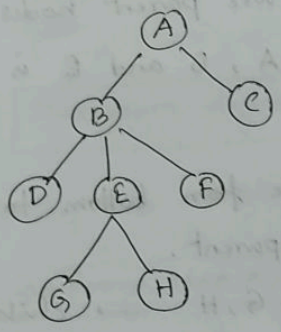
Tree

↓ is

Non-Linear Data Structure.



Node : Element of a tree.

A, B, C, D, E, F, G, H.

Root Node : Starting node of Tree.

A is a Root node.

Tree will have only one root.

Edge : Link (or) Connection between two nodes.

If a tree is having 'N' nodes. then ⇒ (N-1) Edges.

Here N = 8 nodes.

E = 8-1 = 7 Edges.

Parent : Nodes with branches from top to bottom.

A, B, E are parent nodes.

because A, B and E is having branches.

Child : Node with Edge from bottom to top (or) Branches of parent.

B, C, D, E, F, G, H ⟶ childs.

Siblings : Child nodes and same parent node.

B, C ⟶ siblings.

D, E, F ⟶ "

G, H ⟶ "

Leaf : Node without child.

C, D, F, G, H are leaf nodes.

Internal nodes : All nodes other than leaf no

Node with child nodes.

A, B, E ⟶ Internal nodes.

Degree – Number and child nodes represents degree of a node.

Degree (A) = 2

Degree (B) = 3

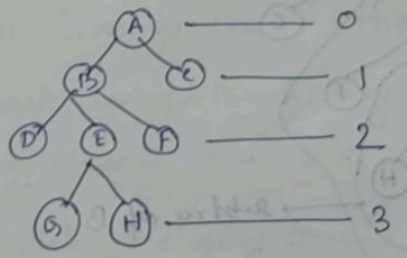
Maximum degree among all nodes = Degree of Tree

∴ Degree of Tree = 3

Level ⇒ * Every step / hierarchy in a tree is a level.

* Level starts from '0'

* For every step / hierarchy level will be incremented by 1.



∴ level of the tree = 3.

level of Root = 0.

Height ⇒ Longest path from leaf node to the node is height.

Height (B) = 2

Height (A) = 3

Depth: Longest path from root node to that node

Depth (E) = 2

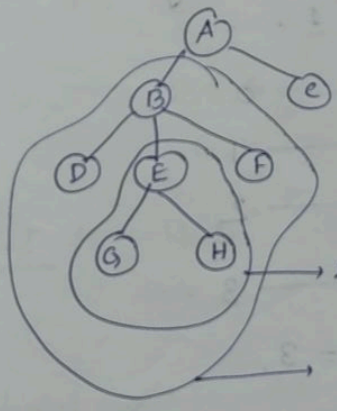
Depth (G) = 3

Depth (B) = 1.

Path : Sequence of nodes from Root to leaf.

Path (A to G) = A → B → E → G

Subtree : Node with child node forms subtree.

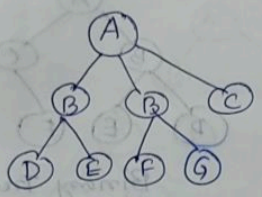


Binary Tree.

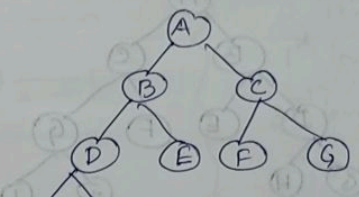
Every node in a tree should have almost 2 children

Almost 2 — 0 nodes
1 nodes
2 nodes.

0, 1, 2



Tree



Binary tree

Types of Binary Tree.

→ Full Binary Tree / strictly Binary Tree.

→ Almost Complete Binary Tree / Incomplete Binary Tree.

