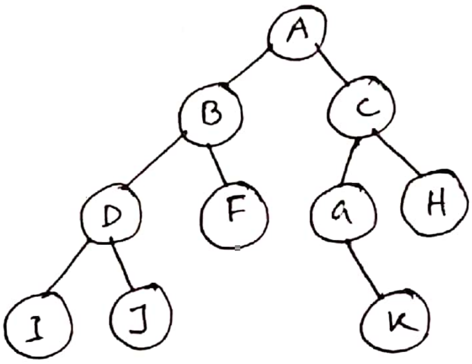
Binary Tree

Binary Tree Representations:

- Array Representation
- Linked List Representation

Consider the following binary tree:



Array Representation of Binary Tree:

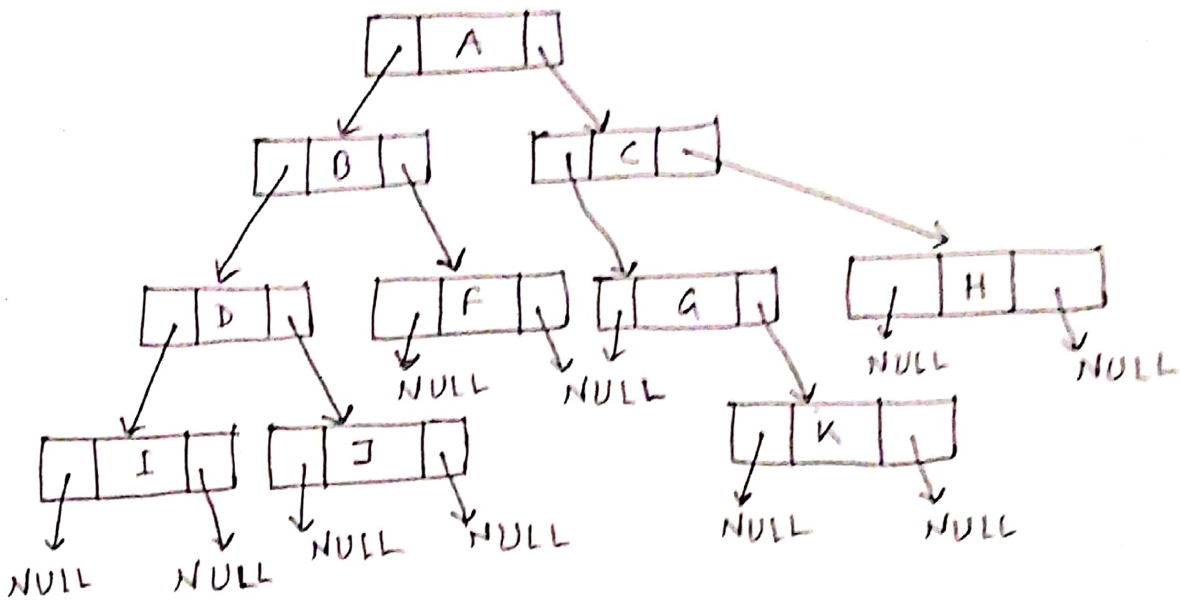
A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

index of A = 1
index of B = 2 * 1 = 2
index of C = index of B + 1 = 2 + 1 = 3

index of B = 2
index of D = 2 * 2 = 4
index of F = index of D + 1 = 4 + 1 = 5

index of C = 3
index of G = 2 * 3 = 6
index of H = index of G + 1 = 6 + 1 = 7

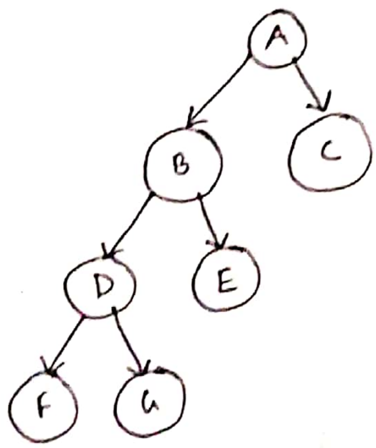
Linked list Representation of Binary Tree:



Classification of Binary Tree:

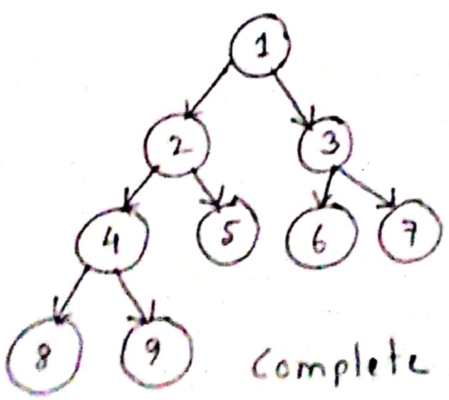
Full Binary Tree:

