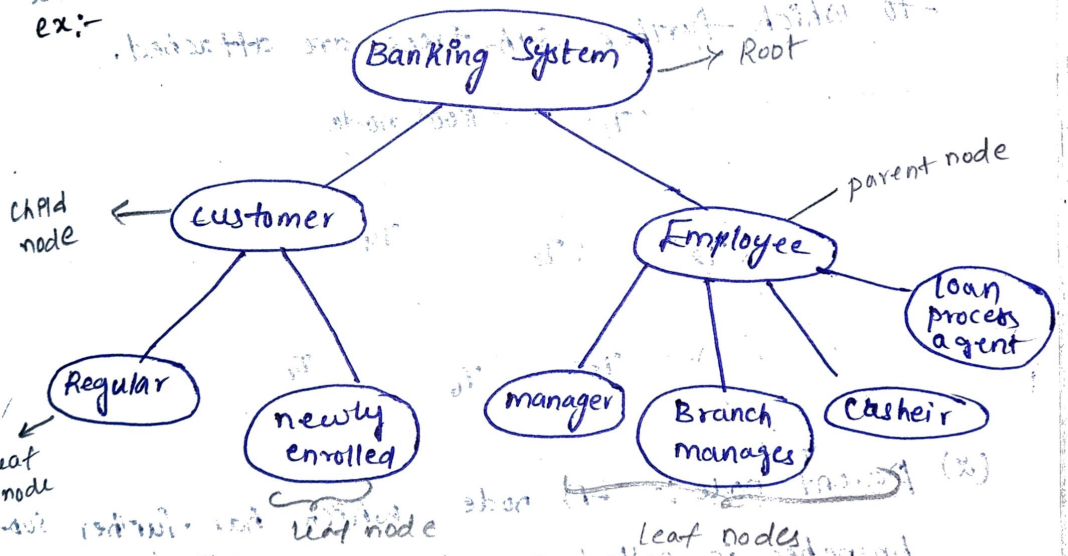
Unit-2

* The linked list usually provides more flexibility than arrays.
* But the linked list concept discuss about linear structure which is very difficult to use when they organize the data in hierarchial manner.
* Stacks and Queues Reflects some hierarchy but they are limited for one dimensional.
* To overcome this limitation we create a new data structure called as TREES.
* TREES are used to represent data objects in a hierarchial manner

ex:-



(1) TREE:-

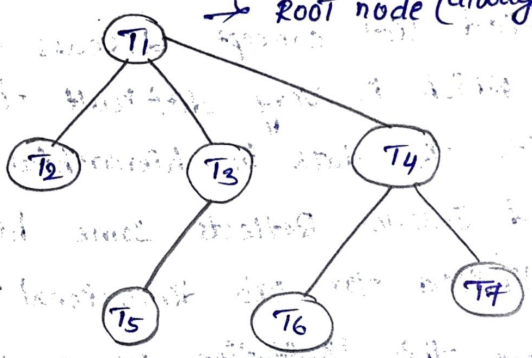
A TREE is finite set of one or more nodes such that

(1) There is a specially designed node called as Root.

(2) The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, T_2, T_3 \ldots\ldots T_n$.

where $T_1, T_2, T_3 \ldots\ldots T_n$ are called the sub-trees of the root

Example :-

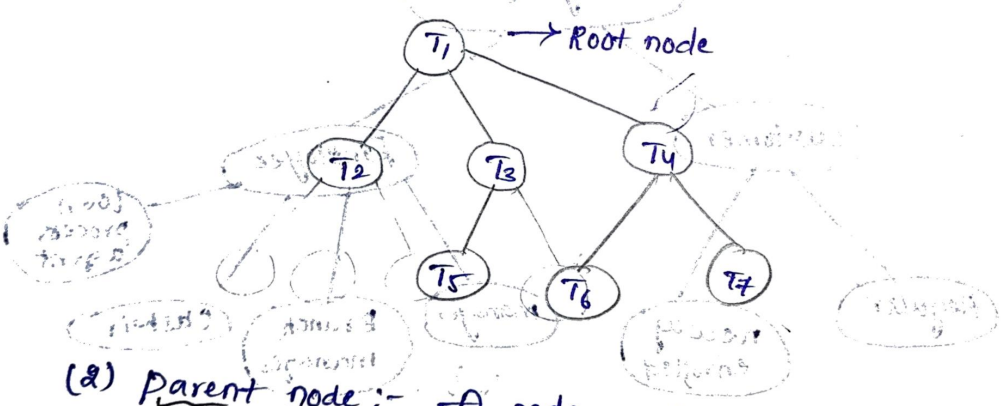


→ Root node (always unique)

TREE structure

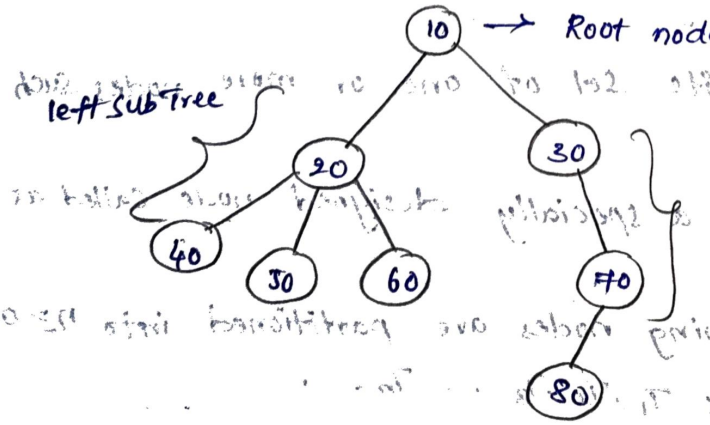
Terminology :-

(1) Root : it is a unique node in the ~~three~~ tree to which further sub-trees are attached.



→ Root node

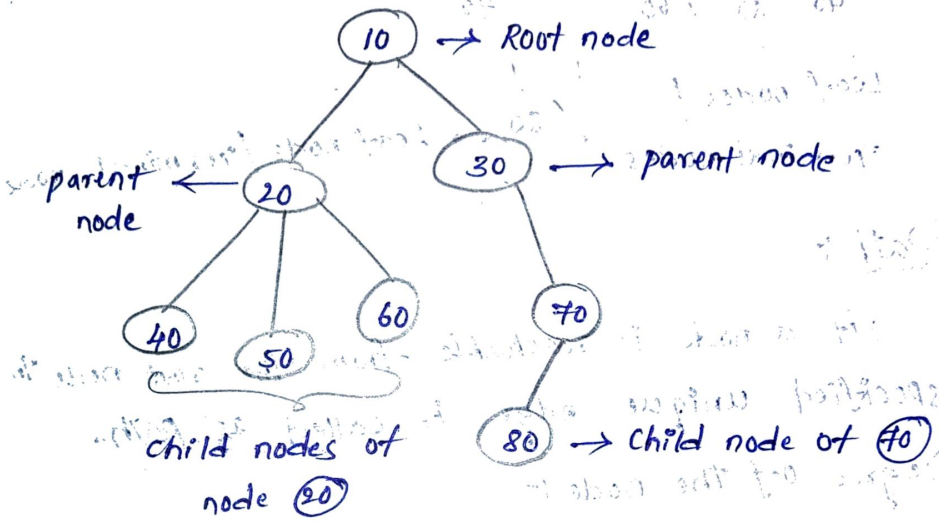
(2) Parent node :- A node which has further sub-branches is called parent node.