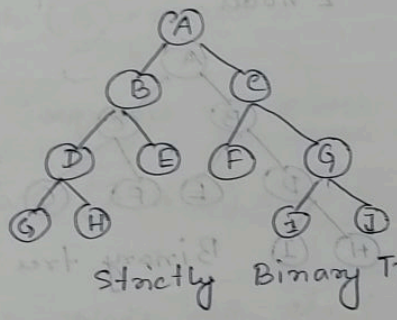
→ Complete Binary Tree / Perfect Binary Tree.

→ Left skewed Binary Tree.

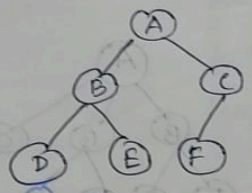
→ Right skewed Binary Tree.

Full Binary / Strictly Binary Tree.

Every node must have two children except the leaf node.



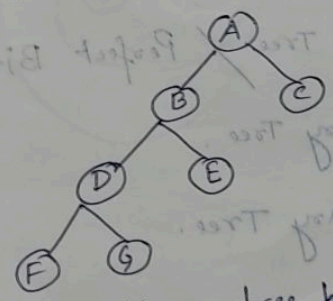
Strictly Binary Tree.



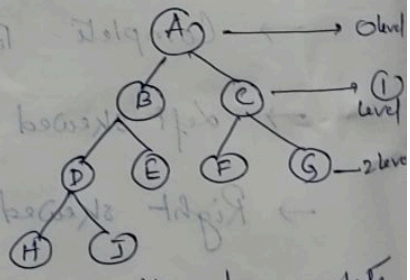
Binary Tree but not strictly binary tree. since C has only one child.

Incomplete Binary / Almost Complete Binary Tree.

Every node must have two children in all levels except in last level but filled from left to right.



Strictly binary tree but not almost complete binary tree.

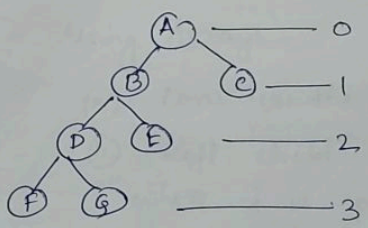


Almost complete binary tree.

Complete Binary / Perfect Binary Tree.

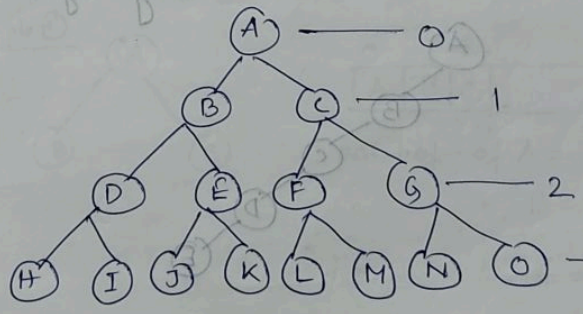
Every node must have two childrens in all the levels.

Each level there must be 2^l nodes, l = level



0^{th} level - 2^0 = 1 node
1^{th} level - 2^1 = 2 nodes
2^{nd} level - 2^2 = 4 nodes
3^{rd} level - 2^3 = 8 nodes.

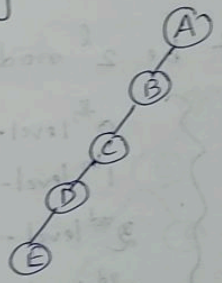
Here in 2nd level, there are only two childs, so it is not complete binary tree.



Complete binary tree since it satisfies the required no. of nodes in (2^l) each level.

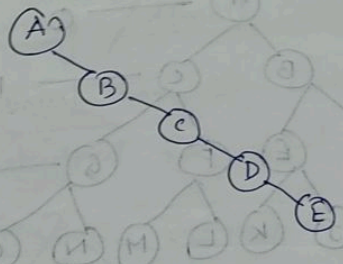
Left skewed Binary tree.

Every node should have only left children.



Right skewed Binary tree:

Every node should have only right children.



Binary Tree Representation.

(1) Using in linear or sequential way. (Arrays)

(2) using list concept.

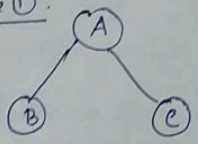
(1) Using Arrays.