Every node has 2 children except the leaves

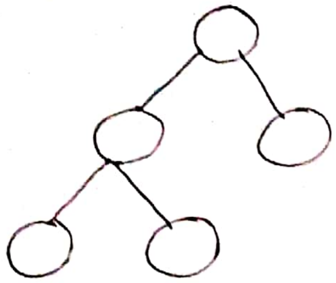


Complete Binary Tree;

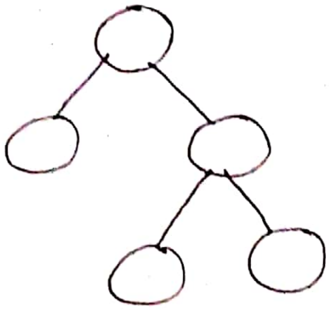
Last level may not be completely filled and the bottom level is filled from left to right.



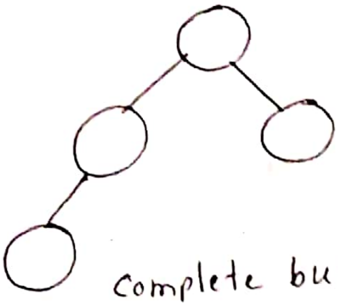
Complete Binary Tree



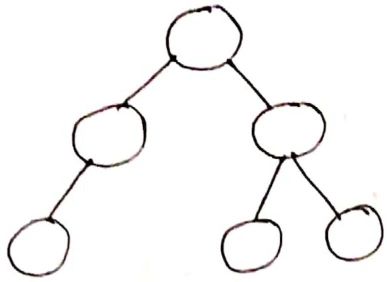
Complete and Full



Full but not Complete

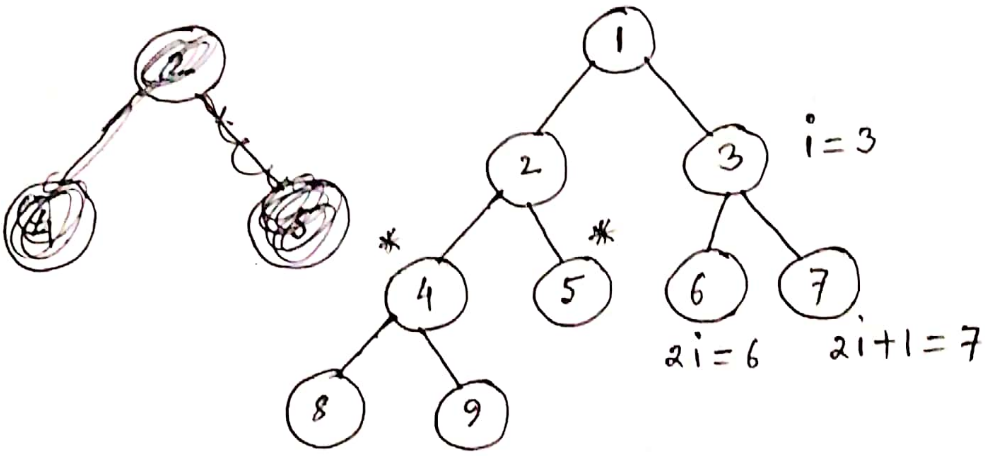


Complete but not Full



Neither Complete nor Full

A Complete binary Tree with some properties



- The parent of node i is $\lfloor \frac{i}{2} \rfloor$

  For example, $i = 4$ parent $= \lfloor \frac{4}{2} \rfloor = 2$

  $i = 5$ parent $= \lfloor \frac{5}{2} \rfloor = 2$

- The left child of node i is $2i$

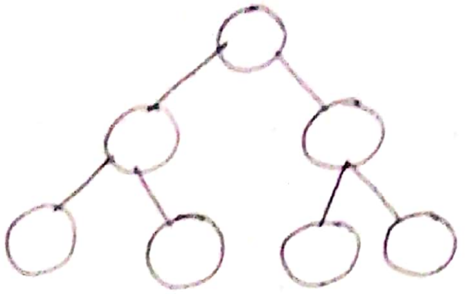
  $i = 3$ Left child $= 2 * i = 2 * 3 = 6$

- The right child of node i is $2i+1$

  $i = 3$ Right child $= 2 * i + 1 = 2 * 3 + 1 = 7$

Perfect Binary Tree:

