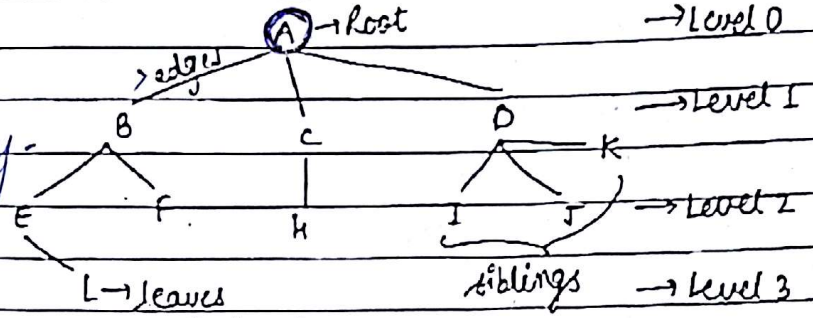
Date

-Trees-



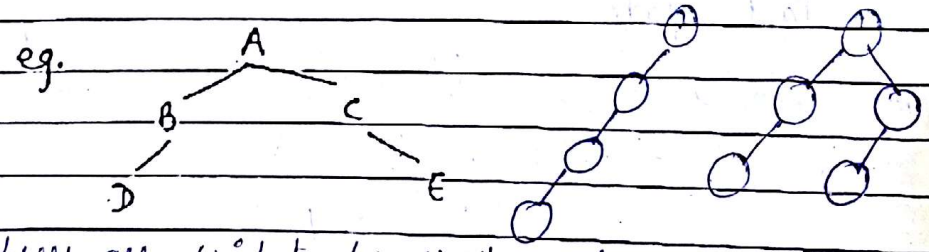
Terminology -
- Root
- Node
- Level
- edge

- Degree of nodes - no. of successors of the node
- Degree of tree - degree of the node having max. degree (here = 3)

- Path - Connecting edges b/w source and destination
  # No backtracking
  eg. path (A → L) : A → B → E → L

- Depth of tree - (Max. level of tree + 1) here, d = 4

→ Binary Trees - Each parent node can have atmost two child nodes



⊙ 2 binary trees are said to be similar when they have the identical structures only.

→ Complete Binary Tree - Every node should have two children except the terminal / last nodes

Depth of a CBT = $D_n = \lfloor (\log_2 n) + 1 \rfloor$

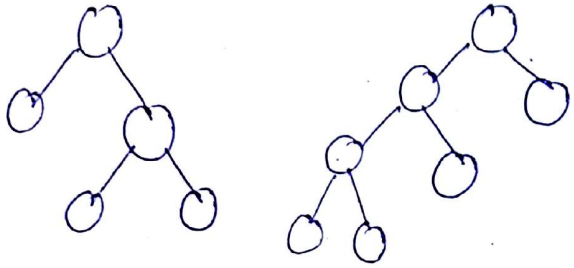
↳ Greatest Integer fn

⊙ copy = same structure + same nodes

Strictly binary tree

- BT with every node N having either 0 or 2 tree.
- also called as extended 2-Tree or 2-Tree.
- nodes with 2 children = internal node
  " " 0 " = external "

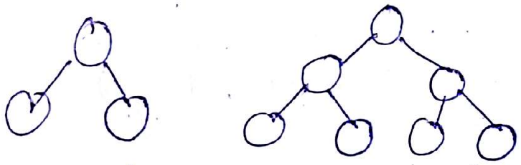
eg.



Complete Binary tree

- spl type of strictly binary tree where all leaves are at the same level.

eg.



- each node should have 2 children except ~~the~~ last node.

Total nodes $= 2^0 + 2^1 + 2^2 + \dots + 2^d$

$= 1 + 2\frac{(2^d - 1)}{2-1}$ $\quad S_n = \frac{a(2^n - 1)}{r-1}$

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

$T = 2^{d+1} - 1$

or $2^{d+1} = T + 1$

Taking log on both sides,

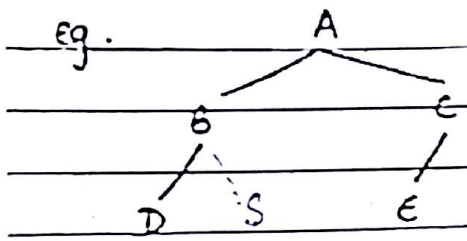
$d+1 = \log_2(T+1)$

or $\boxed{d = \log_2(T+1) - 1}$

$= \theta(\log n)$

→ Representation of Trees –

eg.



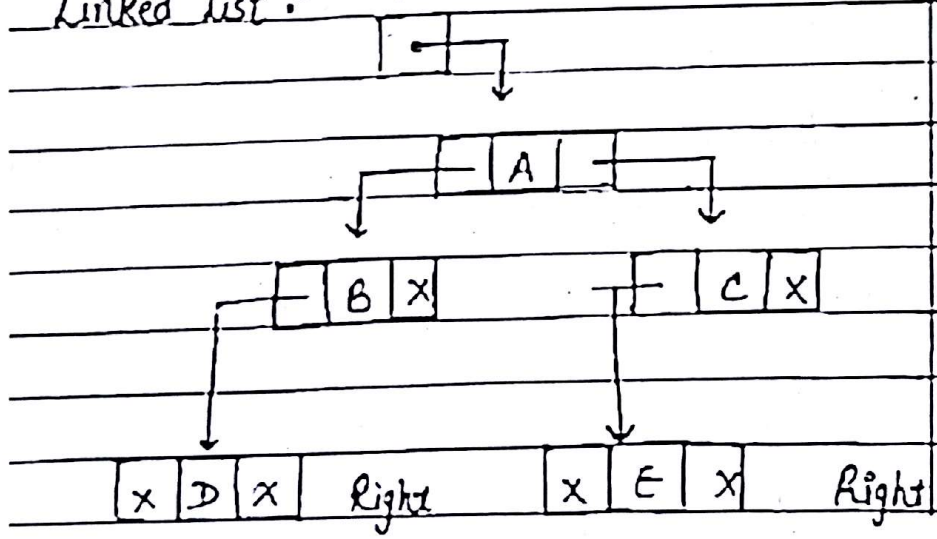
Left child = 2I

Right " = 2I + 1

Array:

A	1	Root = 1 (A)
B	2	
C	3	
D	4	(2X2)
S	5	(2X2)+1
E	6	(2X3)

Linked list :-



Stack –

	INFO	LEFT	RIGHT
1)	D	0	0
2)	A	4	6
3)			
4)	B	1	0
5)		3	
6)	C	7	0
7)	E	0	0

```
Struct node
{ struct node * left;
    char data;
    Struct node * right;
};
```

Date

→ Traversals -

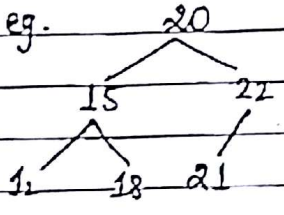
True Traversals

Level order

starts from ROOT	Preorder	Inorder → same as Post	Postorder → starts from extreme left leaf
	Root	Left	Left
	Left	Root	Right
	Right	Right	Root

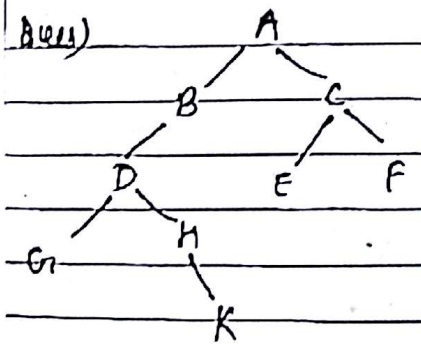
→ Pre-order Traversal using stack -

eg.



1) Stack ϕ
2) Process ϕ, 20
   Stack: ϕ, ~~22~~ 22 P: 20
3) Process 15
   Stack: ϕ, 22, 18 P: 20, 15
4) Process 12 P: 20, 15, 12
5) Pop top of stack of 18
   Stack: ϕ, 22 P: 20, 15, 12, 18
6) Pop and process 22
   Stack: ϕ P: 20, 15, 12, 18, 22
7) Process 21 P: 20, 15, 12, 18, 22, 21

Ans)



1) Stack ϕ
2) Process A P: A
   S: ϕ, C
3) Process B P: A, B
   S: ϕ, C
4) Process D P: A, B, D
   S: ϕ, C, H

5) Process G — P: A, B, D, G

6) Pop stack and process H — P: A, B, D, G, H
S: ϕ, C, K

7) Pop stack & process K — P: A, B, D, G, H, K
S: ϕ, C

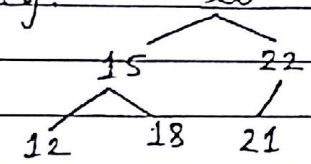
8) Pop stack & process C — P: A, B, D, G, H, K, C
S: ϕ, F

9) Process E — P: A, B, D, G, H, K, C, E

10) Pop stack & process F — P: A, B, D, G, H, K, C, E, F

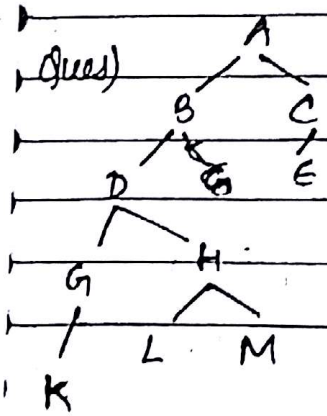
→ In-order Traversal using stack:

eg.



