$ |
| Complete Binary | $\lceil \log_2(n+1) \rceil - 1$ | $\log n$ |

# Traversal in a Binary Tree

(I) BFS (Breadth first search)
↳ level order traversal.

level order traversal is,
7 - 2 - 3 - 1 - 4.




(II) Depth first search (DFS)

1) pre order traversal (Root → left → Right)

7 - 2 - 1 - 3 - 4

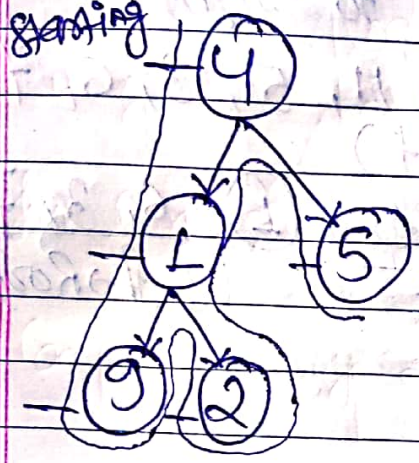
2) postorder Traversal (left → Right → Root)

1 - 2 - 4 - 3 - 7

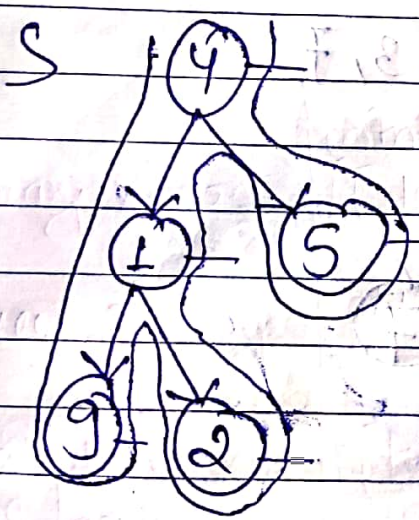
3) Inorder Traversal (left → Root → Right)

1 - 2 - 7 - 3 - 4.

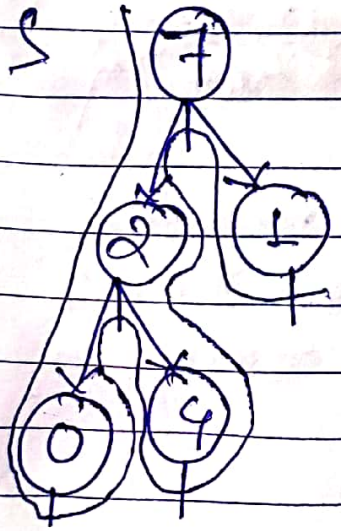
# Trick to find Inorder, preorder & postorder Traversal.




preorder :- 4 - 1 - 9 - 2 - 5




postorder :- 9 - 2 - 1 - 5 - 4



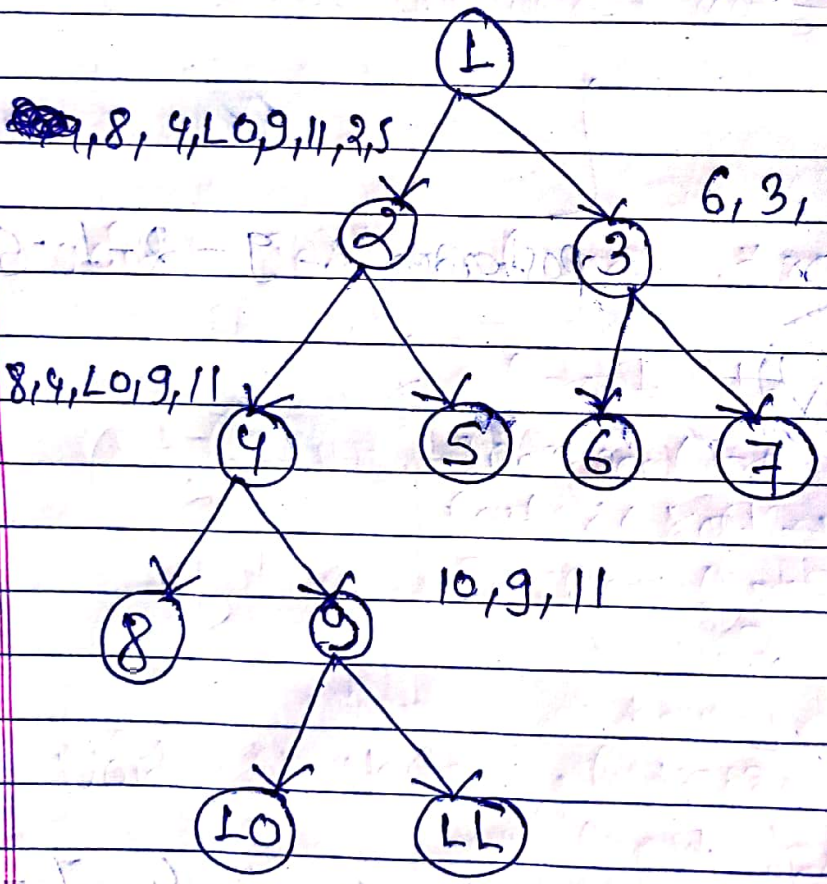

Inorder :- 0 - 2 - 4 - 7 - 1

Construct a Binary tree from preorder & postorder

→ Root scan from left to right.