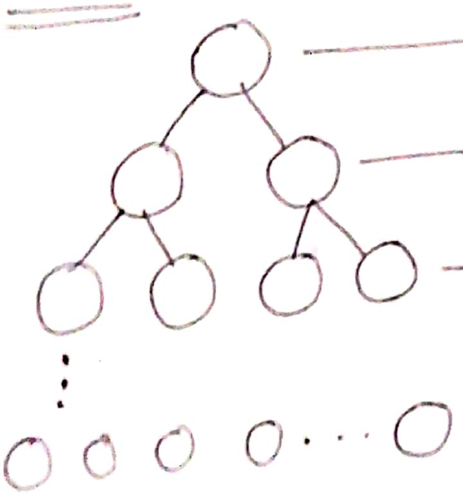
In a perfect binary tree, each leaf is at the same level and all the interior nodes have two children



Perfect Binary Tree

** Prove that the maximum number of nodes in a perfect binary tree is $2^{h+1} - 1$

Proof:



level 0 node $= 2^0 = 1$

level 1 node $= 2^1 = 2$

level 2 node $= 2^2 = 4$

level h node $= 2^h$

$$\text{Total nodes} = 1 + 2 + 2^2 + \cdots + 2^h$$

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

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

$$a + ar + ar^2 + \cdots + ar^{n-1}$$

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

Scanned with CamScanner

Height of a Perfect Binary Tree:

The number of nodes (n) for height (h) of a perfect binary tree $= 2^{h+1} - 1$

$$\therefore \quad n = 2^{h+1} - 1$$

$$\Rightarrow n = 2 \cdot 2^{h} - 1$$

$$\Rightarrow n + 1 = 2 \cdot 2^{h}$$

$$\Rightarrow 2^{h} = \frac{n+1}{2}$$

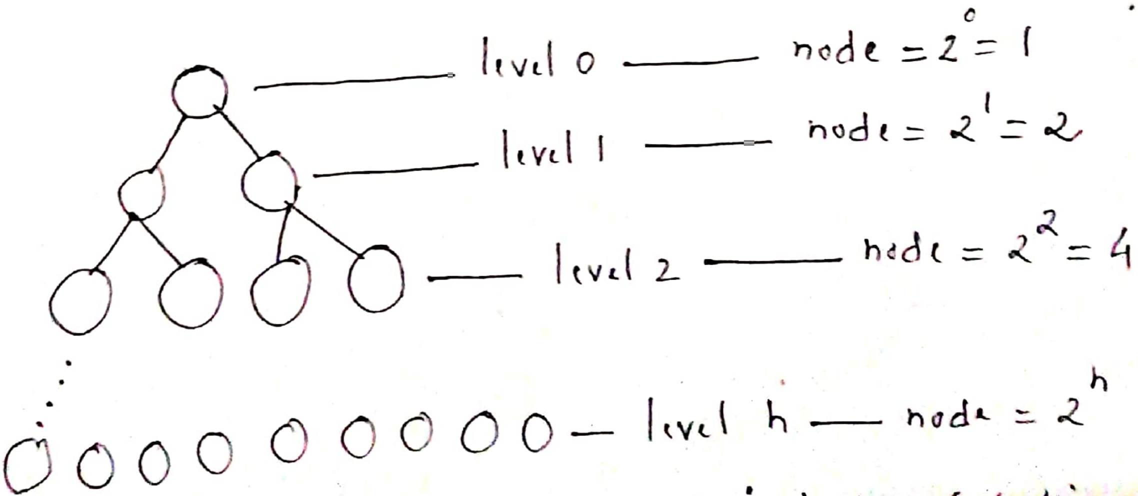
$$\Rightarrow \log_2 2^{h} = \log_2\left(\frac{n+1}{2}\right)$$

$$\Rightarrow h \cdot \log_2 2 = \log_2\left(\frac{n+1}{2}\right)$$

$$\Rightarrow h \cdot 1 = \log_2\left(\frac{n+1}{2}\right)$$

$$\Rightarrow h = \log_2\left(\frac{n+1}{2}\right)$$

Thus, the height of a perfect binary tree with n nodes $= \log_2\left(\frac{n+1}{2}\right) = \lg\left(\frac{n+1}{2}\right)$



The number of nodes at level i in a perfect binary tree $= 2^{i}$

$\therefore$ Thus, the number of leaves (nodes at level h) $= 2^h$

Thus, the total number of non-leaf nodes

$= \underbrace{\text{Total number of nodes} - \text{Total number of}}_{\text{leaves}}$