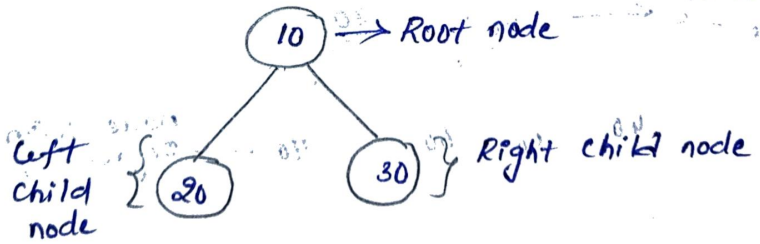
→ Root node.

(20) is the parent node of nodes 40, 50 and 60.
Similarly (70) is the parent node of node 80.

(3) child node :- The immediate successor node of parent node which is not further expanded.



→ A node which is placed on the left side and it is not further expanded is called left child node.
→ Similarly a node which is placed on the right side and not further expanded is called Right child node.



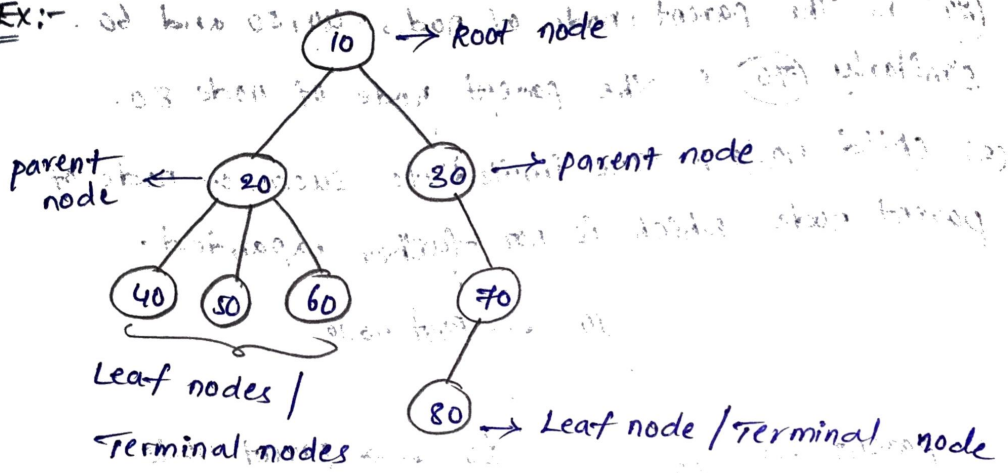
(4) Leaf node :- / Leaf node / Terminal node

In a TREE a node which does not have any child nodes is called leaf node.

Note :-

A node which is not further expanded is called as a leaf node.

Ex:-

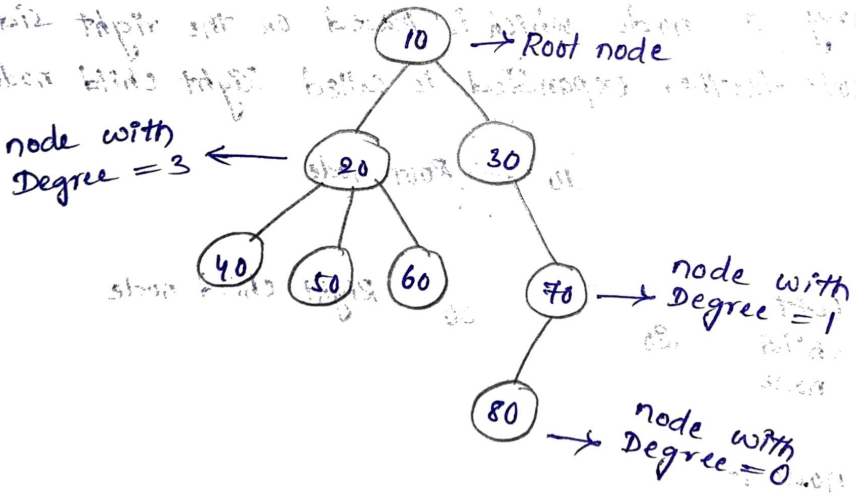


* Path :-

If a node is reachable from the root node through a specified unique edge is called as path.

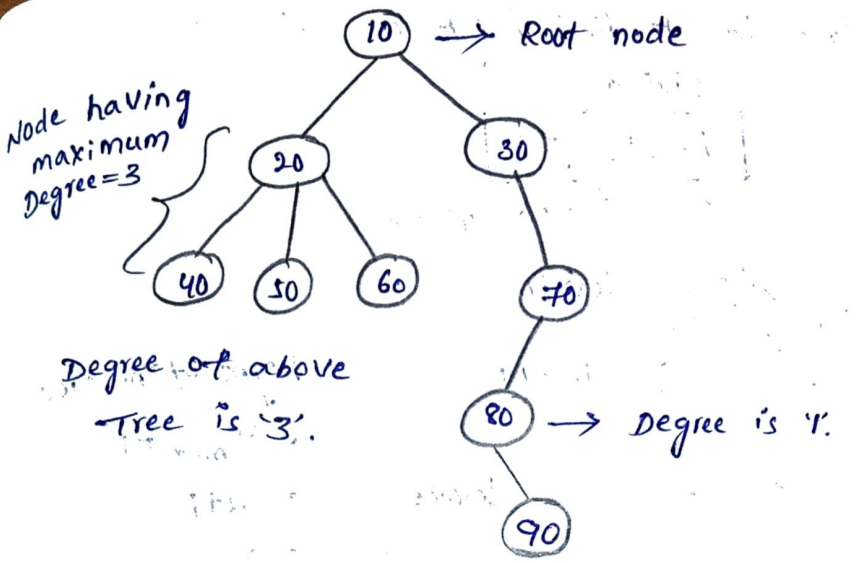
Degree of the node :-

The total no. of sub-trees attached to the node is called Degree of the node.



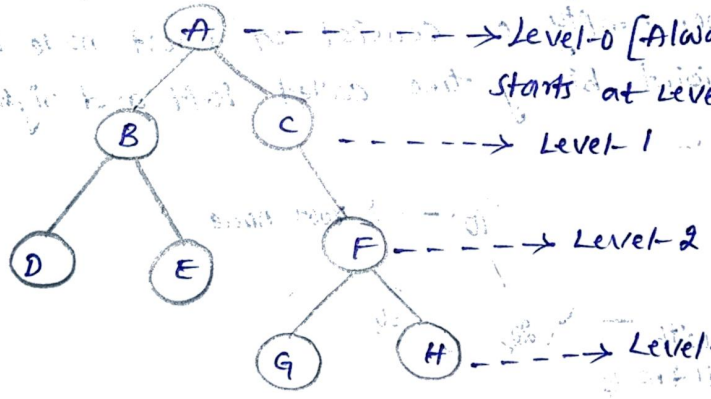
Degree of a Tree :-

The maximum degree obtained in the given TREE structure is called degree of the Tree.



Level of the Tree :-

The no. of edges along the unique path from Root node to the relavant node is called Level of a Tree.



Note :-

(1) If a node is at Level-'l' then the child node is at the level l+1.

(2) The parent node is at the level l-1.

Height of the Tree :-

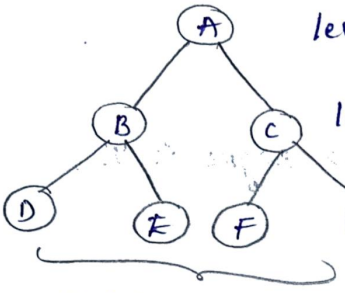
The maximum level in a tree structure is called as the Height of the Tree.