1) Push on stack: ϕ, 20, 15, 12
2) Process 12
3) Process 15
S: ϕ, 20, 18 — P: 12, 15
4) Pop & process 18
S: ϕ, 20 — P: 12, 15, 18
5) Pop & process 20
S: ϕ — P: 12, 15, 18, 20
6) Push 22 & 21
S: ϕ, 22, 21
7) Process 21
8) Process 22 — P: 12, 15, 18, 20, 21, 22

Ques)



1) Push onto stack: P: A, B, D, G, K
2) Process K
3) Process G
4) Process D — P: K, G, D
5) Push on stack H, L
S: ϕ, A, B, H, L

6) Process L
7) Process H — P: K, G, D, L, H
8) Push on stack M
S: ϕ, A, B, M
9) Pop and process M — P: K, G, D, L, H, M
S: ϕ, A, B
10) Process B
11) Process A — P: K, G, D, L, H, M, B, A
12) Push C & E
S: ϕ, C, E
13) Pop & process E
14) Process C — P: K, G, D, L, H, M, B, A

→ Preorder (INFO, LEFT, RIGHT, ROOT)

1) Set TOP = 1, STACK[1] = NULL and PTR := ROOT
2) Repeat steps 3 to 5 while PTR ≠ NULL
3) Apply process to INFO[PTR]
4) If RIGHT[PTR] ≠ NULL
Set TOP := TOP + 1 and STACK[TOP] := RIGHT[PTR]
5) If LEFT[PTR] ≠ NULL
Set PTR := LEFT[PTR]
else
Set PTR := STACK[TOP] and TOP := TOP - 1

: Inorder (INFO, LEFT, RIGHT, ROOT)-

Set TOP := 1, STACK[1] = NULL and PTR := ROOT
Repeat steps while PTR ≠ NULL

(a) Set TOP := TOP+1 and STACK [TOP] = PTR
(b) Set PTR := LEFT [PTR]
3) Set PTR := STACK [TOP] and TOP := TOP+1
4) Repeat steps 5 to 7 while PTR ≠ NULL (Backtracking)
5) Apply process to INFO [PTR]
6) If RIGHT [PTR] ≠ NULL
(a) Set PTR := RIGHT [PTR]
(b) Goto step 3
7) Set PTR := STACK [TOP] and TOP := TOP+1
8) Exit

→ Postorder (INFO, LEFT, RIGHT, ROOT) –

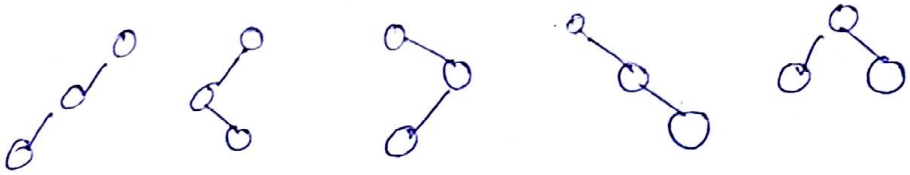
1) Set TOP := 1, STACK [1] = NULL and PTR := ROOT
2) Repeat steps 3 to 5 while PTR ≠ NULL
3) Set TOP := TOP+1 and STACK [TOP] := PTR
4) If RIGHT [PTR] ≠ NULL
Set TOP := TOP+1 and STACK [TOP] := -RIGHT [PTR]
5) Set PTR := LEFT [PTR]
6) Set PTR := STACK [TOP] and TOP := TOP-1
7) Repeat while PTR > 0
(a) Apply process to INFO [PTR]
(b) Set PTR := STACK [TOP] and TOP := TOP-1
8) If PTR < 0
(a) Set PTR := -PTR
(b) Goto step 2
9) Exit

Counting binary trees

if $n = 1$, one binary tree

$n = 2$ 2 trees

for $n = 3$,



In general, No. of BT $= \left(2^n - n\right)$

→ Construction of Binary Tree -

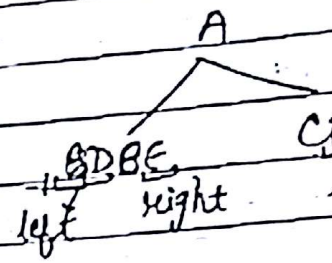
Any two order combinations / traversals are given using which we have to form / construct our binary tree

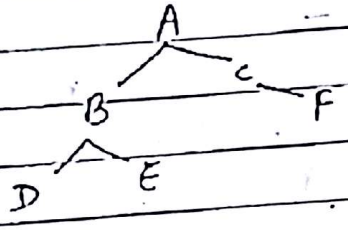
eg. Pre – ABDECF N L R
In – DBEACF L N R

First check in pre-order and then inorder for each element

Root node

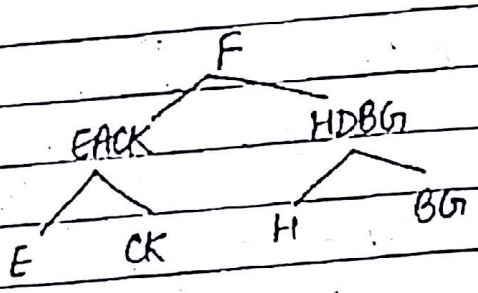
DBE, A CF,
left sub-tree form right sub-tree

