5. Tree

Binary tree

A Binary tree T is defined as a finite set of element called nodes. Such that :

1] T is empty. (Null tree or empty tree

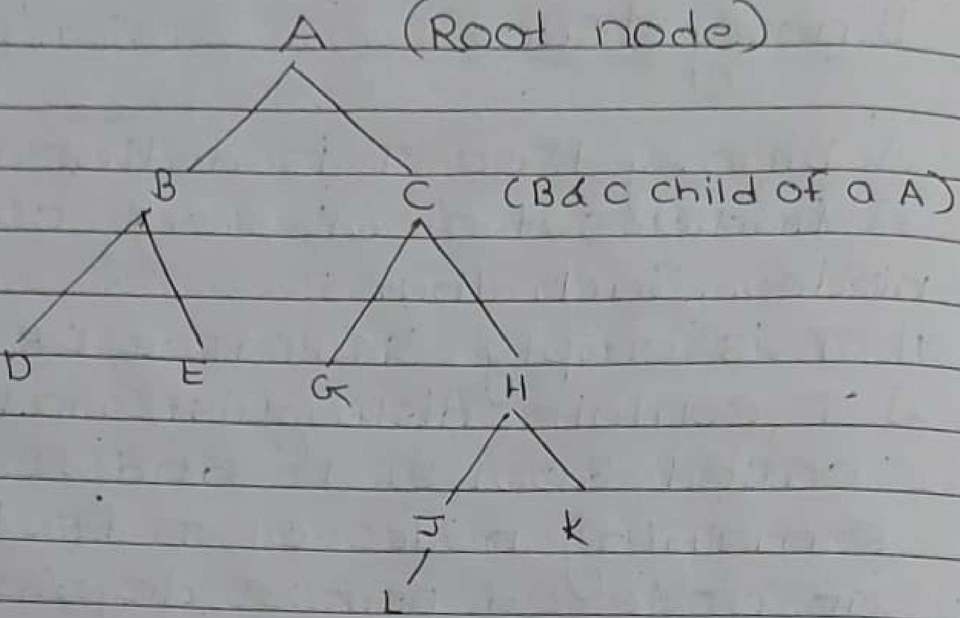
2] T contain distinguish node R called Root of T and the remaining nodes of T ~~fr~~ form an ordered pair of disjoint binary tree T_1 and T_2

If T contain a root R then the two tree T_1 and T_2 are called left and right subtrees of R.

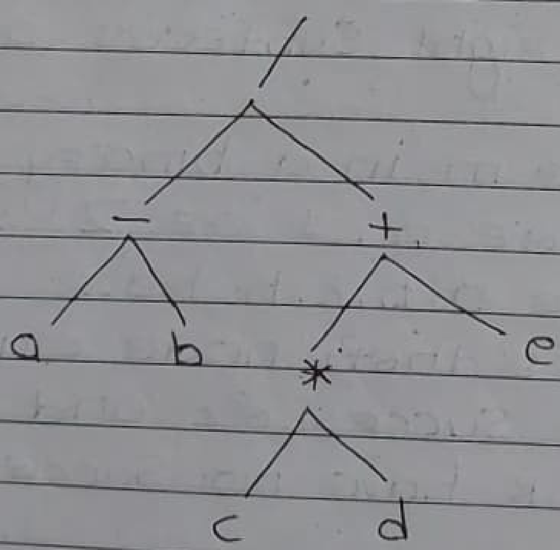
Any node m in a binary tree T has either 0, 1 or 2 succeser. The node a, b, c, h have two successer then node e and j has only one successer and the nodes d, f, g, i and k have no successer.

The nodes with no successer are called as terminal nodes.

Terminal node is also called as leaf node.



Q Consider, an algebric expression
$E = (a-b)/((c*d)+e)$



- edges - − to a, − to b, / to +
- path = (/ − + − *), (/ − + − *)
- branch - (/ − + − * − c)

Page No. ______
Date : __/__/__

• Level number : (Start from 0)

Each node in a binary tree T is assign a level number as follows.
1) The root R of the tree assign level no. zero.
2) Every other node is assign a level no. which is ~~one~~ 1 more than the level number of it's parent.
3) Node with same level number are said to belong to the same generation.

• from fig (1)

⇒ A have level no 0
⇒ B & C -11- 1
⇒ D, E, G, H -11- 2
⇒ J K -11- 3
⇒ L -11- 4

• Depth / Height of tree (Start from 1)

The Height of the tree is a maximum no. of nodes in a branch of a tree.