* The formula to find height of a tree is

↑ height

$$h = l_{max} + 1$$

↓ level of a tree



level 0

level 1

level 2

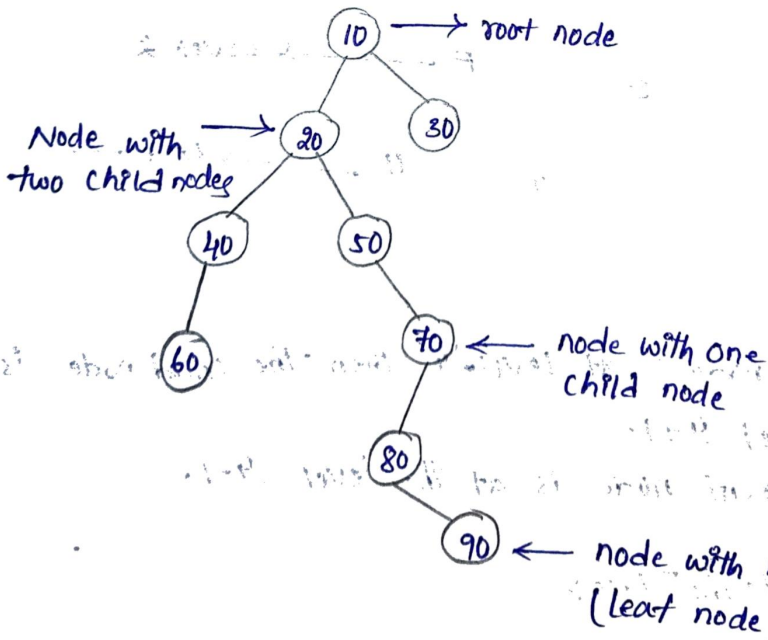
Height of the tree is 3

height of the tree
$= l_{max} + 1$
$= 2 + 1$
$= 3$

Binary Tree :-

A binary tree is a finite set of nodes which is either empty or consist of a root node which has 2 disjoint binary tree called left and right sub-trees.

Example :-



Properties of Binary Tree:-

(1) The no. of external nodes in a binary tree is given by - atleast h+1.

Where h = height of the tree

The maximum no. of external nodes in a binary tree is given by 2^h.

(2) The no. of internal nodes is atleast 'h' and atmost $2^h - 1$.

(3) The total no. of nodes in a binary tree is atleast $2^h + 1$ and atmost $2^{h+1} - 1$.

(4) A binary tree with high as 'h' and the no. of nodes as 'n' then the height of the tree can be atleast $\log n+1$ and atmost n.

(5) A binary tree with 'n' nodes has exactly n-1 edges.

*Types of Binary Tree:-

1. Expression Trees
2. Binary Search Tree
3. Threaded Binary Trees
4. Haff men Trees
5. Heap Trees
6. Height Balanced Trees
7. Decession Tree

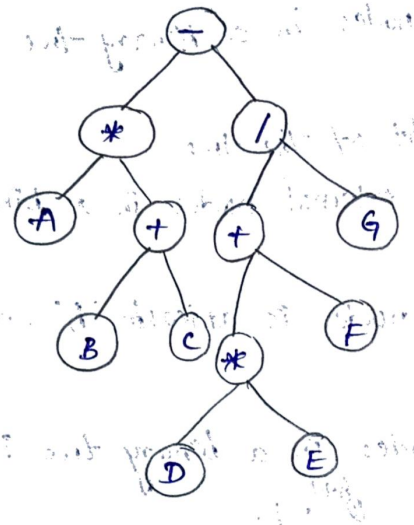
(1) Expression Tree:-

→ It is a tree which stores an arithmetic expression.

→ The leaves of the expression Tree are operands. and all internal nodes are operators.

→ An expression tree is always a Binary Tree because an arithmetic expression contains binary / unary operators.

ex:- (A+B*C) - ((D*E+F)/G).



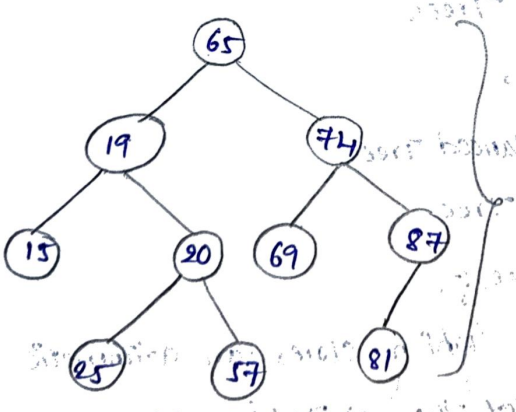
(2) Binary Search Tree

A tree in which except the root node the remaining nodes are arranged as left & Right sub trees such that

→ node less than the root is arranged at left side of the tree.

→ node greater than the root is arranged at right side of the tree.

ex:-



Binary Search Tree with numeric data.

→ A tree is also called as (BST) Binary sorted Tree.

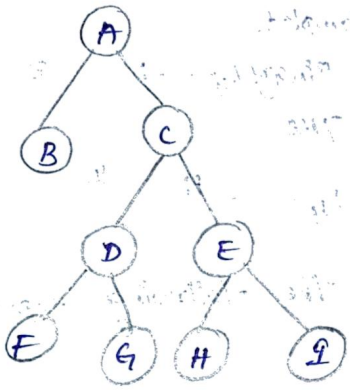
→ we use linked Data structure to represent the nodes of the Tree.

NOTE:- No two elements must have the same key.

Strictly Binary Tree:-

→ If every non-leaf node in a binary tree has non-empty left/Right sub trees. then that tree is called as Strict Binary Tree.

ex:-



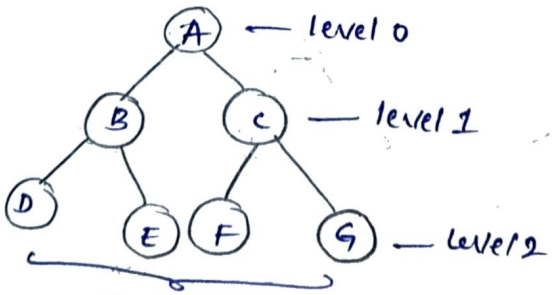
NOTE:-

A strict Binary Tree with 'n' leaves always has $2n-1$ nodes

Complete Binary Tree:-

A binary Tree in which starting from Root node all the levels must be exactly filled with two nodes each, and all the leaf nodes must be at same level. such tree is called as Complete Binary Tree.

ex:-



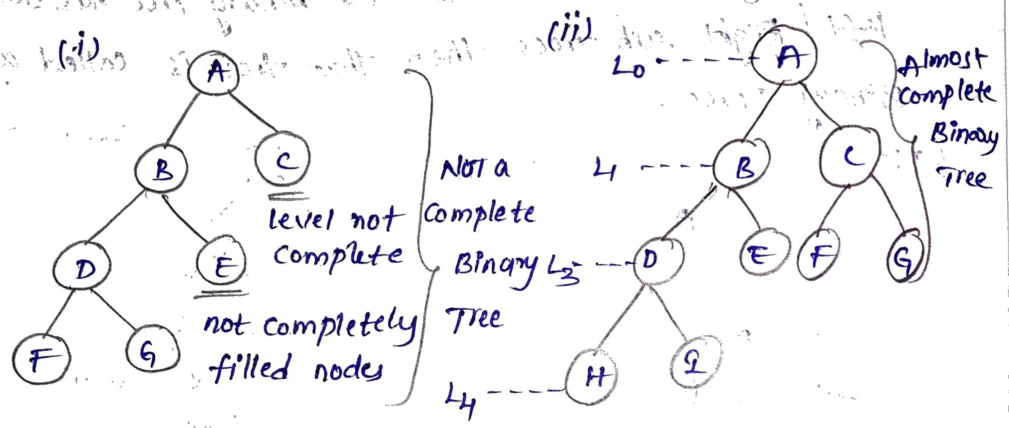
Complete Binary Tree.