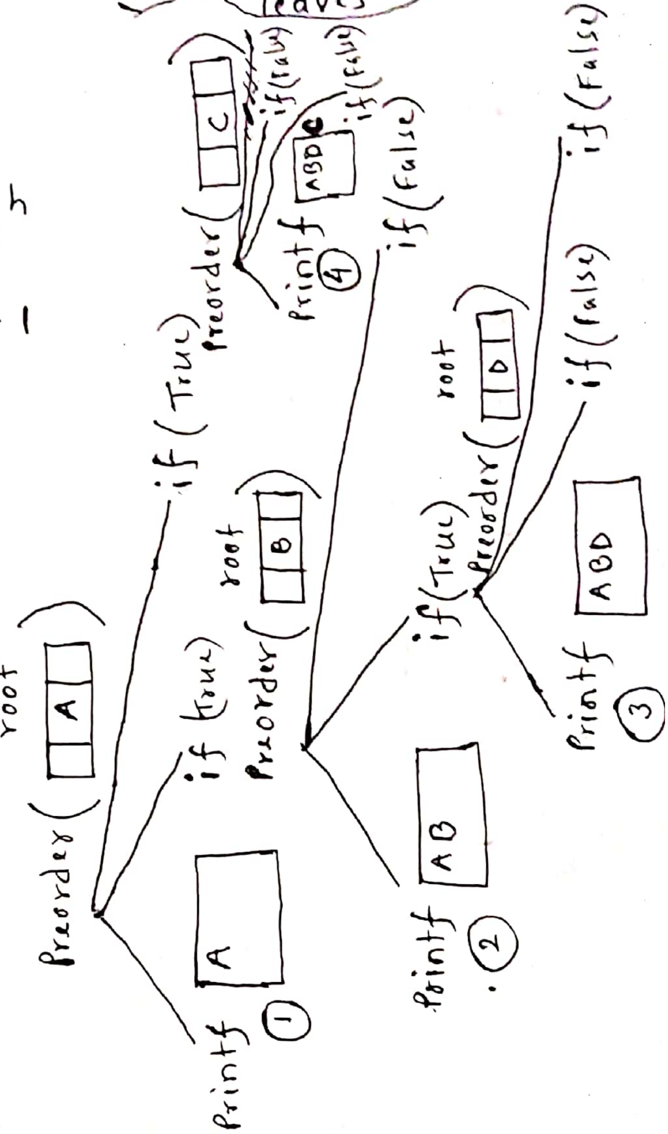
$= (2^{h+1} - 1) - 2^h$

$= 2 \cdot 2^h - 1 - 2^h$

$= (2 \cdot 2^h - 2^h) - 1$

$= 2^h (2-1) - 1$

$= 2^h - 1$

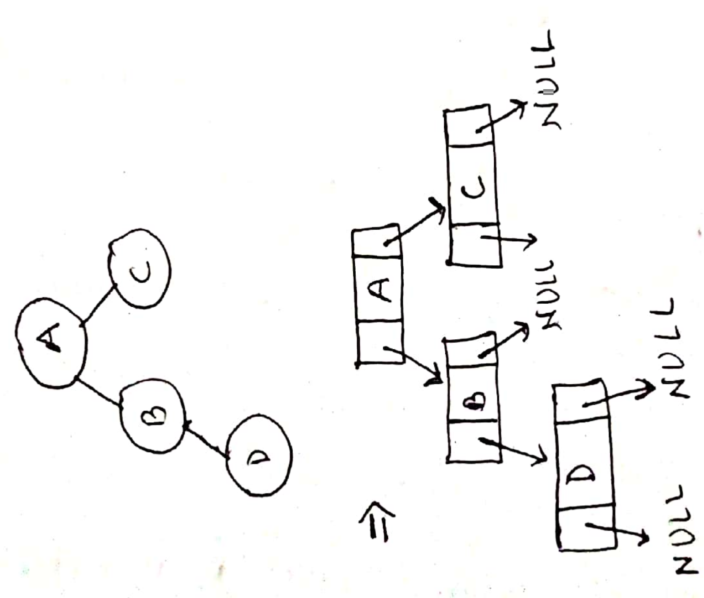


Preorder Traversal

Using linked list:

```
void Preorder(node *root){
    if(root != NULL){
        printf("%c", root->data);
        if(root->left != NULL)
            Preorder(root->left);
        if(root->right != NULL)
            Preorder(root->right);
    }
}
```

Preorder: ABDC



Draw a binary Tree using the following sequences of tree traversal Techniques

Preorder: ABDEFCG (root- left- right)