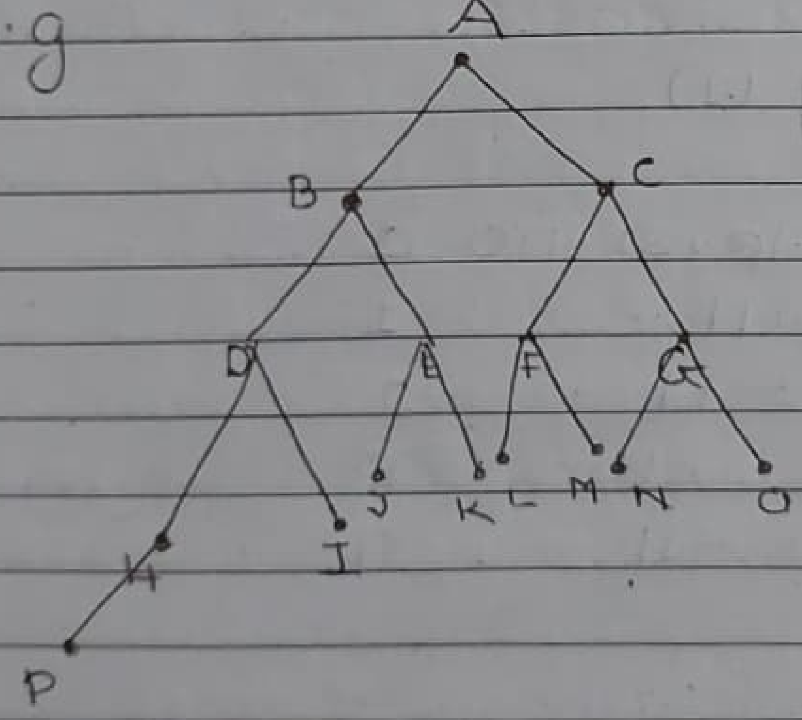
Fig(1) Depth of tree is 5

Complete binary tree

Consider any binary tree T. Each node of T can have atmost two children. accordingly one can show that level ~~of T~~ r of T can have atmost 2^r nodes.

The tree T is said to be complete if all it's level except possiblely the last, have the maximum no. of possible node and if all the node at last level appear as far left as possible.

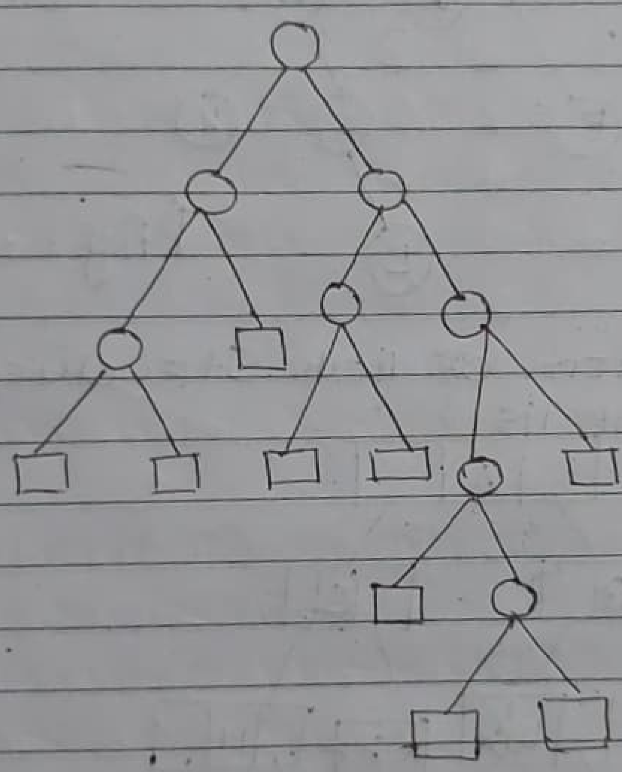
e.g



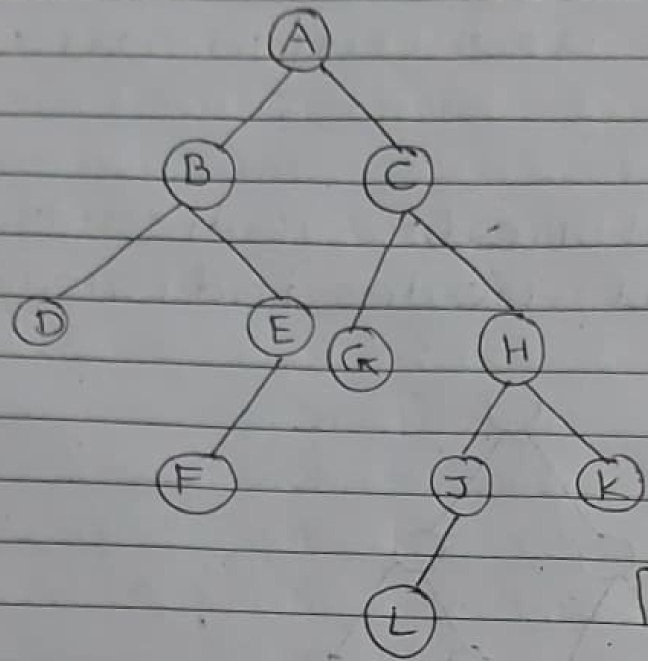
Page No. ____
Date : __/__/__

two children in such case the node with two children are called internal and node with zero children are called external node.