NOTE:- Total number of nodes in a complete Binary Tree is given by $\sum_{j=0}^{d} 2^j$ where $d \rightarrow$ level of the Tree.

Almost Complete Binary Tree:-

A binary Tree in which the nodes are filled in a systematic order (level by level) that tree is called Almost Binary Tree.

Ex:-



Draw a binary Tree for the following expression.

(i) (A+B) *(C+D)

(ii) (a-b) / ((C*d)+e)

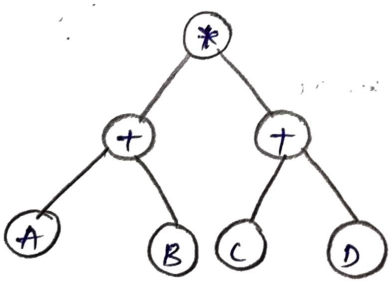
(iii) / ((a*x+b)* x+e) * x+f))

→ The root node is an operator.

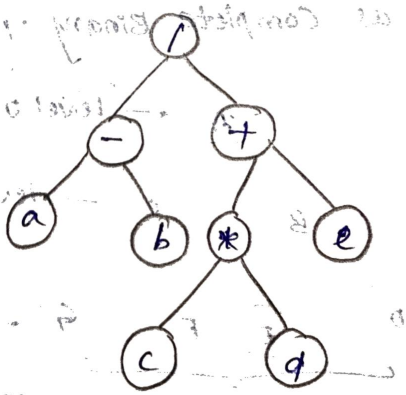
→ The left and right childs are operands.

→ In case of unary operations if left child is not present and right child is an operand.

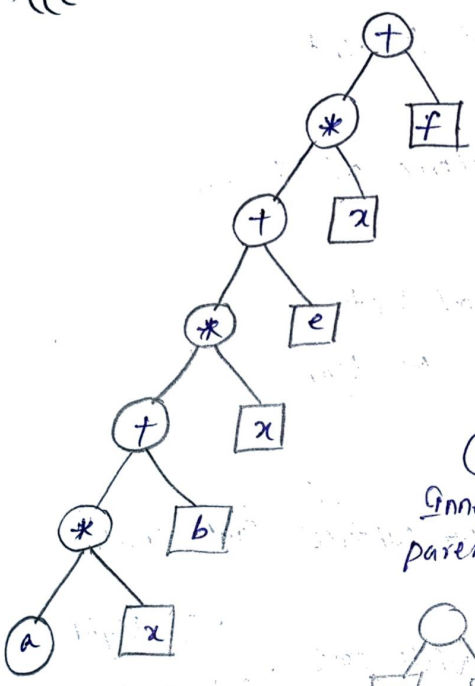
(i) (A+B) * (C+D)



(ii) (a-b) / ((C*d)+e)



(iii) $(((a*x+b)*x+e)*x+f)$

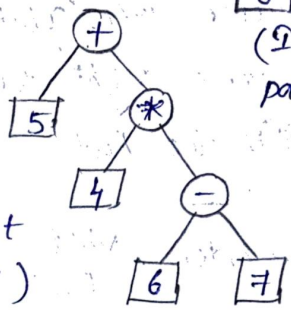


(iv) $(5+4*(6-7))/(5+8)$

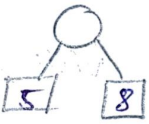
(i)

(Inner most parenthesis)

(ii)

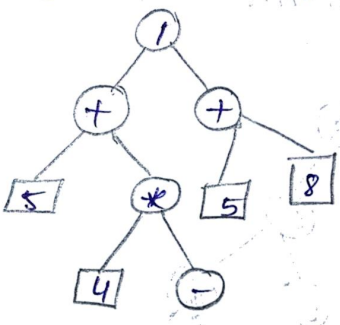


(next Inner most parenthesis)



(now consider outer parenthesis)

merge all the content & arrange in an order given the answer.



Binary Tree Traversals :-

(1) Pre Order Traversal (NLR Traversal) — also known as
(2) In Order Traversal (LNR Traversal) — L: Left, N: Node, R: Right
(3) Post order Traversal (LRN Traversal) — N: Root Node

(1) Pre order Traversal.

(i) Visit the Root
(ii) Traverse the left - subtree of the root
(iii) Traverse the Right - subtree of the root

(2) In order Traversal

(i) Traverse left-subtree of the root

(ii) Visit the root

(iii) Traverse the Right-subtree of the Root.

(3) post order Traversal

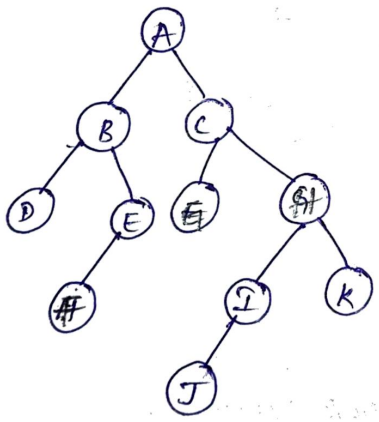
(i) Traverse left subtree of Root

(ii) Traverse Right subtree of Root

(iii) Visit the Root.

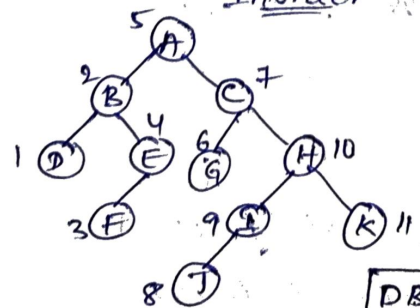
pre order	Visit Root	left subtree	Right subtree
Inorder	Visit left subtree	Visit Root	Vist Right Sub-tree
post order	Visit left Sub-tree	Visit Right Subtree	Visit Root

ex:-



*Write the pre-order, In-order, post-order for the Binary Tree.

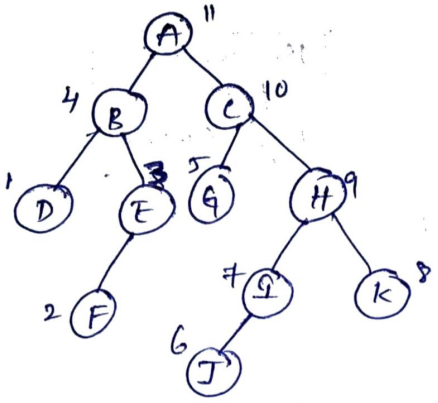
(ii) Inorder



left sub tree
root
right subtree

D B F E A G C J I H K

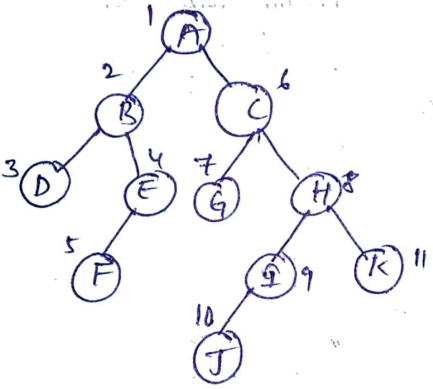
(iii) post order.