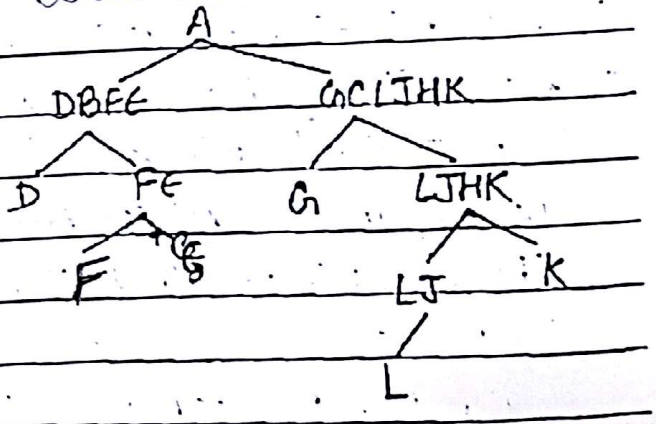




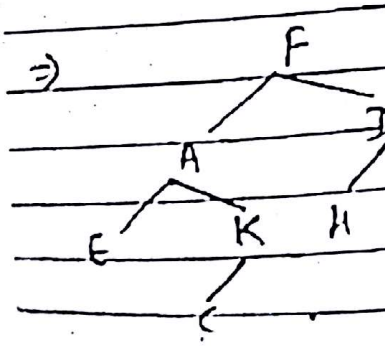
eg. In: EACKFHDBG
Pre: FAEKCDHGB

Post: DFEBGLJKHCA
In: DBFEAGCLJHK

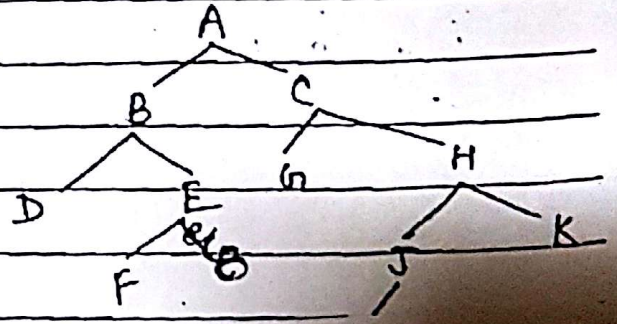




⇒

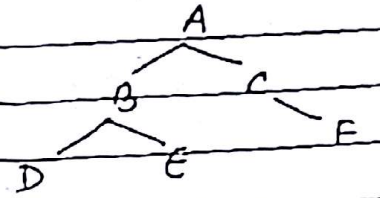


⇒

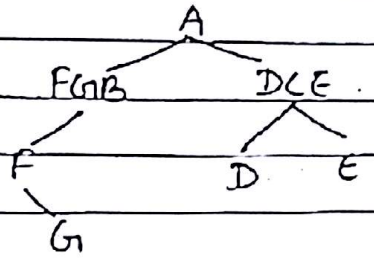


eg. Post: DEBFCA
In: DBEACF

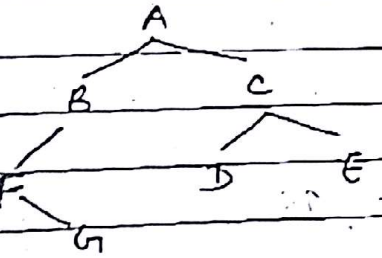
⇒



Post: GFBDECA
In: FGBADCE



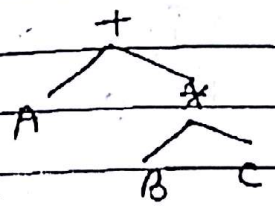
⇒



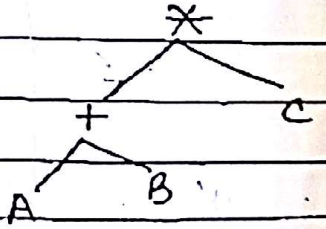
→ Expression Trees –

eg. A+B

eg. -A+B*C



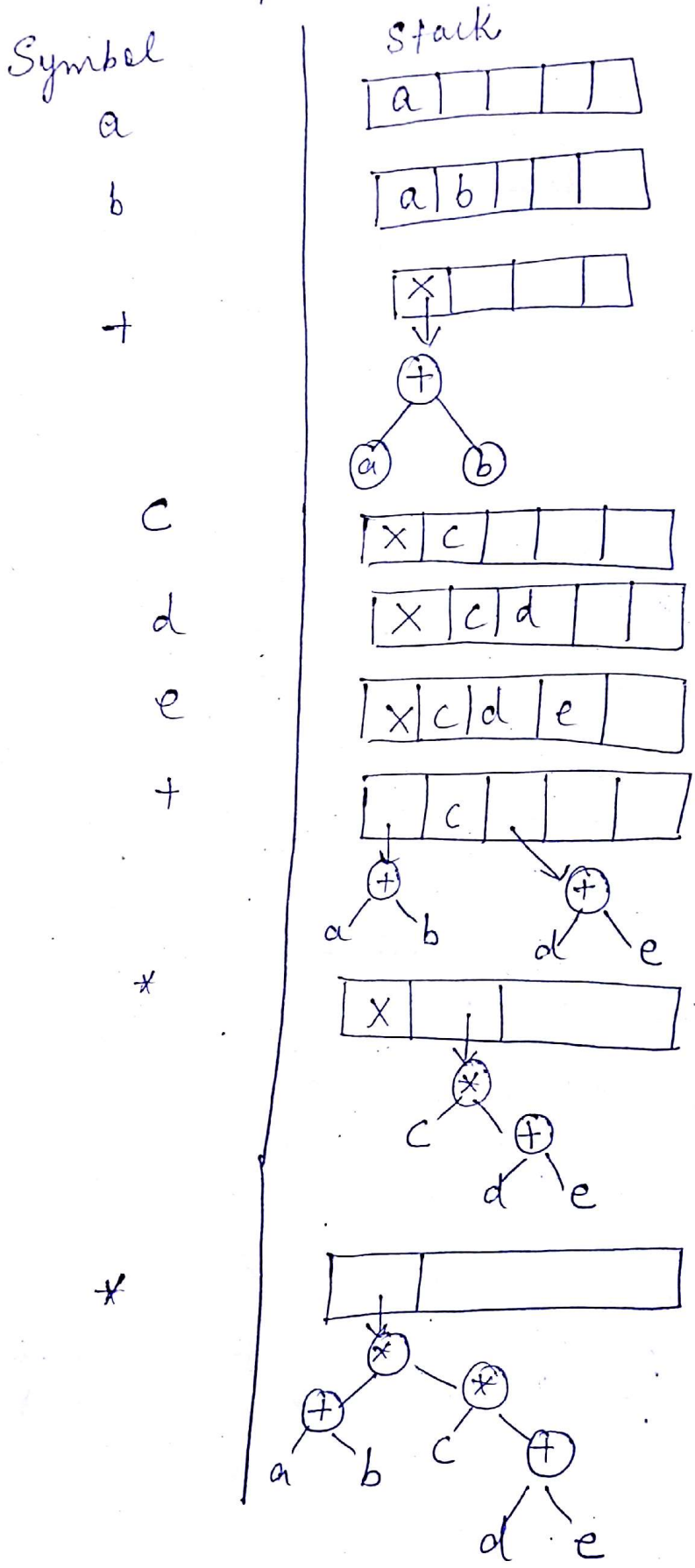
eg. (A+B)*C



leaves : three operands
other nodes : operators.

eg. ab + cde + * *

construct expression tree from postfix.

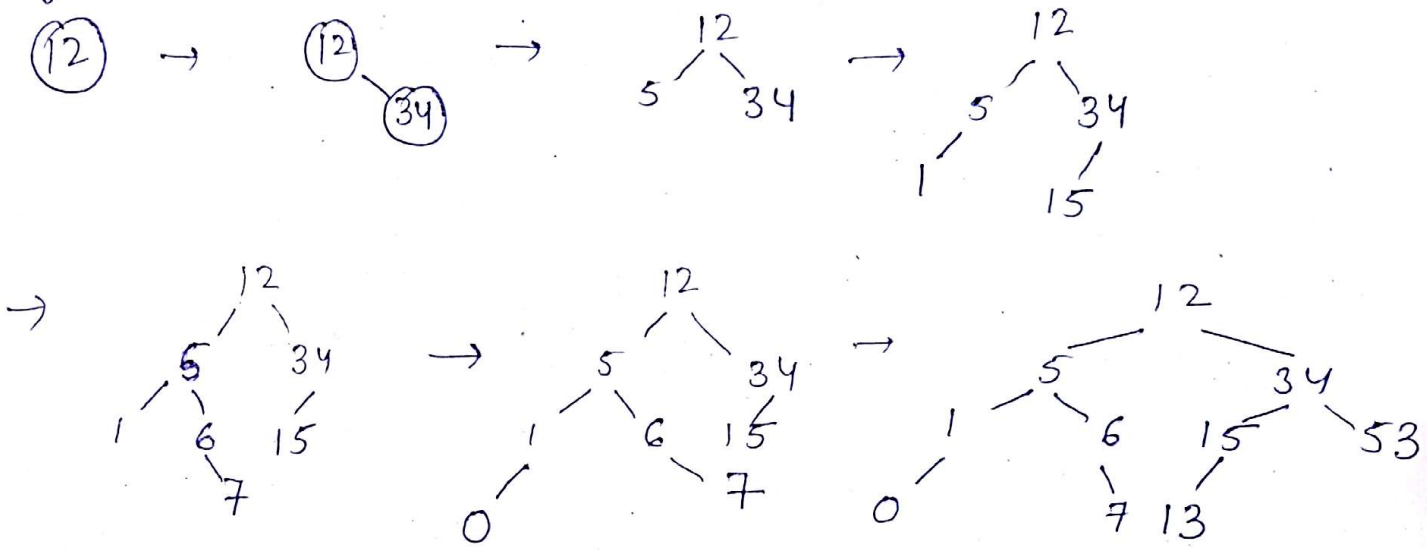


Binary search trees

operations :
- Insertion
- deletion
- searching
- getting data in sorted order (inorder traversal of BST).

(1) Insertion

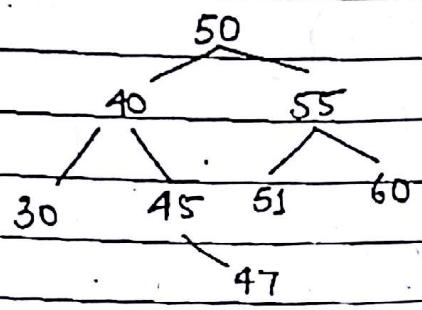
eg. 12, 34, 5, 1, 15, 6, 7, 0, 13, 4, 53



ans.

Date

→ Binary Search Tree (BST) -

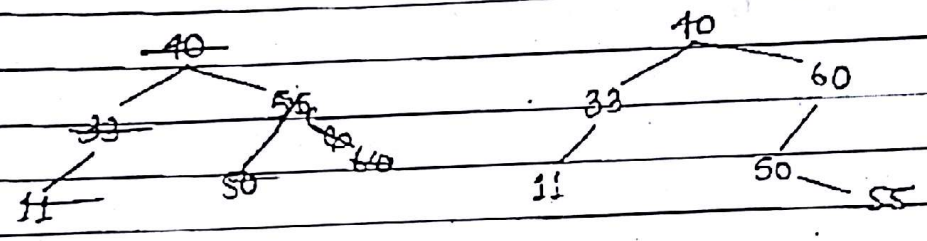


Left child always has smaller value than root value while right child always has higher value than root.