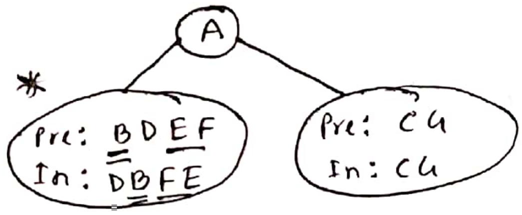
Inorder: DBFEACG (left- root- right)

Ans:

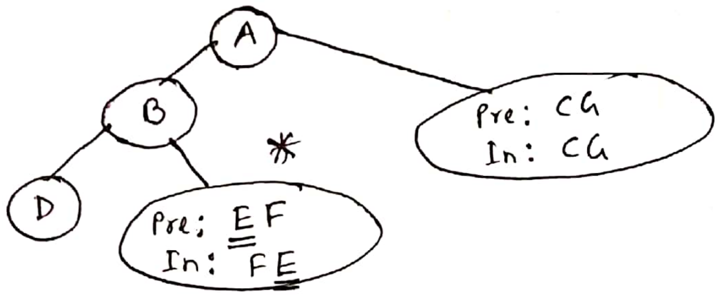
root

Tree

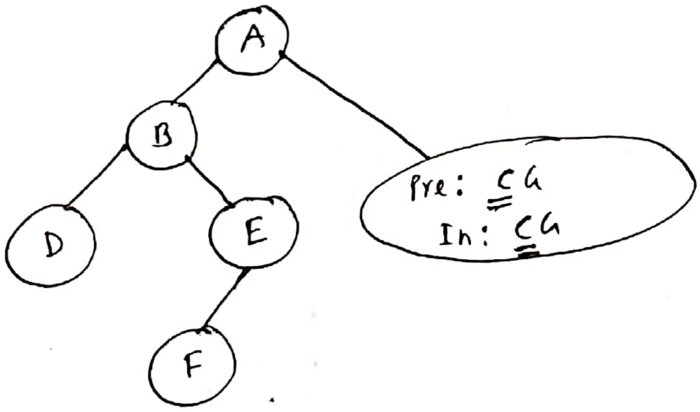
A



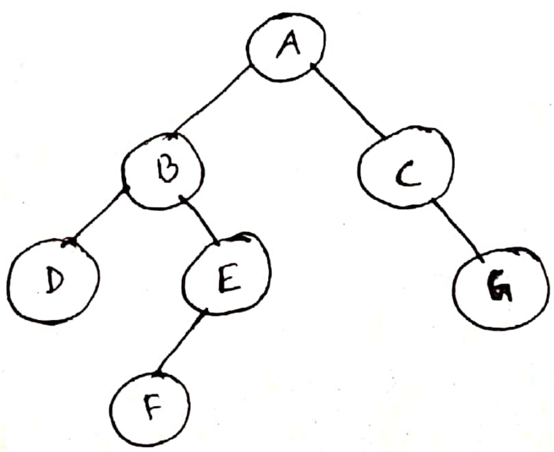
B



E



C



Non-Recursive Preorder Algorithm

stack ⇐ Empty

stack ⇐ root

while (stack ≠ Empty)

  v ⇐ stack

  Print*, v

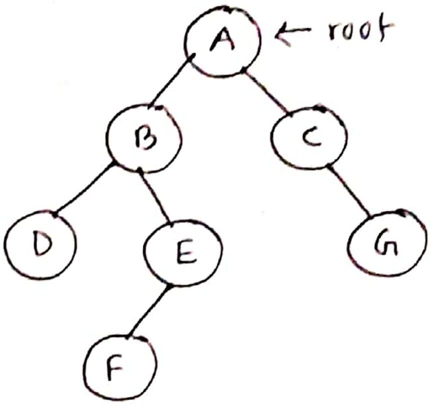
  if right(v) exists

    stack ⇐ right(v)

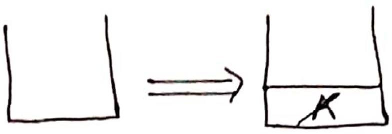
  if left(v) exists

    stack ⇐ left(v)

end while



Simulation



stack

v ⇐ stack	Print v	stack ⇐ right(v)	stack ⇐ left(v)
v = 'A'	A	C	~~B~~ / C
v = 'B'	B	E / C	~~D~~ / ~~E~~ / C
v = 'D'	D	×	×
v = 'E'	E	×	~~F~~ / ~~C~~
v = 'F'	F	×	×
v = 'C'	C	~~G~~	×
v = 'G'	G	×	×

Preorder Sequence ≡ ABDEFCG