- Internal nodes indicated by → ○
- External nodes indicated by → □

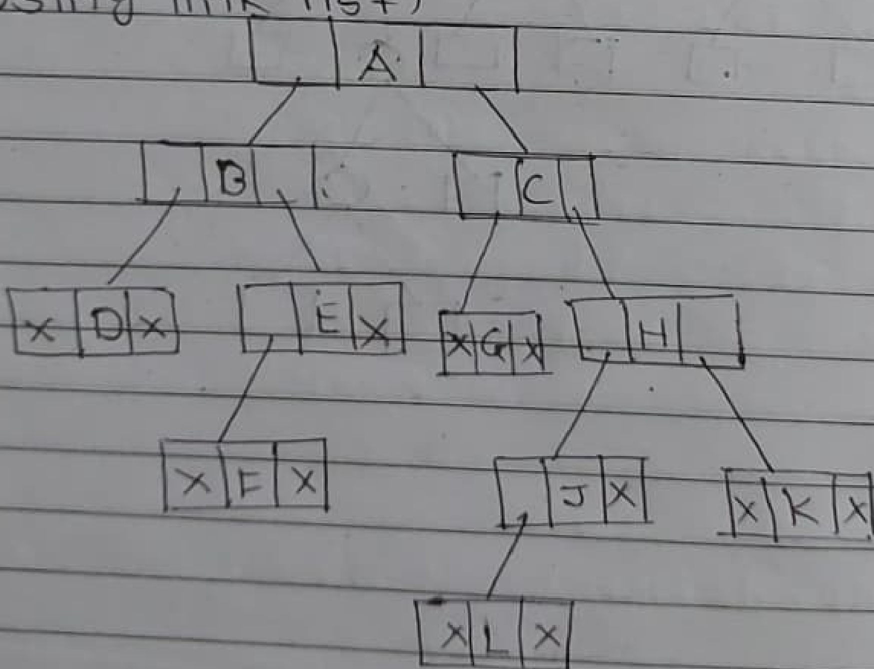


* Representation of Binary tree in Memory.



[fig : 1]

⇒ In the form of link. Structure
(using link list)



⇒ memory representation

Start		INFO	LEFT	RIGHT
9	1	C	10	2
	2	H	~~11~~6	4
	3	B	11	5
	4	K	-1	-1
	5	E	7	-1
	6	J	8	-1
	7	F	-1	-1
	8	L	-1	-1
	9	A	2	1
	10	G	-1	-1
	11	D	-1	-1

- Sequential Representation

Suppose T is a binary tree that is complete or nearly complete. then their is efficient way of maintaing T in memory called Sequential Representation of tree. This representation uses only a single linear array tree as follows.

1) The root R of T is stored In TREE[1]
2) If node n occupy TREE(k) then

Page No.
Date : / /

· left child is stored in TREE [2 * k] and right child is stored in Tree [2 * k + 1]

e.g of sequential repsentation from fig: [1]

Tree

1	A
2	B
3	C
4	D
5	E
6	G
7	H
8	
9	
10	F
11	
12	
13	
14	J
15	K
|	
|	
28	L

Traversing Binary tree

1. Pre ordered

Rules

1. processes the root
2. Traverse the left subtree in pre-ordered
3. Traverse a right subtree in pre-ordered

2. In-ordered

Rules

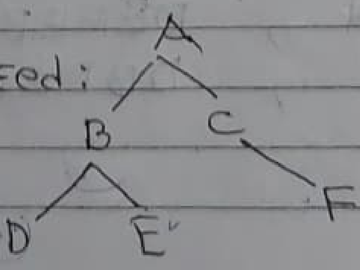
1. Traverse the left subtree in In-ordered
2. processes the root
3. Traverse the right sub-tree in In-ordered.

3. Post-ordered.

Rules

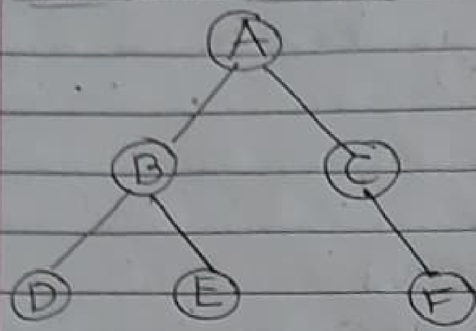
1. Traverse the left subtree in Post-ordered.
2. Traverse the right subtree in post ordered
3. Processe the root

eg. Pre-ordered:



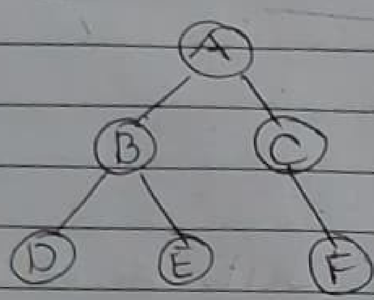
$\Rightarrow$ ABDECF

In-orderred



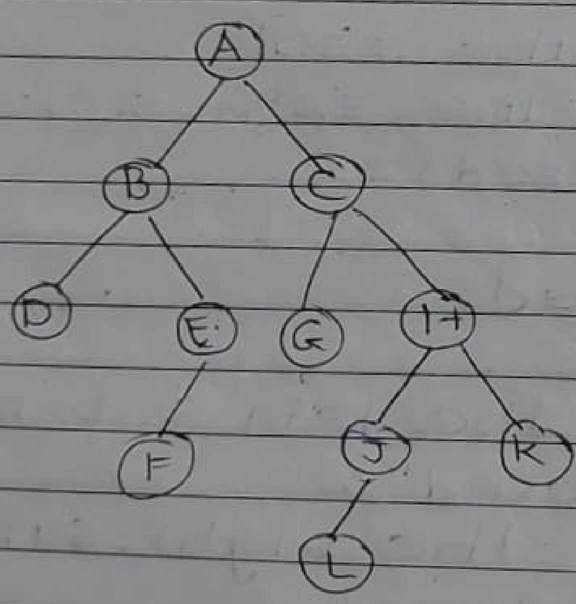
⇒ DBEACF

Post orderred



⇒ DEBFCA

Q.



Post orderred
left to Right
bottom to up

⇒ Pre orderred ⇒ ABDEFC GHJ~~KL~~LK

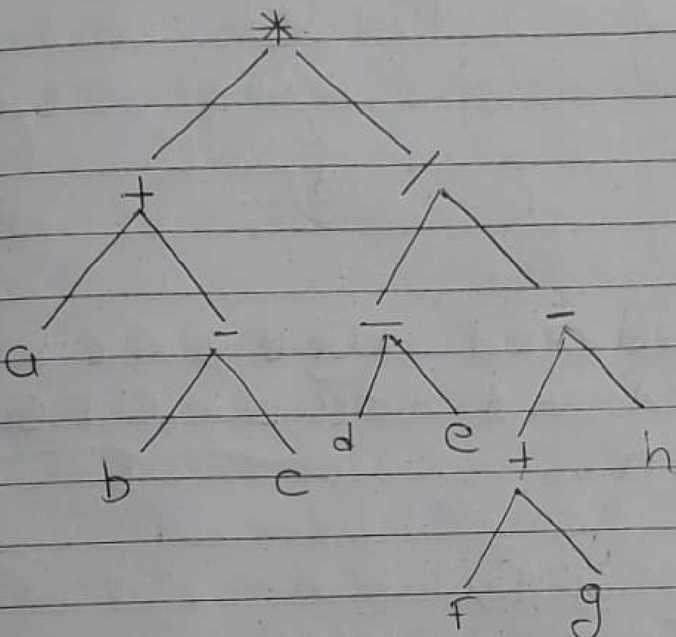
LOR bottom toup Post orderred ⇒ DEFEBGLJKHCA

In orderred ⇒ ~~DBFEB~~ DBFEAGCLJH

Q. Let, D denote the following algebraic expression

[a+(b-c)] * [(d-e)/(f+g-h)]
Draw the tree and determine pre ordered and post ordered

Ans:-



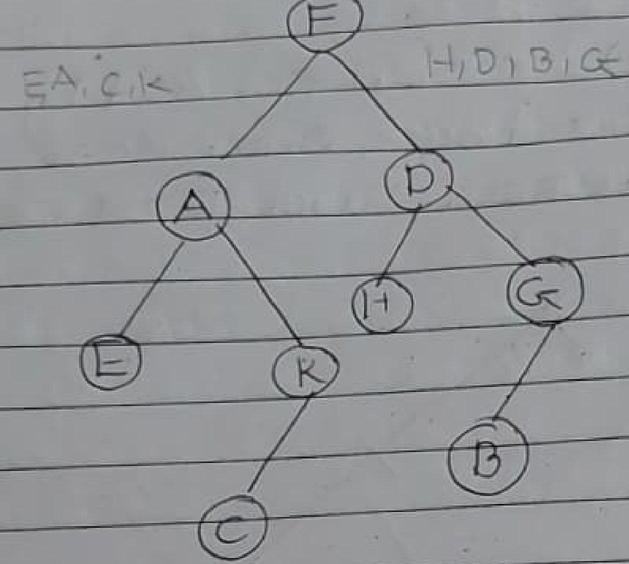
Pre-ordered :- *+a-bc/-de-+fgh

Post ordered :- abc-+de-+fgh-~~*~~/*

Q. A Binary tree T has 9 nodes the Inordered and preordered traversal of tree contain following sequences of node.
identify
L. Richild Inorder ⇒ E A C K F H D B G
Root Preordered → F A E K C D H G B
root