Ques) 40, 60, 50, 33, 55, 11 — form a binary search tree

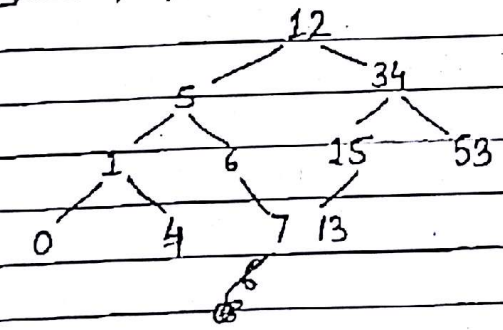
→ root (always the first element)

11, 33, 40, 50, 55, 60



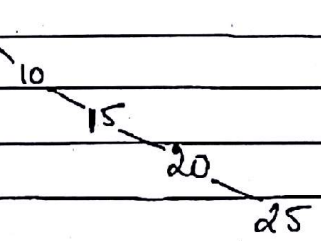
Ques) 12, 34, 5, 1, 15, 6, 7, 0, 13, 4, 53

→ root



Right-skewed tree - All the element of BST are on the right side.

eg. 5, 10, 15, 20, 25 → ASCENDING ORDER



Left-skewed tree - All the elements are on the left side (Descending Order

BST insertion

1. If (tree = NULL)
   tree → left = tree → right = NULL
   tree → num = digit ; // item

   else
   if (digit < tree → num)
   tree → left = insert (tree → left, digit);

   else
   if (digit > tree → num)
   tree → right = insert (tree → right, digit);

   else
   if (digit == tree → num)
   printf (" Duplicate node ");
   exit (0).

2. Return

BST searching

1. void search (struct rec * tree, digit)
2. if (tree = NULL)
   printf (" No. does not exist ").
   else if (digit == tree → num)
   printf (" %d \n", digit);
   else if (digit < tree → num)
   search (tree → left, digit)

   else
   search (tree → right, digit);

3. Return

Disadvantages — i) BSTs are not balanced. This leads to wastage of memory space while storing the data in any array

Deletion:-

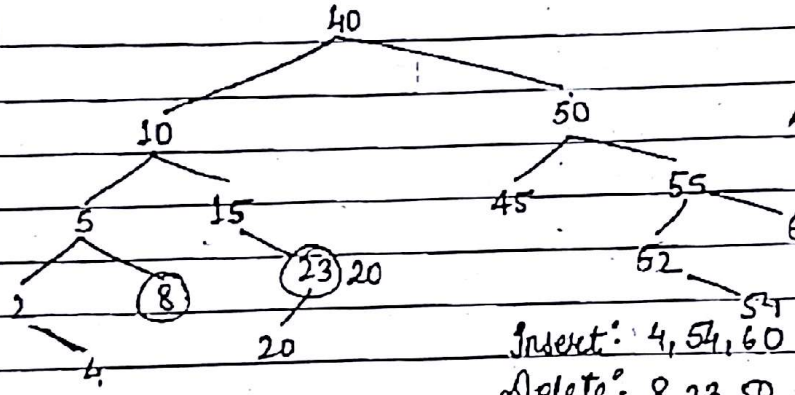
There can be three types of nodes in BSTs —

i) No child - node is simply deleted

ii) Single child - child takes the place of its parent after parent node is deleted.

iii) Two children - Inorder successor takes the place of the parent node

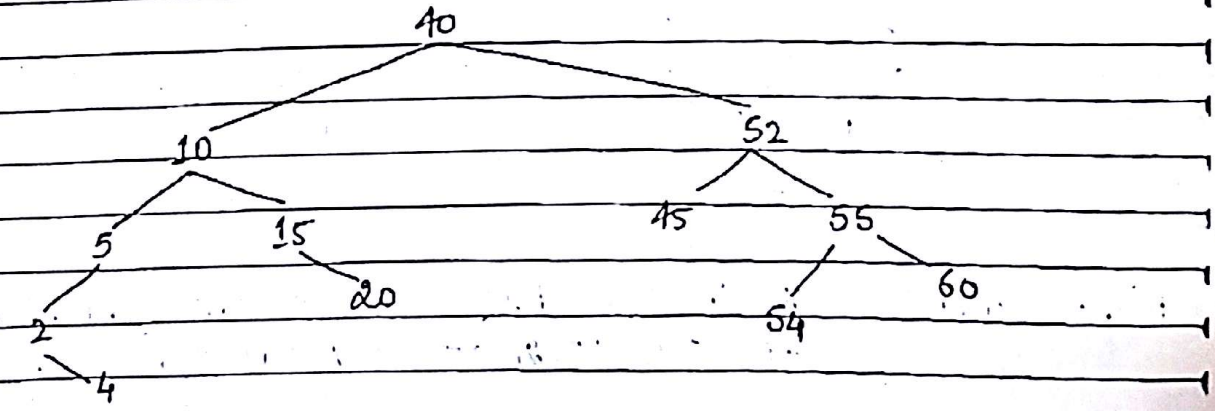
Ques)



45, 50, 52, 54, 55, 60

Insert: 4, 54, 60

Delete: 8, 23, 50

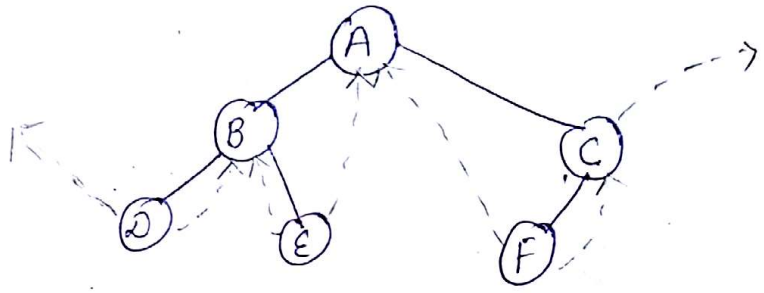


Threaded binary tree

In a linked representation of a binary tree, the no. of null links are actually more than non-null pointers.

(*) Idea is to replace all the null pointers by the appropriate pointer values called 'threads'.

⇒ Lchild = points to previous node in inorder
Rchild = " " next " " "



Inorder =

D B E A F C

•) These trees make in-order tree traversal a little faster.

→ Heap (tree):-

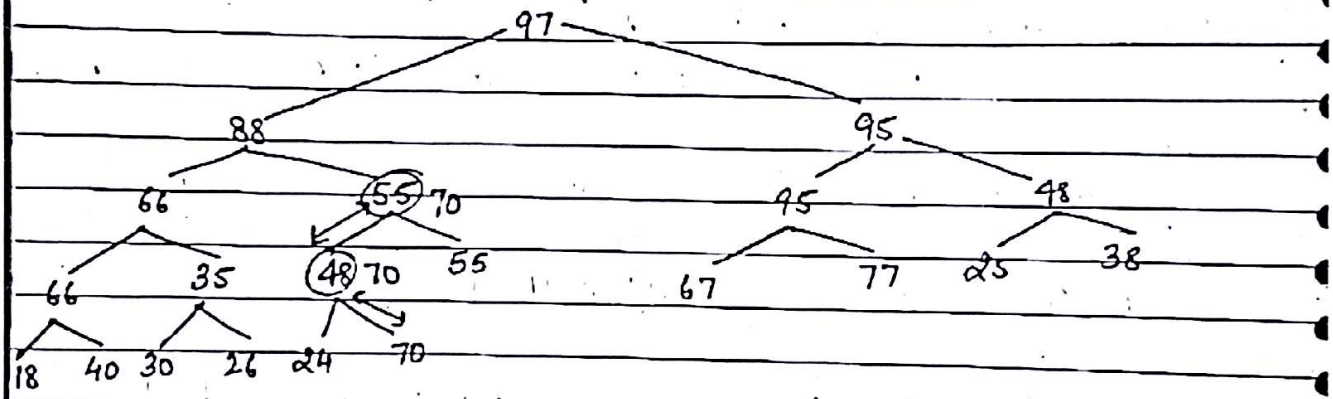
Root is either smaller or greater than both its children.

Maxheap – Root ≥ both children
Minheap – Root < both children

Insertion-

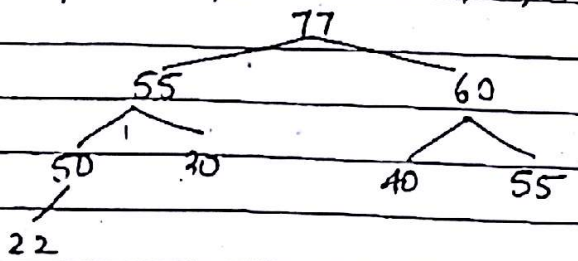
Always try to make a CBT while inserting a new node.

eg. Insert 70 to the given heap.