left subTree
Right subTree
ROOT

DFEBGJIKHCA

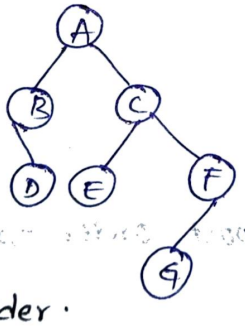
(ii) pre-order



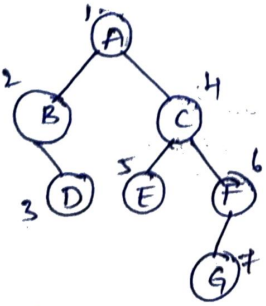
ROOT
left sub-tree
Right sub tree

ABCDEFGHIJK

ex2 :-



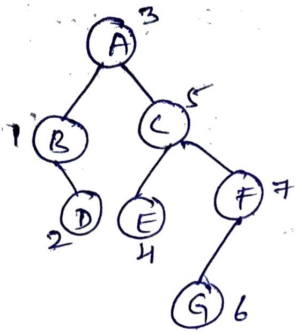
(i) Pre-order.



Root
left sub-tree
Right sub-tree

ABCDEFG

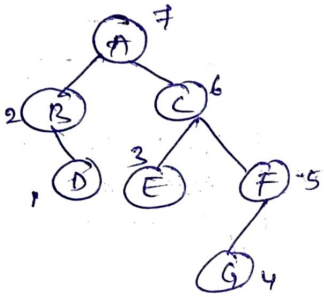
(ii) In-order:-



left subtree
Root
Right subtree

BDAECGF

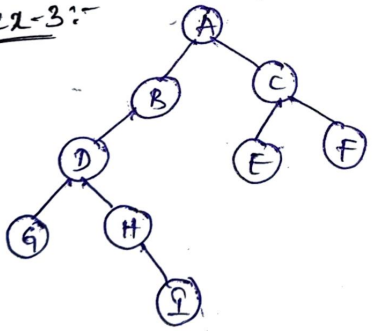
(iii) post order



left subtree
Right sub-tree
Root

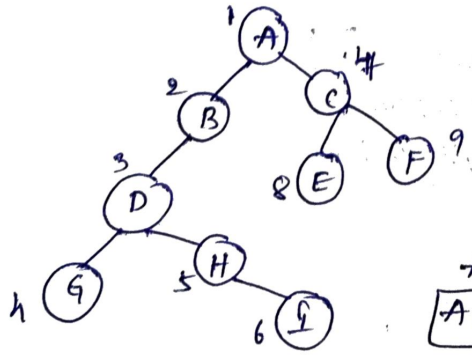
DBEGFCA

ex-3:-



* Write the pre-order, In-order, post-order for the Binary Tree.

(i) pre-order

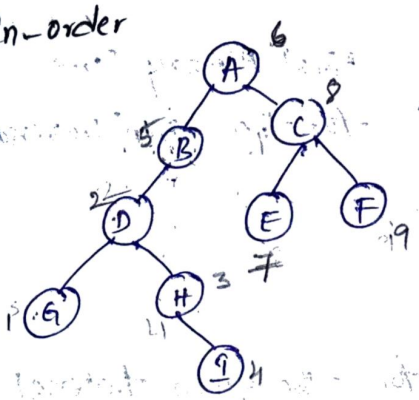


Root
left Sub-tree
Right sub-tree

~~ABEDC~~

ABDGHICEF

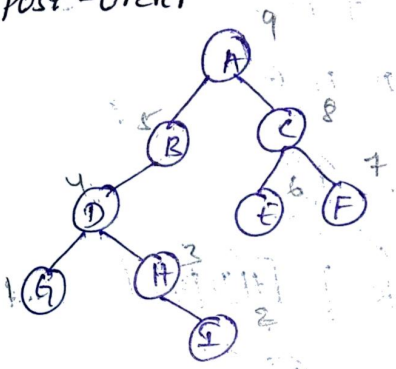
(ii) In-order



left subtree
Root
Right subtree

GDHIBAECF

(iii) post-order



left subtree
Right subtree
Root

GIHDBEFCA

* Creation of a Binary Tree using Tree Traversal Techniques

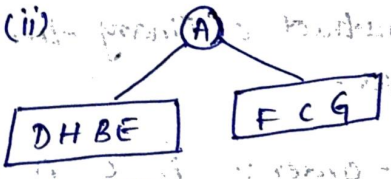
(1) Construction of a Binary Tree when pre-order & In-order techniques are given

Pre order :- A B D H E C F G

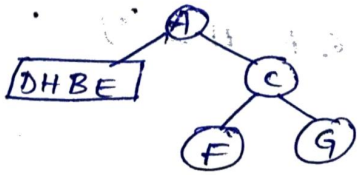
In-order :- D H B E A F C G

Ans :- (i) (A) Root from given pre-order Traversal Data.

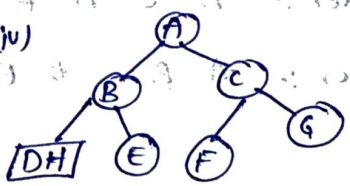
(ii)



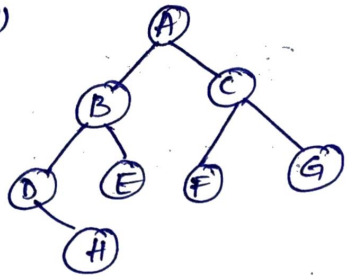
(iii)



(iv)



(v)



Final Binary Tree for the given Traversals.

(2) Construct a Binary Tree for the given traversal pre-order & In-order techniques are given.

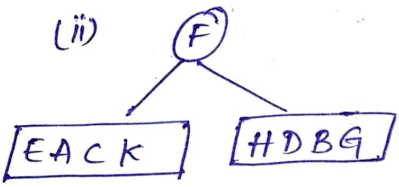
pre-order :- F A E K C D H G B

In order :- E A C K F H D B G

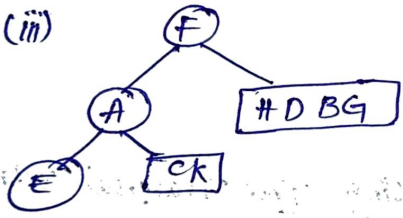
Ans :-

(i) (F) Root from given pre-order Traversal Data.

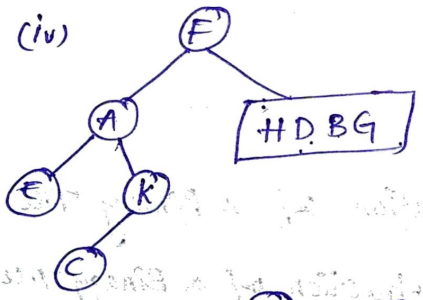
(ii)



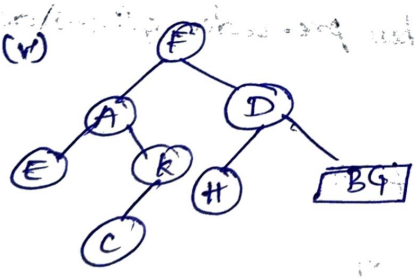
(iii)



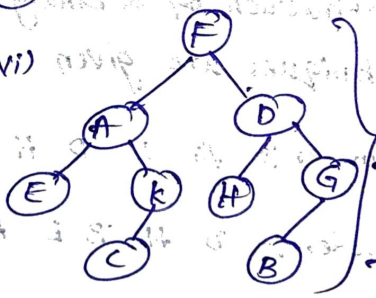
(iv)



(v)



(vi)



Final Binary Tree for the given Traversal.