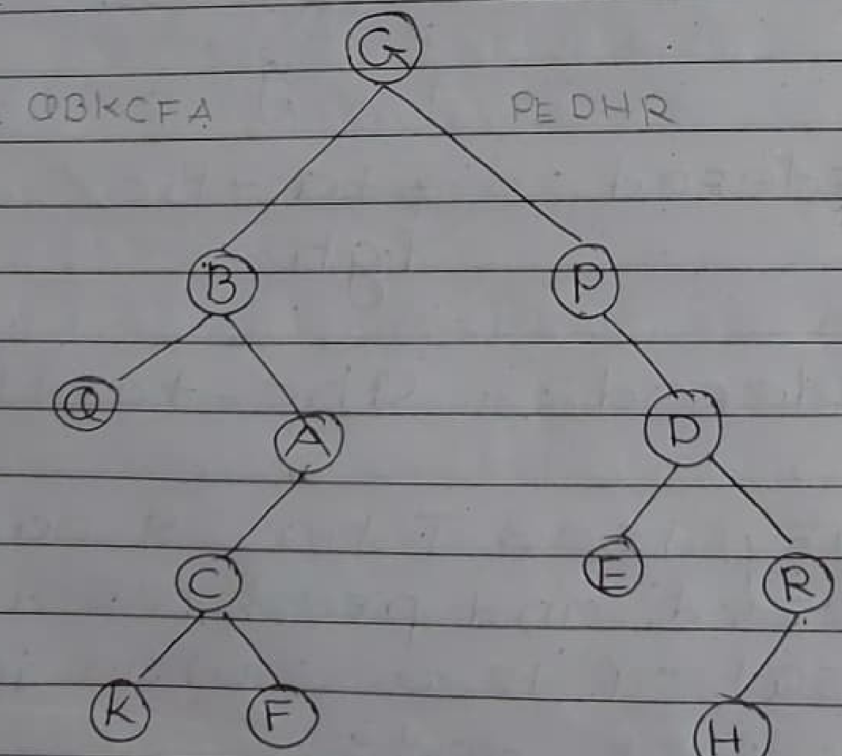
Draw the Tree

write step by step in paper puri diagram ek sath nahi nekalneki



Q.2 Pre ordered = G B Q A C K F P D E R H
In ordered = $\underbrace{Q\ B\ K\ C\ F\ G\ A}_{L}\ G\ \underbrace{P\ E\ D\ H}_{R}$

Ans :-



• Traversing algorithm using stack

1. Pre-order Traversal (short cut

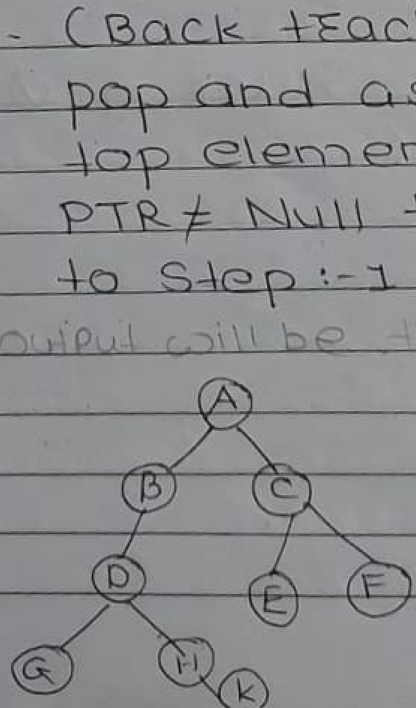
Algorithm

Intially push Null on to statack and set PTR := Root, then Repeat following step until PTR = Null

Step 1 :- procced down the left most path rooted at PTR, processing each node N on the path and pushing each right child R(N), if any onto stack. The traversing end after a node N with no left child is processes.

Step 2 :- (Back tracking) :
pop and asign to PTR the top element on stack. If PTR ≠ Null then written to step :-1 otherwise exit.

output will be the processes



Date : / /

① PTR := A   Stack ⇒ ∅

② Process A
Process B   Stack ⇒ ∅, C
Process D
Process G   Stack ⇒ ∅, C, H
(Back tracking)
Pop H from the stack
PTR := H
Process H   Stack ⇒ ∅, C, K
(Back tracking)
Pop K
PTR := K
Process K   Stack ⇒ ∅, C
Pop C
PTR := C
Process C   Stack ⇒ ∅
Process E   Stack ⇒ ∅, F
(Back tracking)
Pop F
PTR := F
Process F   Stack ⇒ ∅

Page No.______
Date : __/__/__

- **Insertion Traversing.**

2. **In-order Traversal**

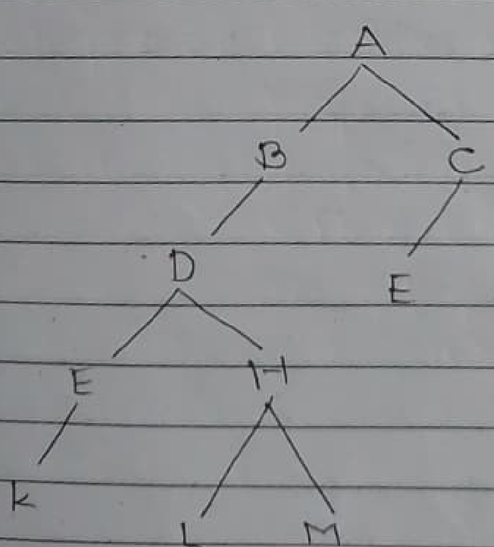
* Initially push Null on to stack Set PTR = Root, then repeat the following step until Null is pop from stack

<u>Step - I</u> :

proceed down the left most path rooted at PTR pushing each node N on to stack and stopping when node N with no left child push onto stack.