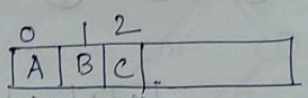
(a) Consider the root node at index 0.

(b) Left child is placed at $2i+1$ where i is position of parent.

(c) Right child is placed at $2i+2$. where is the position of parent.

Example (1).



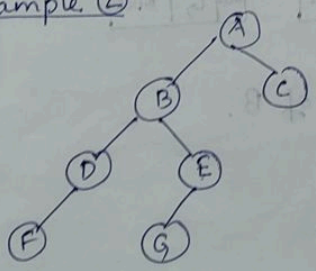


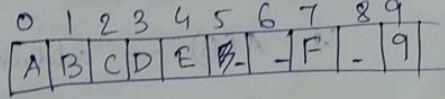
position of A = 0

" " B = $2i+1 = 2.0+1 = 1$

" " C = $2i+2 = 2.0+2 = 2$

Example. (2).





position of A = 0

" "B = $2.0+1 = 1$

" " C = $2.0+2 = 2$

" " D = $2.1+1 = 3$

" " E = $2.1+2 = 4$

" " F = $2.3+1 = 7$

" " G = $2.4+1 = 9$

(2) Using list.

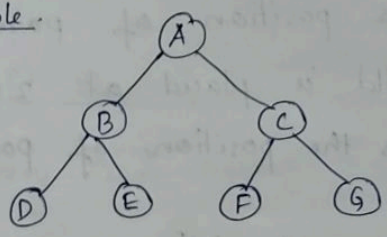
* Double linked list for representing each node of a binary tree.

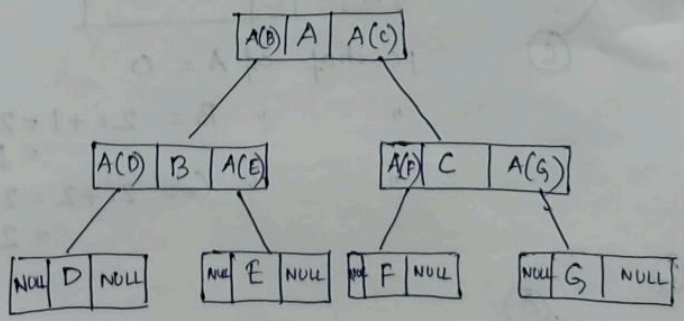
Address of previous node	data	Address of next node

i.e.

Address of left child	data	Address of right child

Example.





i.e. A(B) → address of B.

Tree Traversals

1. Inorder Traversal.

   Left child - Root node - Right child.

   (LNR)

2. Preorder traversal.

   Root node - Left child - Right child

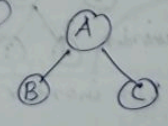
   (NLR)

3. Postorder traversal.

   Left child - Right child - Root node.

   (LRN)

Ex. 1

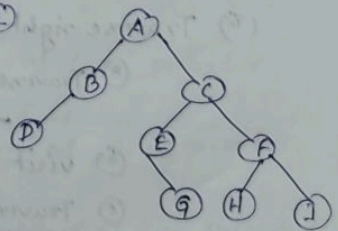


Inorder → BAC

Preorder → ABC

Postorder → BCA

Ex. 2

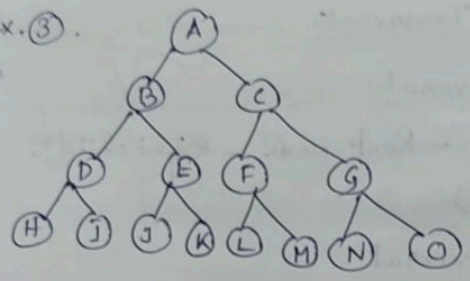


Inorder → DBAEGCHFI

Preorder → ABDCEGFHI

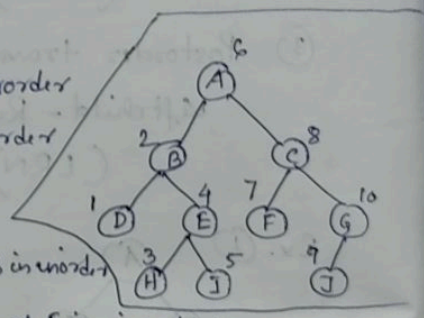
Postorder → DBGEHIFCA

Ex. (3).