preorder :- (1), 2, 4, 8, 9, 10, 11, 5, 3, 6, 7
(Root → left → Right)

Inorder :- 8, (4), 10, (9), 11, (2), 5, (1), 6, 3, 7
(left → Root → Right)



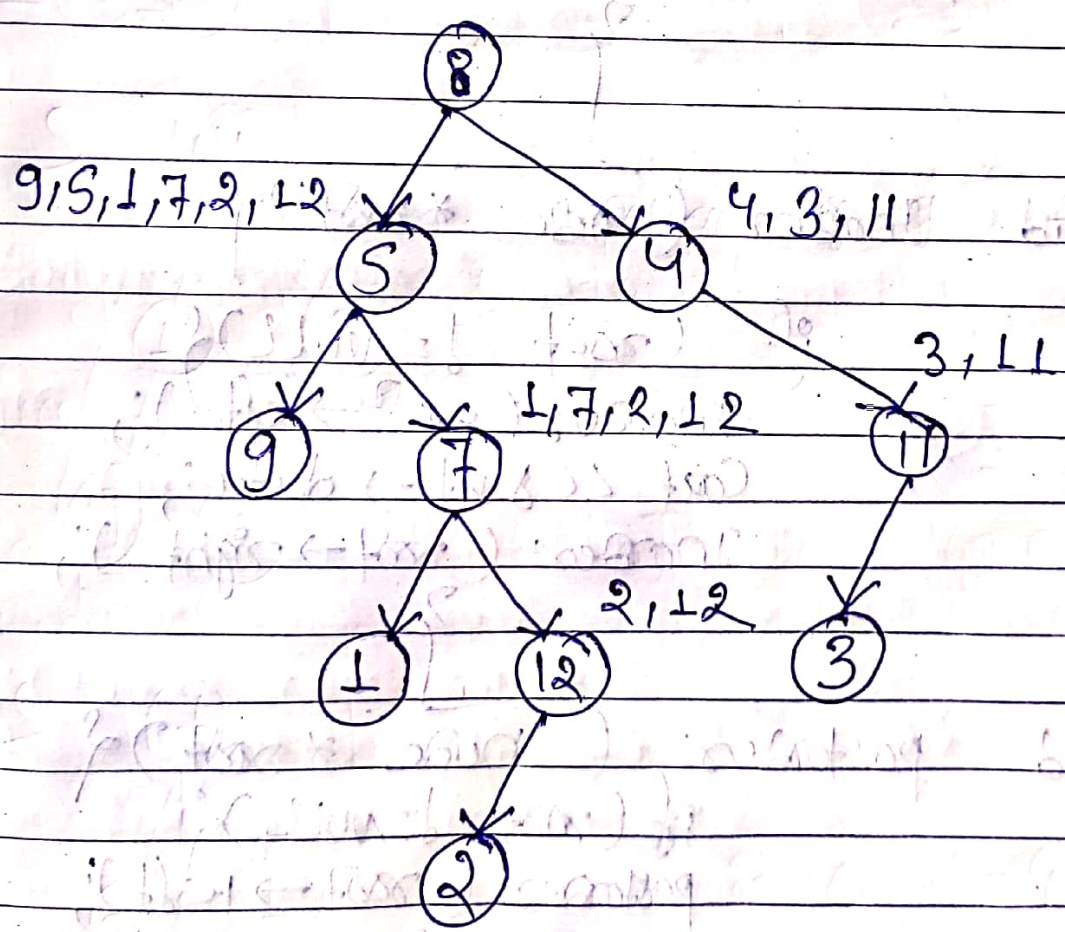
Construct a Binary tree from postorder & Inorder

Scan from right to left.

postorder:- 9, 1, 2, 12, 7, 5, 3, 11, 4, (8) (left → Right → root) ← Root

Inorder:- 9, (5), 1, (7), 2, 12, (8), (4), 3, 11 (left → Root → Right)

left | right



```
void preorder(node *root){
    if(root != NULL){
        cout << root->data << " ";
        preorder(root->left);
        preorder(root->right);
    }
}

void Inorder(node *root){
    if(root != NULL){
        Inorder(root->left);
        cout << root->data;
        Inorder(root->right);
    }
}

void postorder(node *root){
    if(root != NULL){
        postorder(root->left);
        postorder(root->right);
        cout << root->data;
    }
}
```

# Application of Tree

(i) Storing naturally hierarchical data → e.g file system.
(ii) organize data for quick search, insertion, deletion → e.g Binary search Tree.
(iii) Try → dictionary.
(iv) Network Routing algorithm.