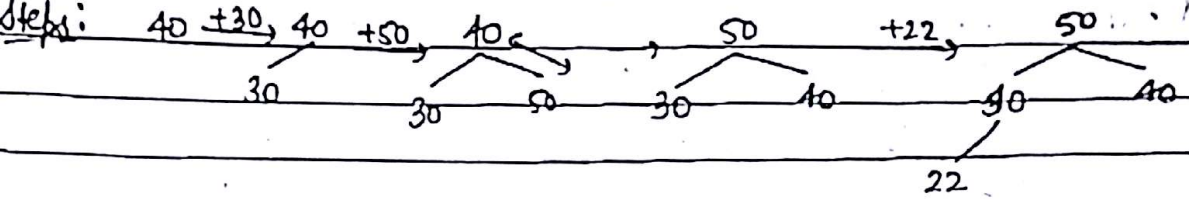
i) In order to complete the binary tree [CBT], we initially add 70 onto the right side of 48.
ii) Then we compare the value of child 70 & parent 48 and thus interchange the nodes.
iii) Second comparison with node 55 result in another interchange.

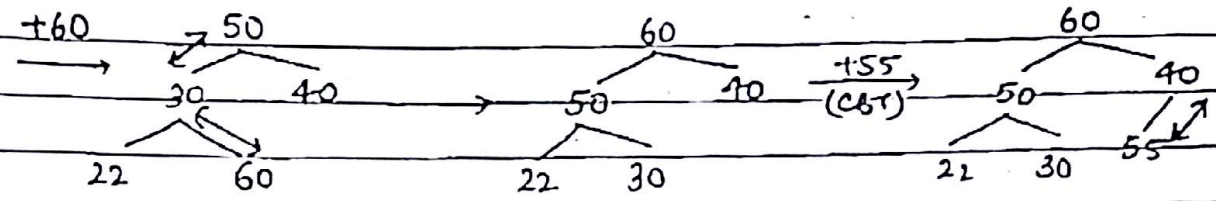
eg. Form a heap: 40, 30, 50, 22, 60, 55, 77, 55

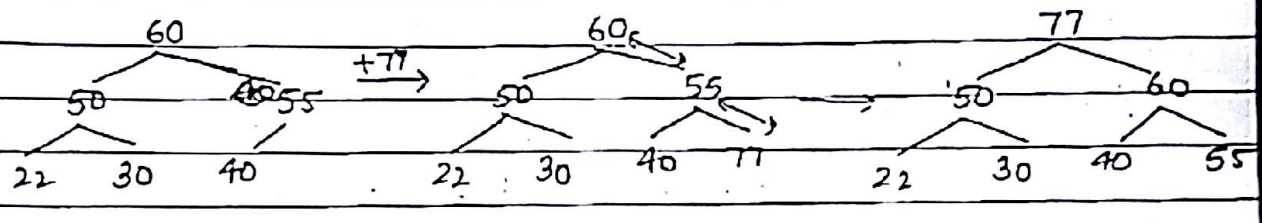


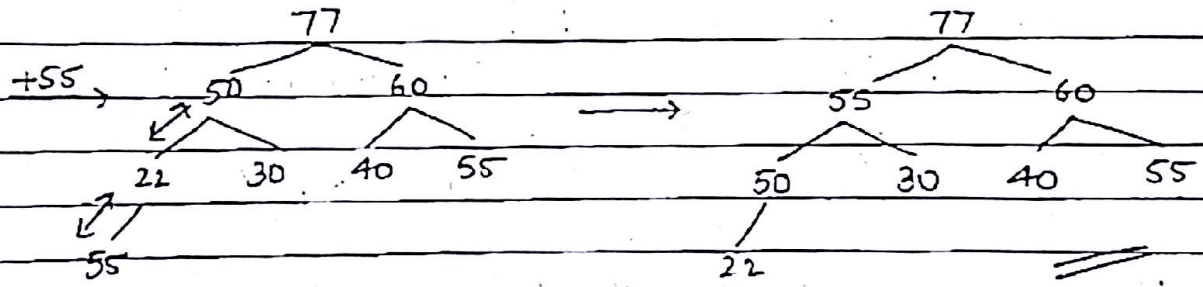
Date

Steps:





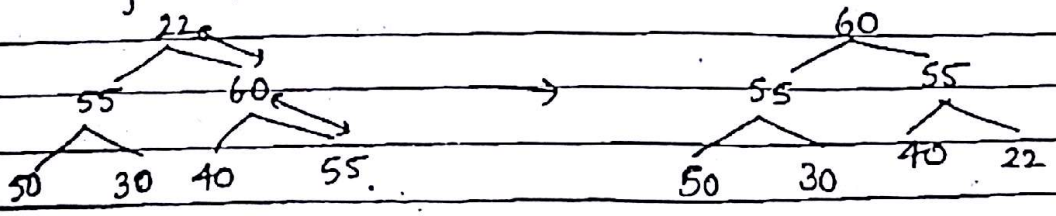




→ Deletion (from the top root) -

Replace the root with terminal node and then compare and interchange values

eg. deleting 77 from above heap

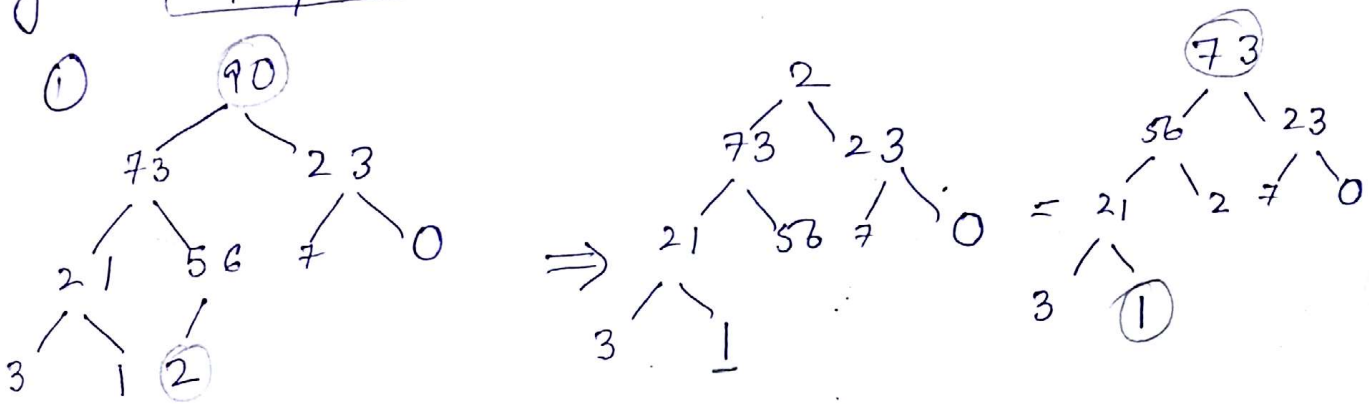


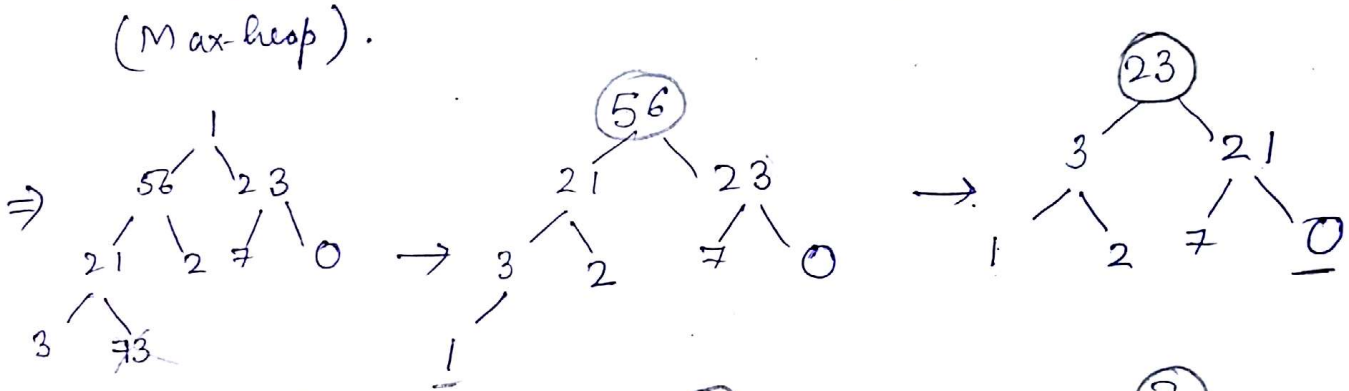
Heap Sort

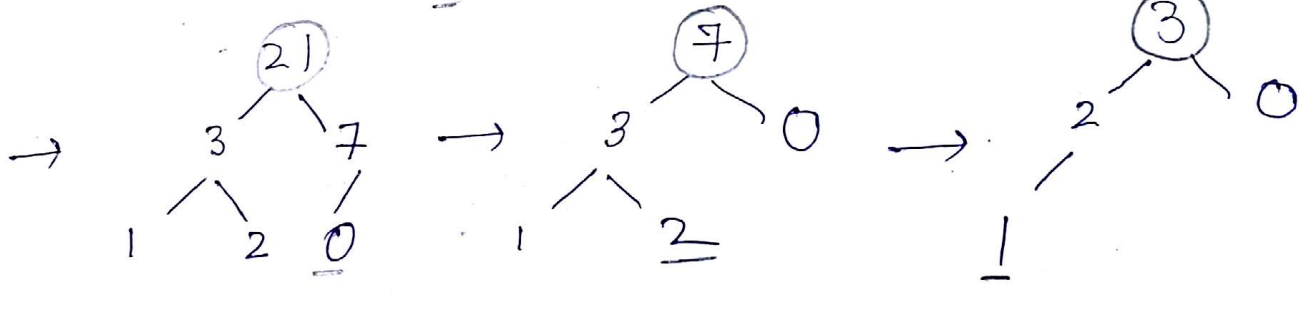
① Create a heap from the given array
② Swap root node & last node, decrease the size of heap by 1. Repeat.

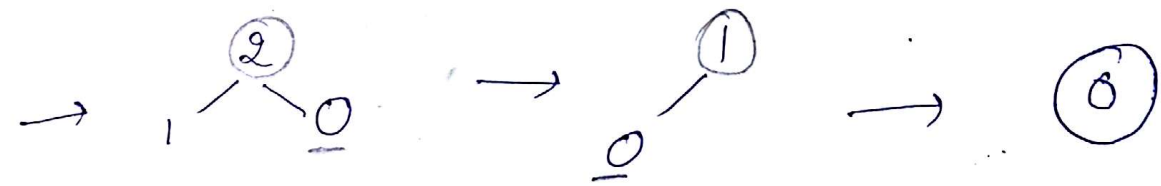
eg.

0	1	2	3	4	5	6	7	8	9
23	56	7	1	2	90	0	3	21	73





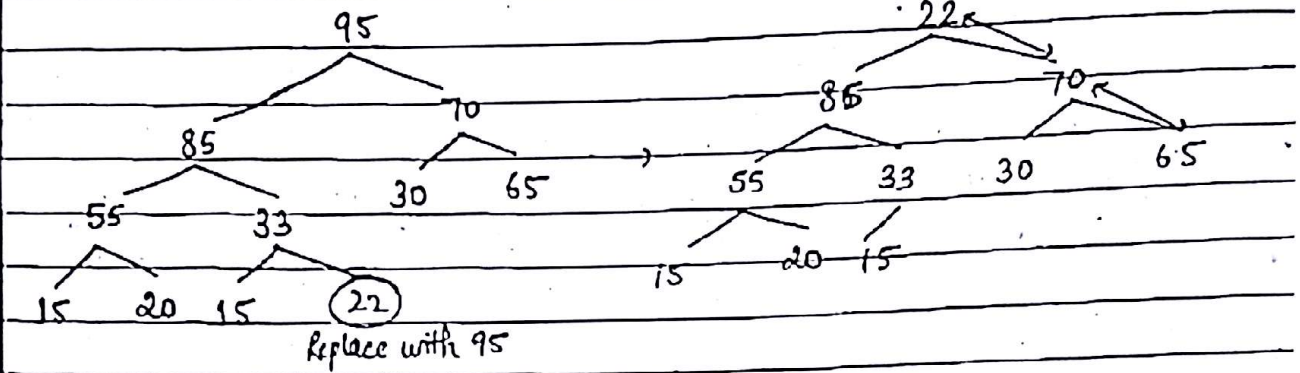




o/p = 90, 73, 56, 23, 21, 7, 3, 2, 1, 0.

Date

Ques) Perform deletion-



→ AVL Search Tree: Adelson, Velskii & Landi
(balanced Binary search tree)

$|h(T_L) - h(T_R)| \leq 1$ → Necessary condition

where $h(T_L)$ = height of left subtree $h(T_R)$ = height of right subtree

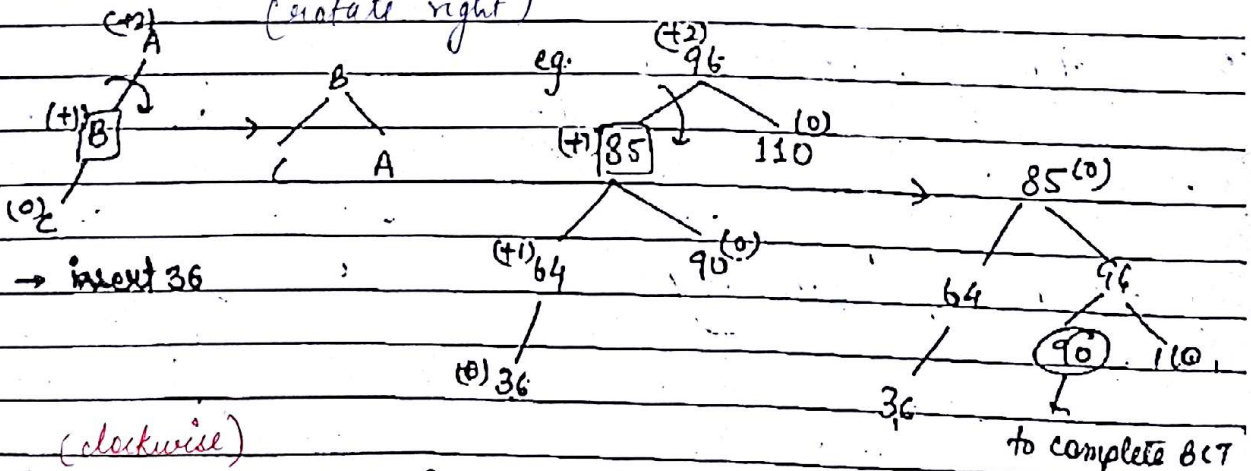
Balancing factor: -1, 0, +1 ⇒ balanced

Any other value of Balancing factor means Tree is not balanced.

At such points, rotation needs to be formed.

$BF = h(T_L) - h(T_R)$

1) RR Rotation — element inserted at left side of left child
(rotate right)



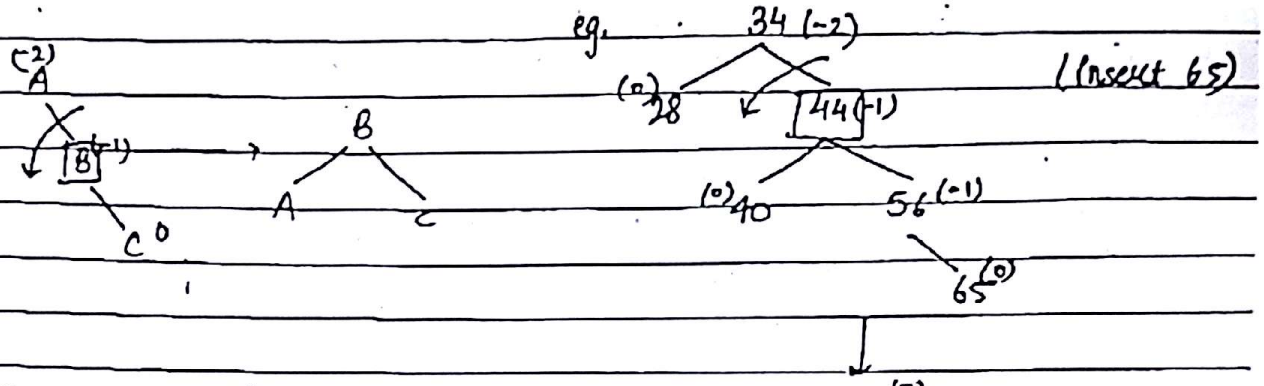
(clockwise)

cw rotation about the node immediately next to the node having the unbalancing factor (+2)