—x—

Inorder Traversal:



(1) Traverse left subtree of A in inorder
  (i) Traverse left subtree of B in inorder
    - visit D
  (ii) visit B
  (iii) Traverse right subtree of B in inorder
    (a) Traverse left subtree of E in inorder
      - visit H
    (b) visit E
    (c) Traverse right subtree of E in inorder
      - visit I

(2) Visit A.

(3) Traverse right subtree of A in inorder
  (i) Traverse left subtree of C in inorder
    - visit F
  (ii) visit C
  (iii) Traverse right subtree of C in inorder
    (a) Traverse left subtree of G in inorder
      - visit J
    (b) visit G
    (c) Traverse right subtree of G in inorder
      - Empty.

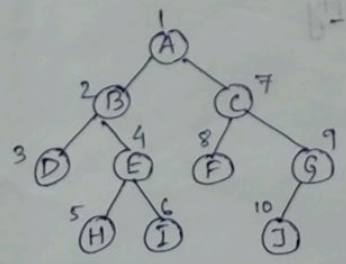
Inorder Traversal: D B H E I A F C J G

Preorder Traversal :

① Visit A

② Traverse left subtree of A in preorder
   (i) Visit B
   (ii) Traverse left subtree of B in preorder
      - visit D
   (iii) Traverse right subtree of B in preorder
      ⓐ visit E
      ⓑ Traverse left subtree of E in preorder
         - visit H
      ⓒ Traverse right subtree of E in preorder
         - visit I

③ Traverse right subtree of A in preorder.
   (i) Visit C
   (ii) Traverse left subtree of C in preorder
      - visit F
   (iii) Traverse right subtree of C in preorder
      ⓐ visit G
      ⓑ Traverse left subtree of G in preorder
         - visit J
      ⓒ Traverse right subtree of G in preorder
         - empty.



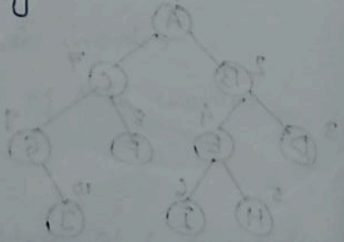
Preorder Traversal : A B D E H I C F G J

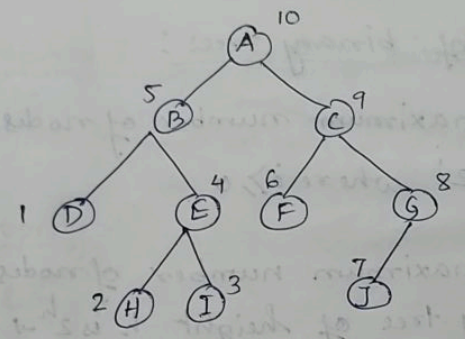
Post Order Traversal :

① Traverse left subtree of A in postorder.
  (i) Traverse left subtree of B in postorder.
    - visit D
  (ii) Traverse right subtree of B in postorder
    (a) Traverse left subtree of E in postorder.
      - visit H.
    (b) Traverse right subtree of E in postorder
      - visit I
    (c) visit E.
  (iii) visit B.

② Traverse right subtree of A in postorder.
  (i) Traverse left subtree of C in postorder
    - visit F
  (ii) Traverse right subtree of C in postorder.
    (a) Traverse left subtree of G in postorder.
      - visit J
    (b) Traverse right subtree of G in postorder
      - Empty.
    (c) visit G.
  (iii) Visit C

③ visit A.





Postorder Traversal: D H I E B F J G C A

—x—