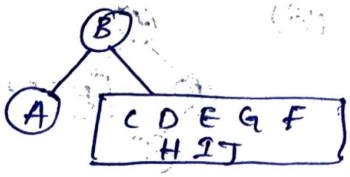
(3) Construct a Binary tree for the given Traversal techniques.

pre-order :- B C A E G D H F I J

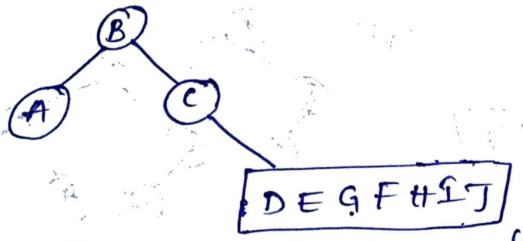
In-order :- A B C D E G F H I J

Ans:- (i) (B) Root from the given pre-order Traversal Data.

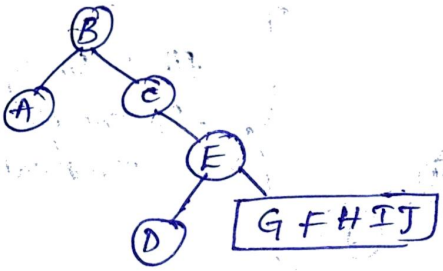
(ii)



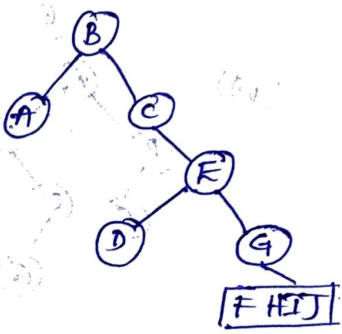
(iii)



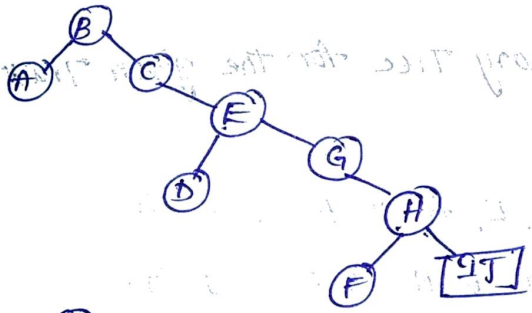
(iv)



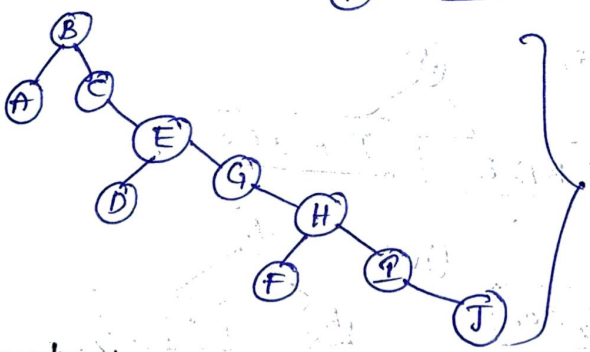
(v)



(vi)



(vii)



Final Binary Tree for the given Traversal.

(4) construct a Binary Tree for the given traversal techniques.

In-order:- B C A E G D H F I J

pre-order:- A B C D E G F H I J

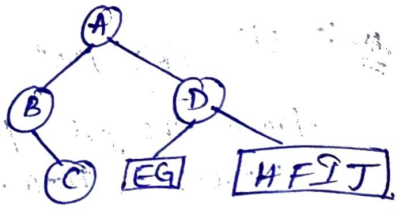
Ans:-

(A) Root from given the traversal data

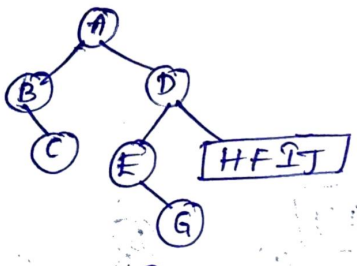
(ii)

(iii)

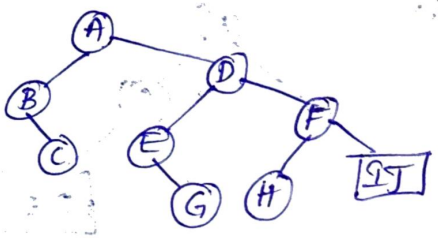
(iv)



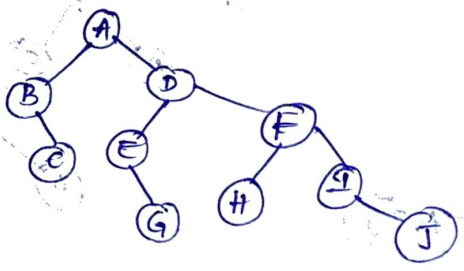
(v)



(vi)



(vii)



Final Binary Tree for the given Traversal.

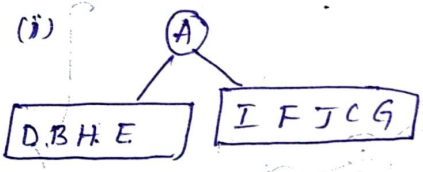
(5) Construct a Binary Tree for the given Traversal technique.

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

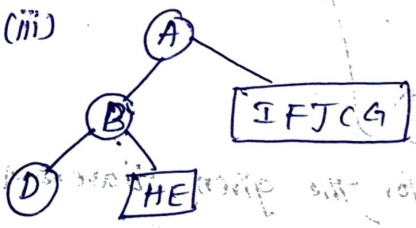
preorder :- A B D E H C F I J G.

(i) A Root

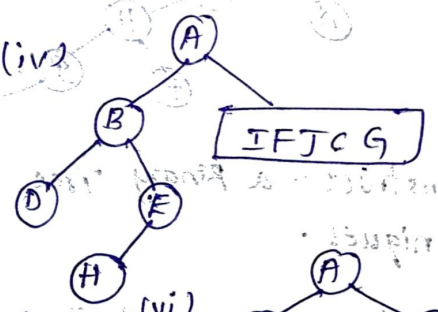
(ii)



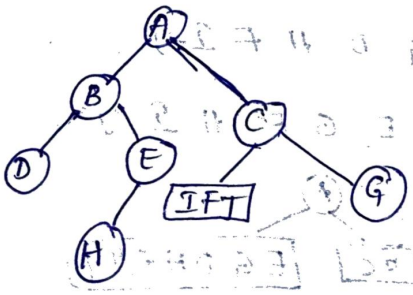
(iii)



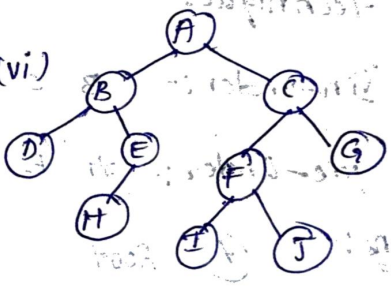
(iv)



(v)



(vi)



(6) Construct a Binary Tree for the given Traversal.