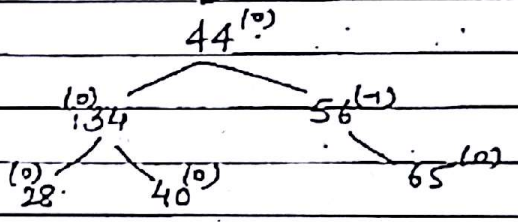
AVL Tree = Balanced binary search tree

Date

2) RR Rotation – element inserted on right side of right child



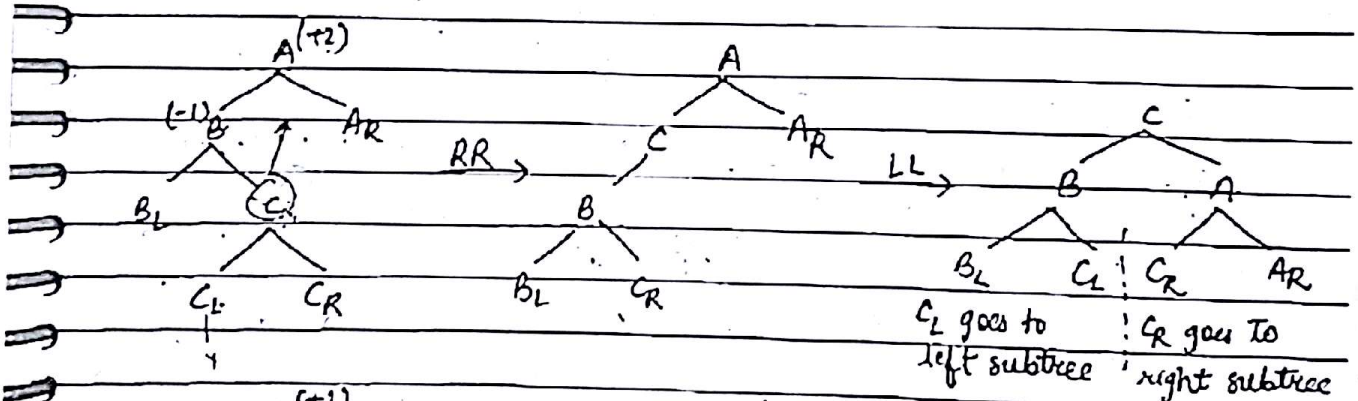
acw rotation
(anti-clockwise)