<u>Step - II</u> :- (Back tracking)

pop and process the node on stack, if Null is pop then exit If a node n with right child is process then Set PTR = Right child and return to step (I)



KL M H D B E C A

① PTR := A   Stack ⇒ ∅
Stack ⇒ ∅, A, B, D, E, K
PoP K   PTR:= K   PROcess K
PoP E   PTR := E   PROcess E
PoP D   PTR := D   PROcess D
Stack ⇒ ∅, A, B, H, L
PoP L   PTR := L   PROcess L
PoP H   PTR := H   PROcess H
Stack ⇒ ∅, A, B, M.
PoP M   PTR := M   PROcess M
PoP B   PTR := B   PROcess B
PoP A   PTR := A   PROcess A
Stack ⇒ ∅, C, E
PoP E   PTR := E   PROcess E
PoP C   PTR := C   PROcess C
Stack ⇒ ∅.
PoP ∅   PTR = ∅
Exit.

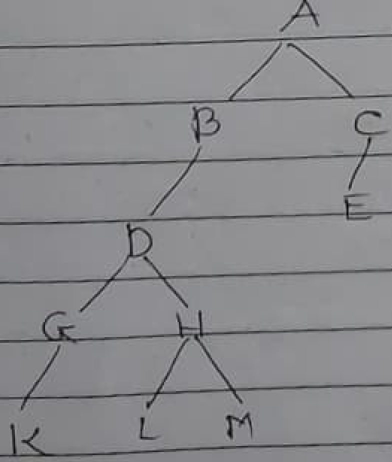
3. Post-Order Traversal

Intially push Null onto stack and set PTR := Root, then Repeat the following step Until PTR :- Null

Step 1 :-

Procced down the left most path rooted at PTR at each node N of the path push N on to stack. and if N has right child R(n), push -R(n) on to stack.

Step 2 :- (Back Tracking)
pop and process the positive nodes on stack if null is pop then exit if a negative node is pop i.e If PTR := -N for some node N Then set PTR = N and return to step(1)



[Post order :-
KGLMHDBECA]

Page No.____
Date : __/__/__

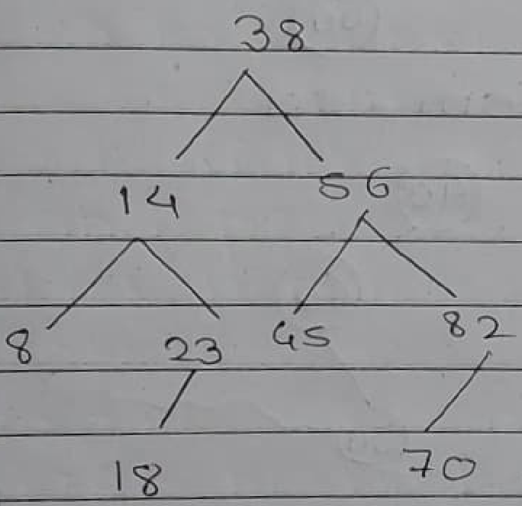
(1) PTR := A   Stack ⇒ Ø, A, -C, B, D, -H, G, K
POP K   PROCESS K
POP G   PROCESS G
POP -H   PTR = -H , PTR = H
Stack ⇒ Ø, A, -C, B, D, H, -M
POP L   PROCESS L
POP -M   PTR = -M , PTR = M
Stack ⇒ Ø, A, -C, B, D, H, M

POP M   PROCESS M
POP H   PROCESS H
POP D   PROCESS D
POP B   PROCESS B
POP -C   PTR = -C , PTR = C
Stack ⇒ Ø, A, C, E
POP E   PROCESS E
POP C   PROCESS C
POP A   PROCESS A
POP Ø
Exit

left child < root ≤ right child

Binary Search Tree

Suppose T is a binary tree then T is called binary search tree or binary sorted tree if " Each node N of T has the following property :-
The value of N is greater than every value in the left subtree of N and is less than every value in the right subtree of N.