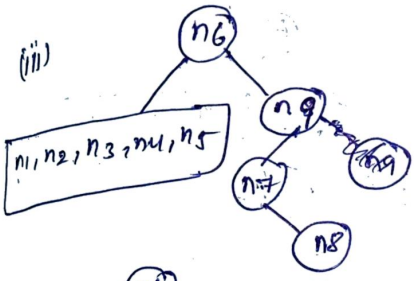
Inorder :- $n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9$

post order :- $n_1, n_3, n_5, n_4, n_2, n_8, n_7, n_9, n_6$.

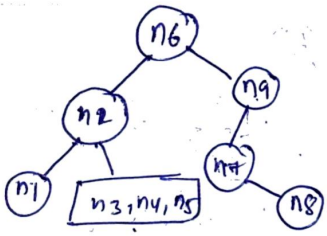
(i) n_6 root

(ii)

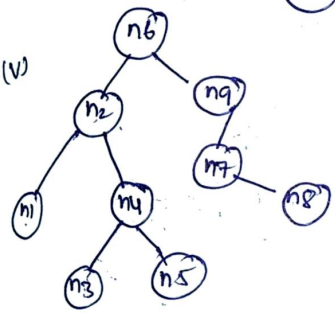
(iii)



(iv)



(v)



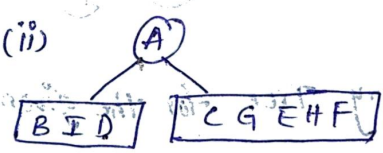
(7) Construct a Binary Tree for the given Traversal.

Inorder :- B I D A C G E H F

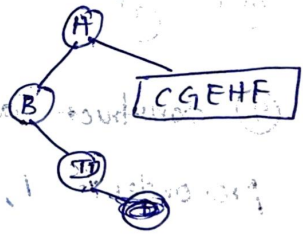
Post order :- I D B G C H F E A

(i) A root

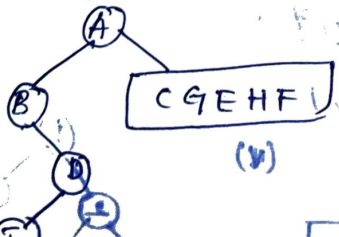
(ii)



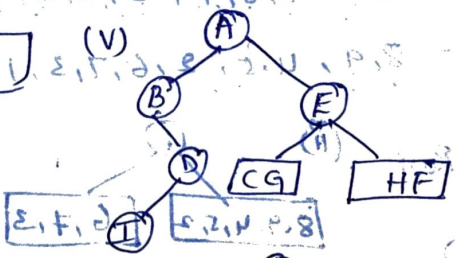
(iii)



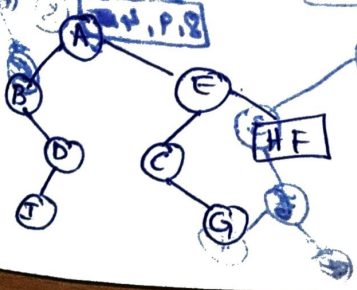
(iv)



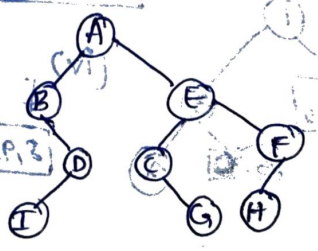
(v)



(vi)



(vii)



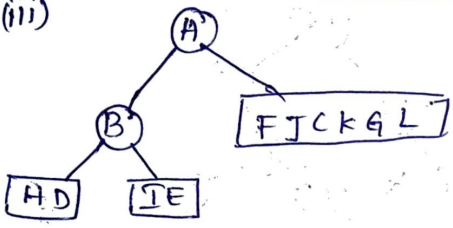
⑧ construct a binary Tree for the given Traversal.

Inorder :- H D B I E A F J C K G L

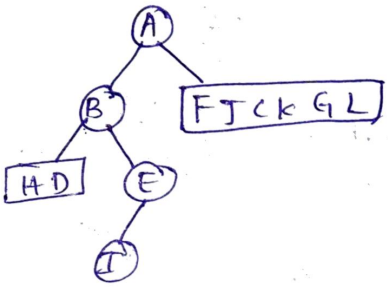
post order :- H D I E B J F K L G C A

(i) (A) root (ii)

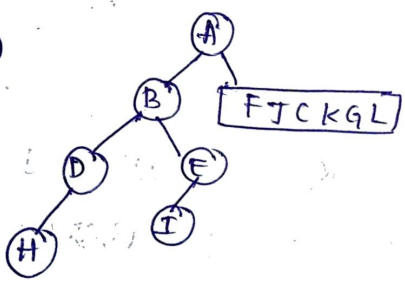
(iii)



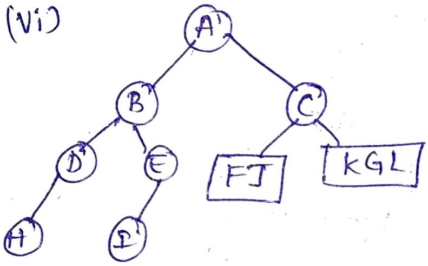
(iv)



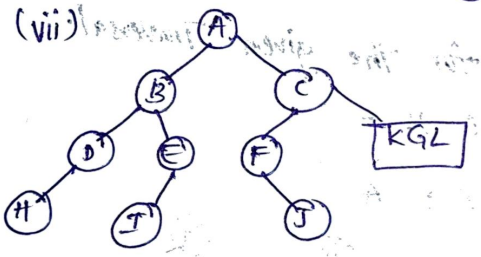
(v)



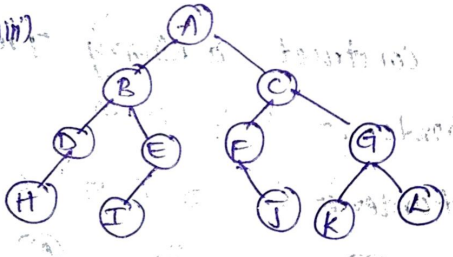
(vi)



(vii)



(viii)

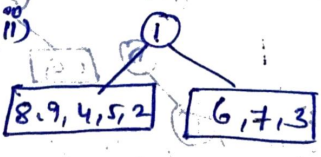


⑨ construct a binary Tree for the given Traversal.

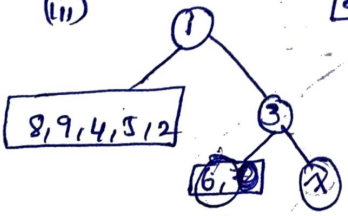
pre-order :- 1, 2, 4, 8, 9, 5, 3, 6, 7.

post order :- 8, 9, 4, 5, 2, 6, 7, 3, 1

(i) (1) Root (ii)

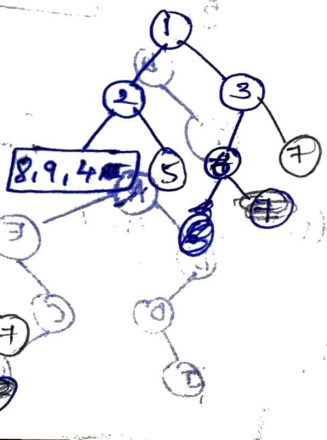


(iii)

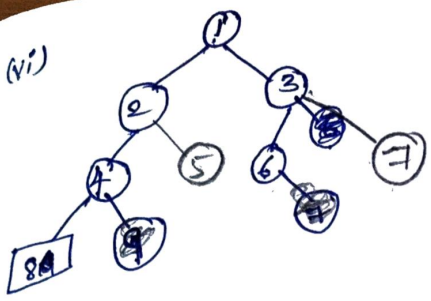


(iv)