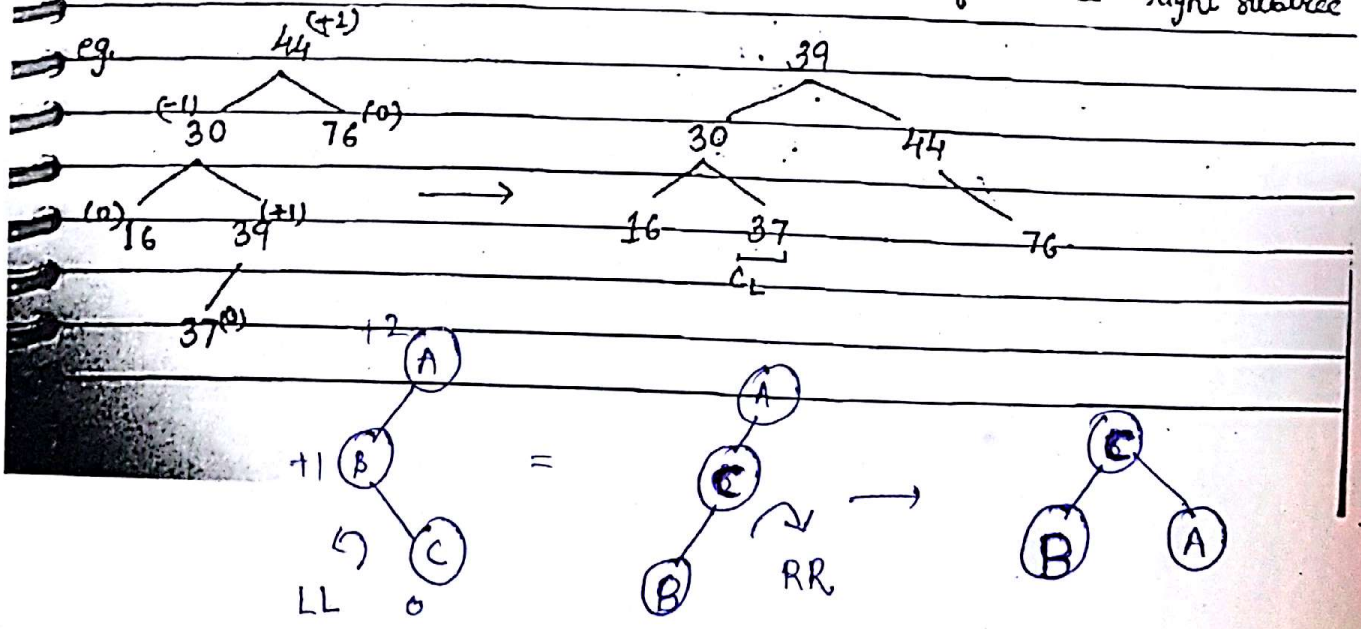


3) LR Rotation – Insertion at right of left

~~# RR followed by LL~~

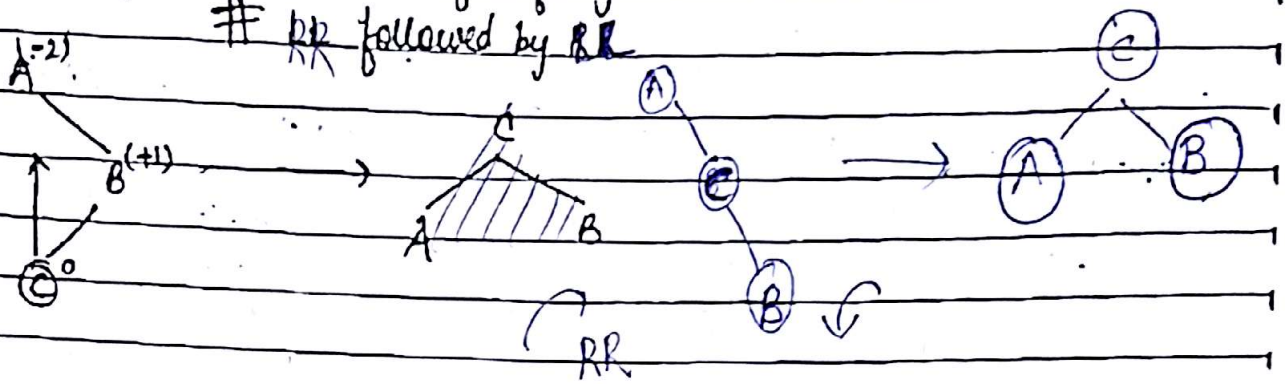
LR = ~~RR + LL~~
LL + RR



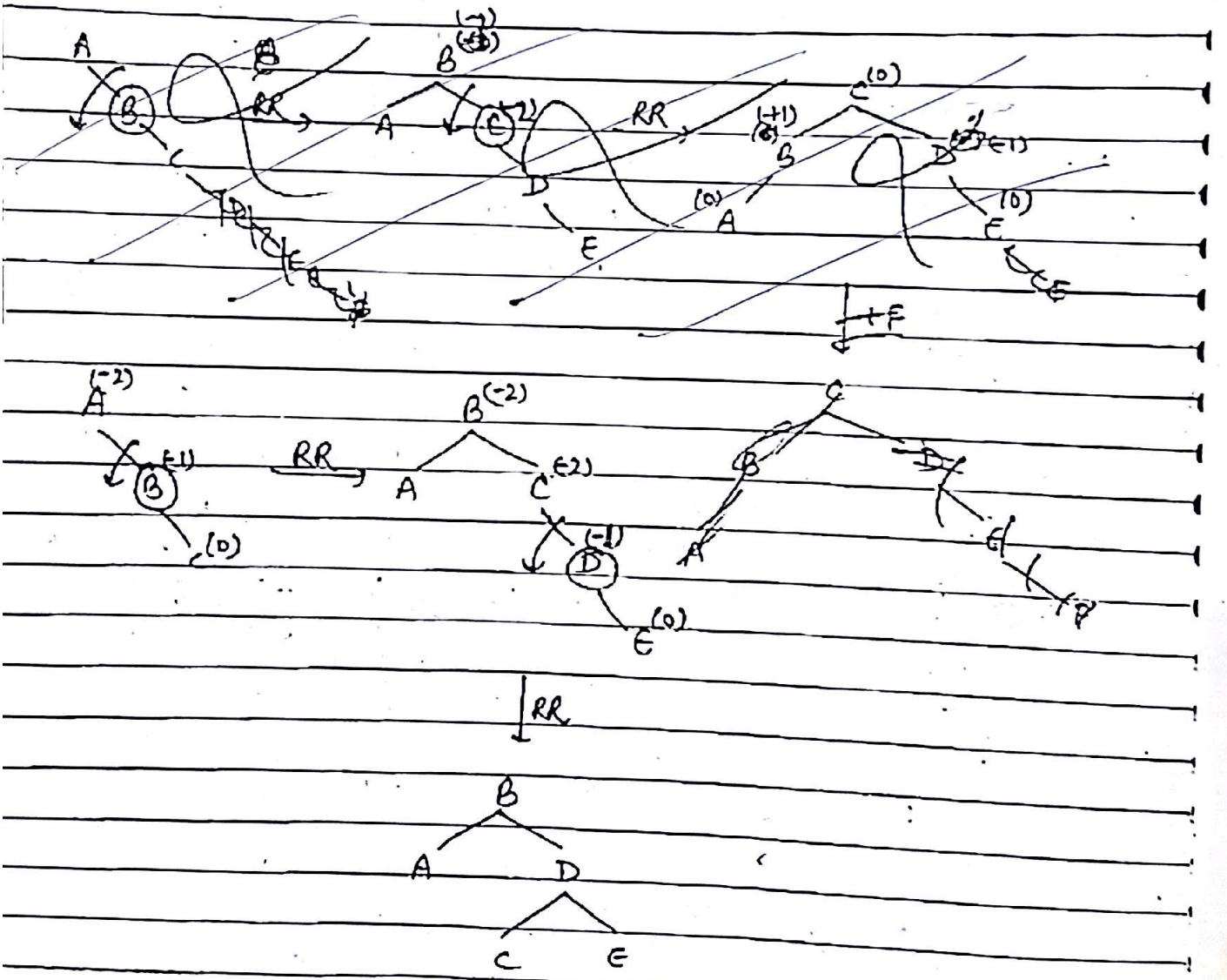


Date

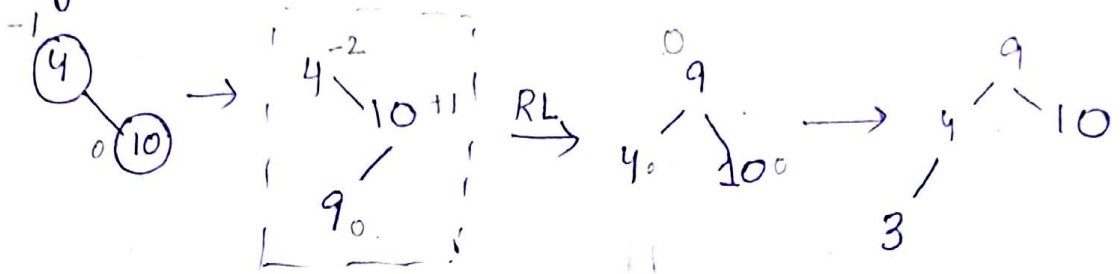
4) RL Rotation - insertion at left of right
RR followed by RL

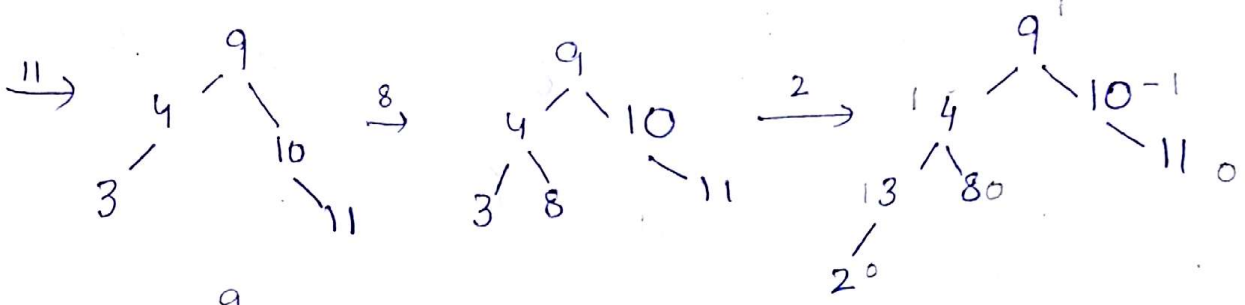


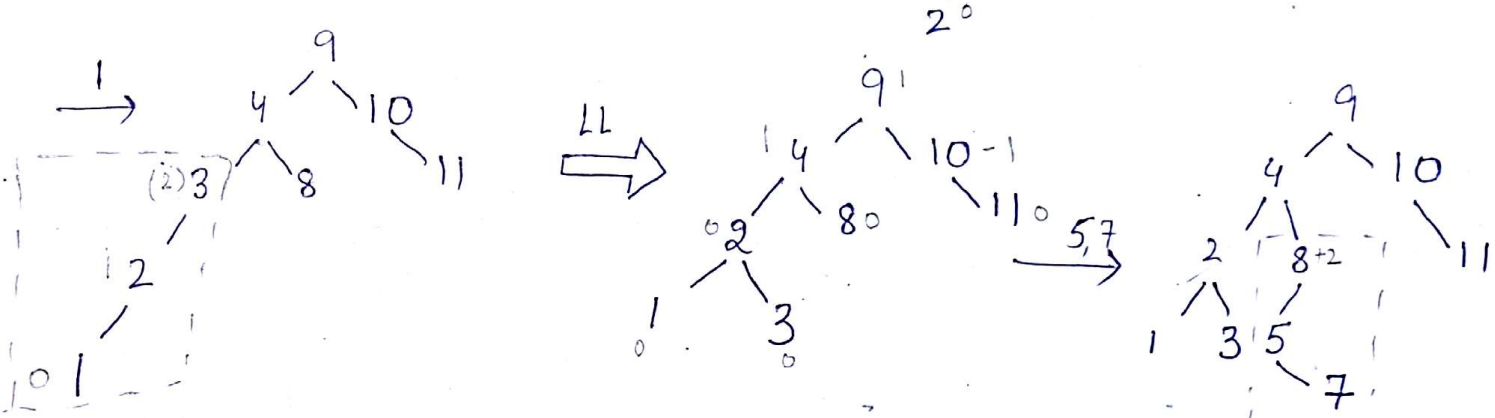
Ques) A, B, C, D, E → form an AVL Search Tree

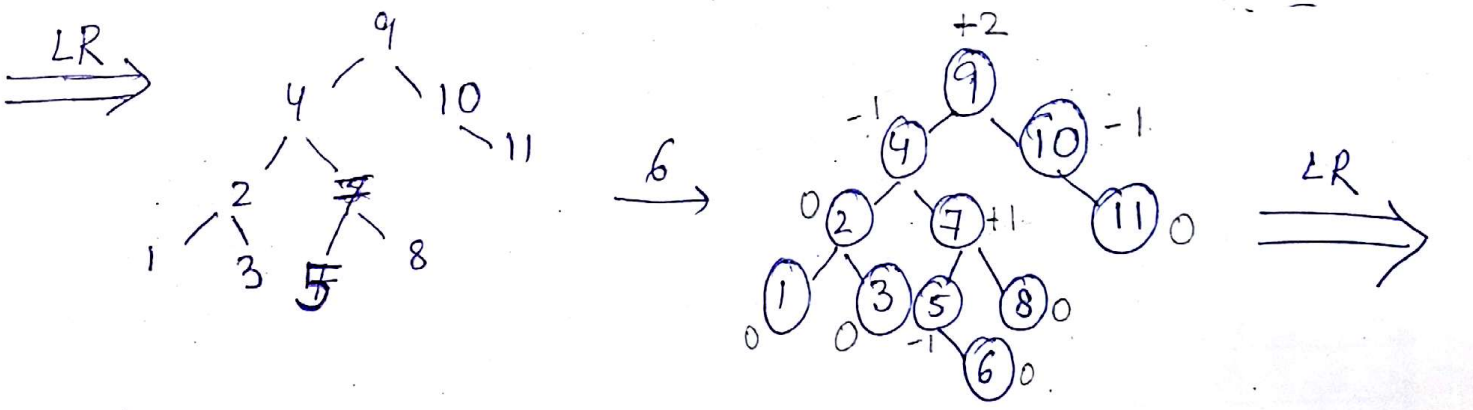


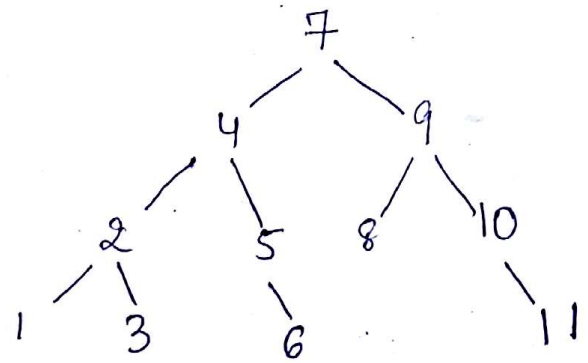
eg. 4, 10, 9, 3, 11, 8, 2, 1, 5, 7, 6











Q. eg. 55, 66, 77, 15, 11, 33, 22, 35, 25