Q1. ITEM 70 Search.
Q.2 ITEM 20 Insert

Date: / /

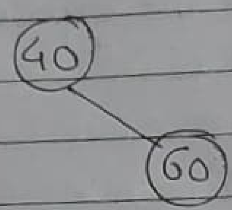
by Default first item is Root

Q. Suppose the following six numbers are inserted in ordered into an empty binary search tree
40, 60, 50, 33, 55, 11

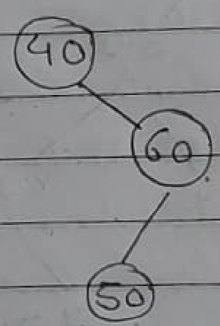
Ans:-

Step I :- Item = 40

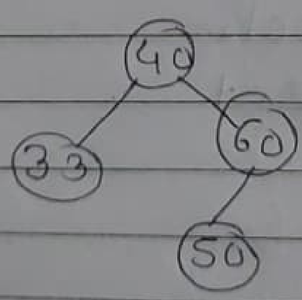
Step II :- Item = 60



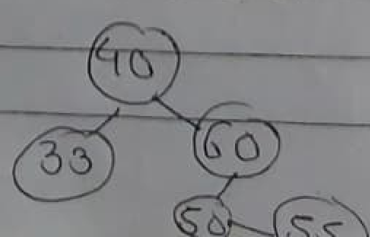
Step III :- Item = 50



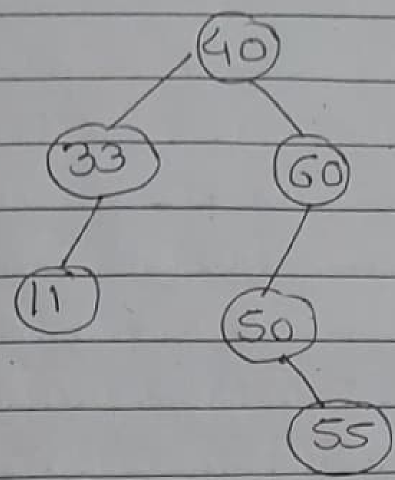
Step IV :- 33 = Item



Step V :- Item = 55

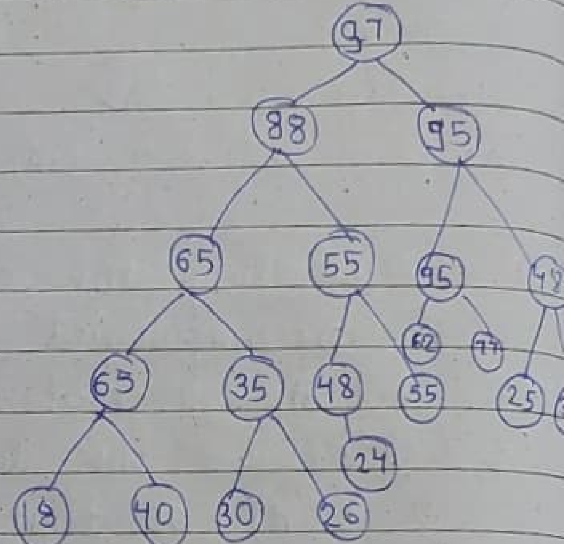


Step VI :- Item := 11



Page No.
Date

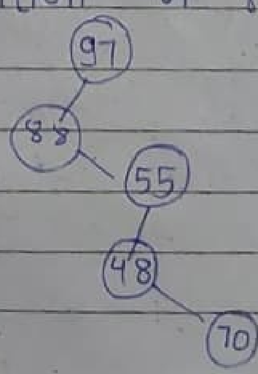
- **Heap Sort** : If Suppose H is a complete binary tree with n elements then H is called heap or max heap if each node N(H) has following properties.
" The value at N is greater than or equal to the value at each of the children of N



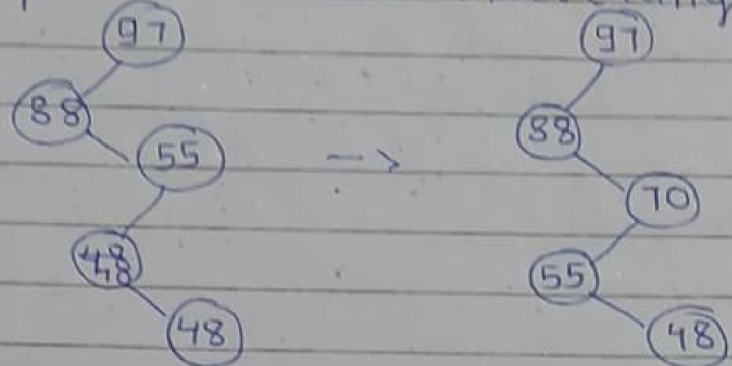
Inserting into a heap : We insert item into heap as follows

1) First adjoint item at the end of H so that H is still a complete tree but not neccessarily a heap.
2) Then let item rise to its appropriate place in H so that H is finally a heap.

→ Insert item 70 into a heap
To make it a complete tree, 70 is inserted at position of right child of 48.