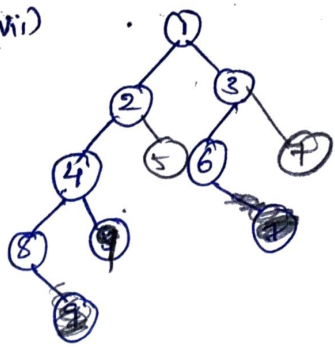
(v)



(vi)



(vii)



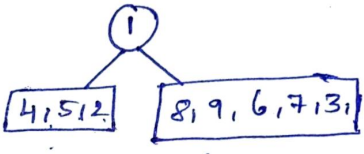
RD Left right

(10) pre order :- 1, 2, 4, 5, 3, 6, 8, 9, 7

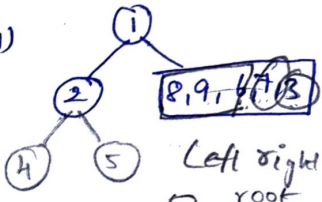
post Order :- 4, 5, 2, 8, 9, 6, 7, 3, 1

(i) (1) root

(ii)

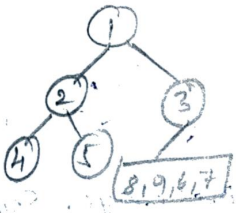


(iii)

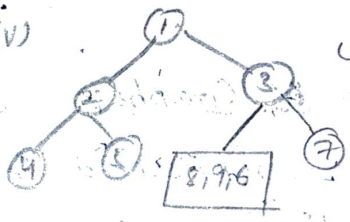


Left right root

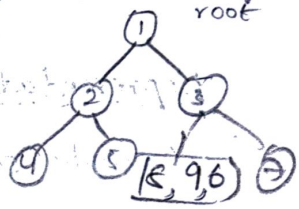
(iv)



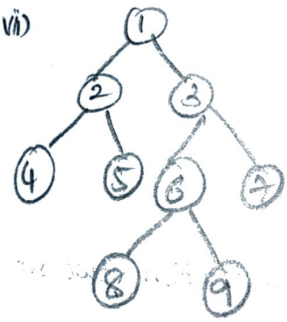
(v)



(vi)



(vii)

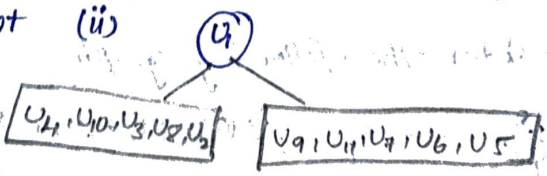


(11) pre-order :- $U_1, U_2, U_3, U_4, U_{10}, U_8, U_5, U_9, U_6, U_{11}, U_7$

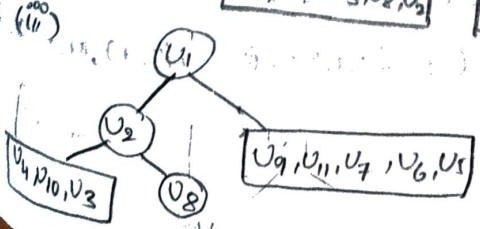
post order :- $U_4, U_{10}, U_3, U_8, U_2, U_9, U_{11}, U_7, U_6, U_5, U_1$.

(i) (U_1) root

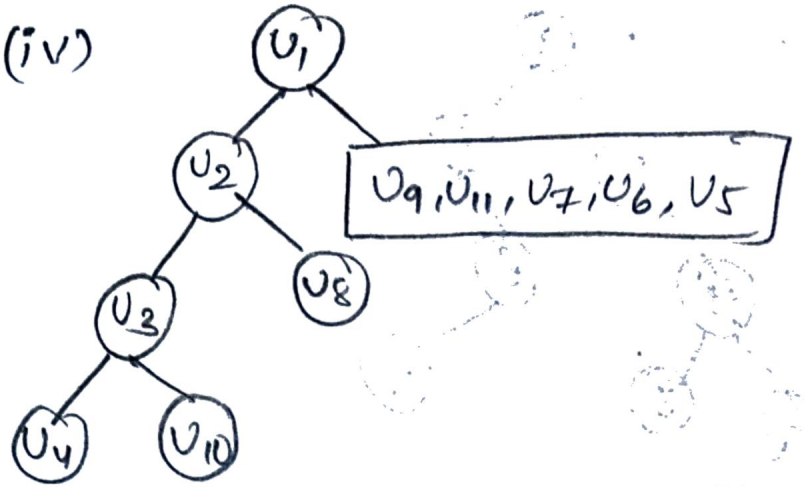
(ii)



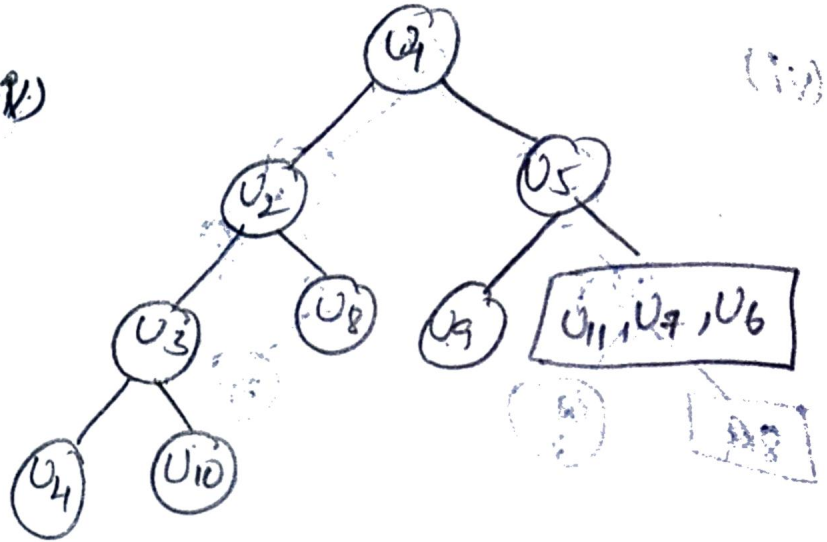
(iii)



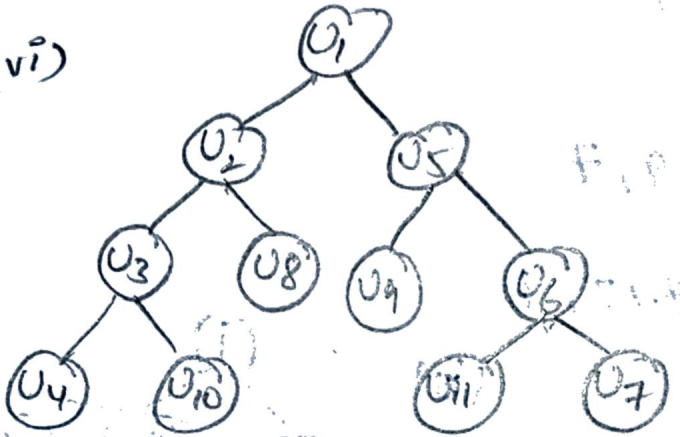
(iv)


U_1
U_2
U_3
U_8
U_4
U_{10}
$U_9, U_{11}, U_7, U_6, U_5$

(v)


U_4
U_2
U_5
U_3
U_8
U_9
U_{11}, U_7, U_6
U_4
U_{10}

(vi)


U_1
U_2
U_5
U_3
U_8
U_9
U_6
U_4
U_{10}
U_{11}
U_7