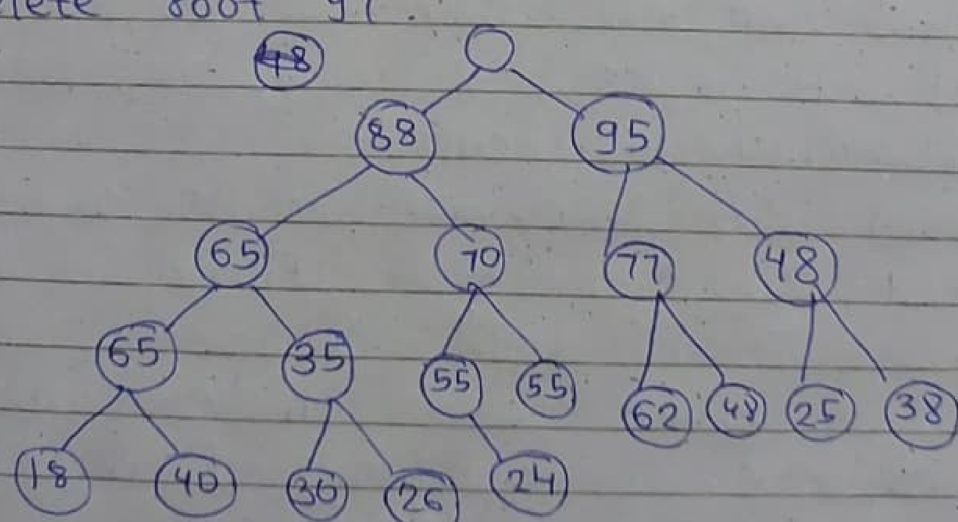
Now to make it is heap tree 70 is compared with 88 48 and as 48 is smaller their positions are interchange.

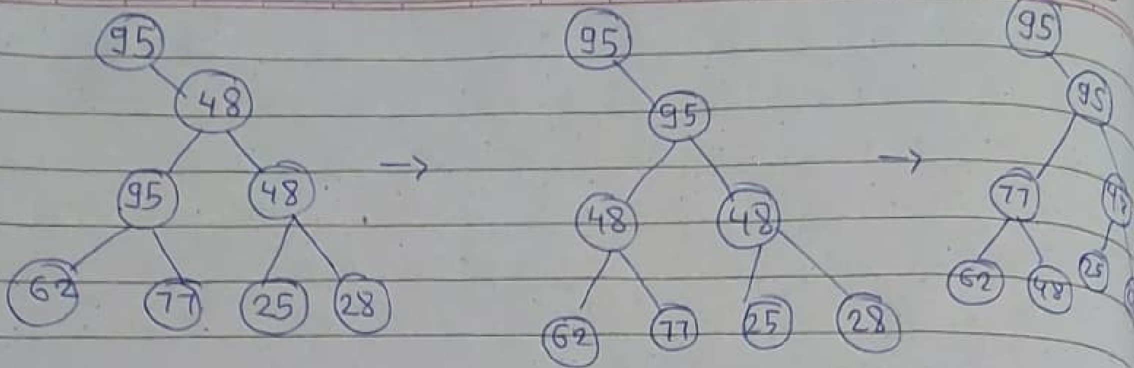


- Deleting the root of a heap : This is accompolished as follow:

1) Assign root r to some variable item
2) Replace the deleted node R by last node L(H) so that H is still a complete tree but not necessarily a heap
3) Let L sink to its appropriate place in H so that H is finally a heap.

⇒ Delete root 97.



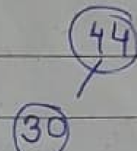


Q. Consider the following elements and construct the heap tree.

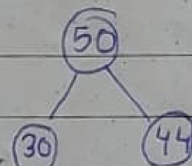
* 44 30 50 22 60 55 77 55

item = 44

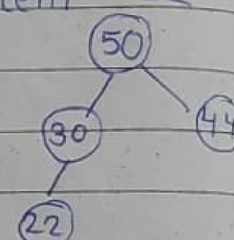
item = 30



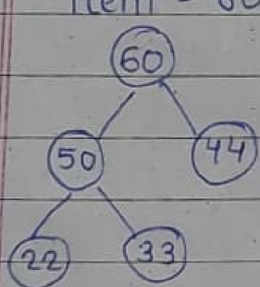
item = 50



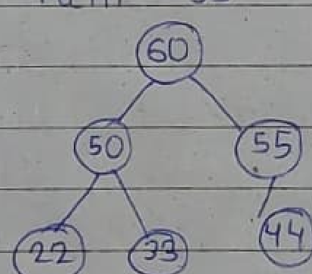
item = 22



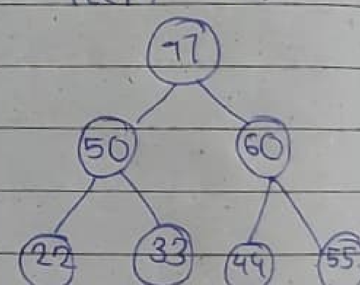
item = 60



item = 55



item = 77



Q. Draw the tree * 44 33 11 55 77 90 40 60 99 22 88 66

item = 44

ite