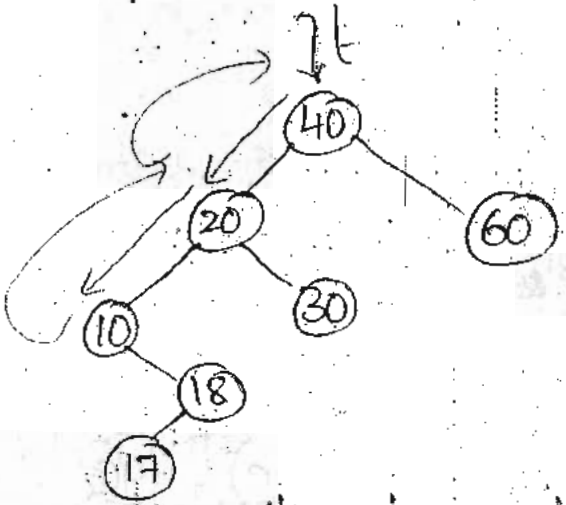
first

p) int DO (struct BTNode *t)
{
if (!t->LC)
return (t->data);
else
return DO(t->LC);
}

Sol:



It returns the minimum element in BST

obs: To get maximum element (greatest element) traverse descend right.

Sep

Oct Sep

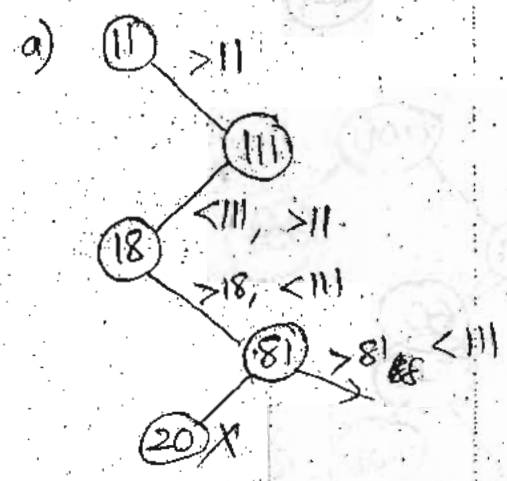
p) Identify valid/Invalid search sequences to search for 88 on a BST
(BST will not be given)

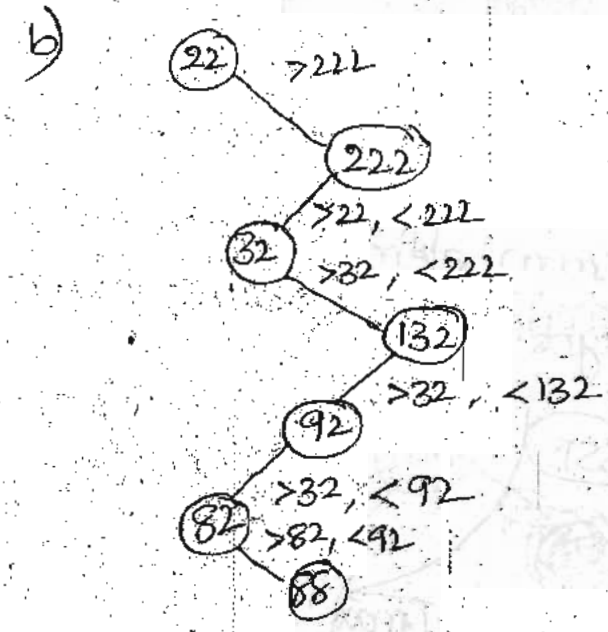
a) 11, 111, 18, 81, 20, 90, 60, 70, 88 — Invalid

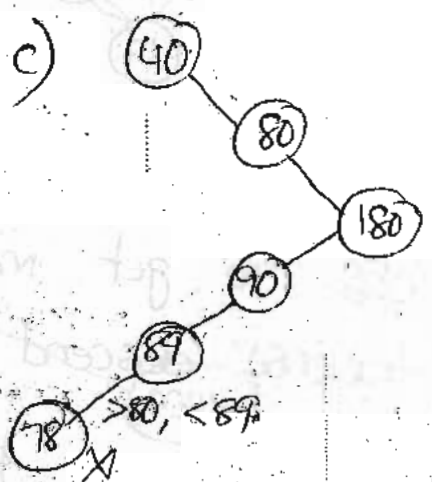
b) 22, 222, 32, 132, 92, 82, 88 — valid

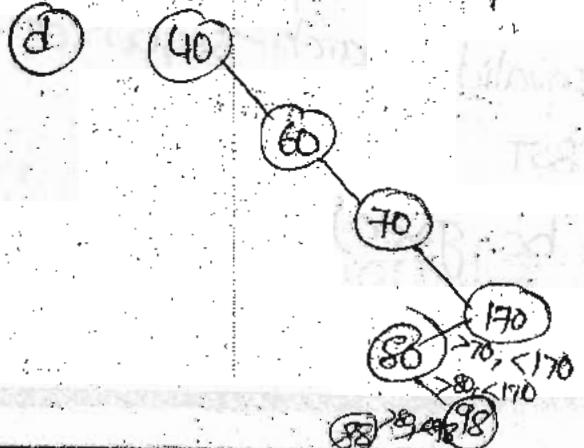
c) 40, 80, 180, 90, 89, 18, 88 — Invalid

d) 40, 60, 70, 170, 80, 98, 88 — valid

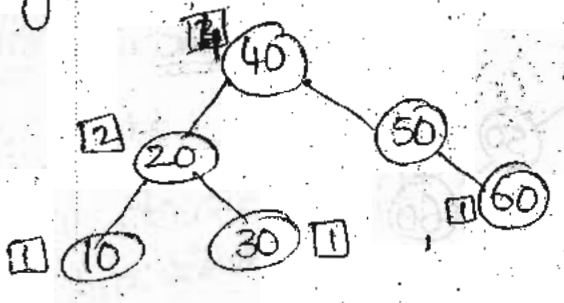








Q what is the Index (rank) of node 50 in the following BST



Sol:

Index = No of nodes in left subtree + 1

∴ Index of node 50 = 0 + 1 = 1

Note:

Index indicates the position of the node in the inorder (of root)

Deletion of node from BST:

Even after the deletion, the order must be intact

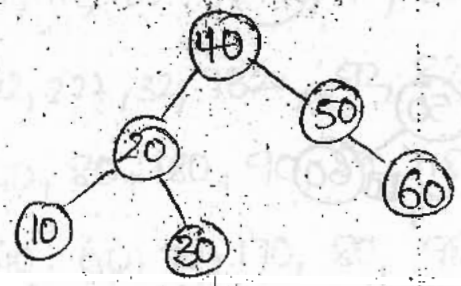
i.e remains unchanged

i.e Tree can be modified but not order

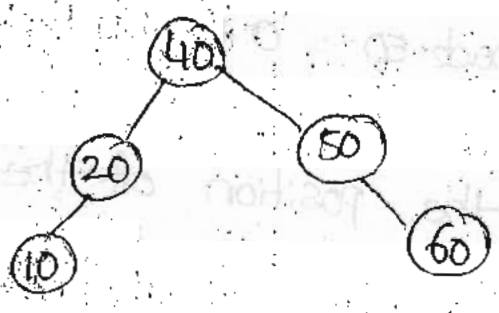
Procedure:

A Node to be deleted	Action
case(i): No children	i) simply delete
case(ii): only one child (or) subtree	ii) connect to its grand parent
case(iii): Both children (or) subtrees	Replace with a) Inorder successor (or) min of Right subtree (RST) } Default b) Inorder predecessor (or) max of Left Subtree (LST)

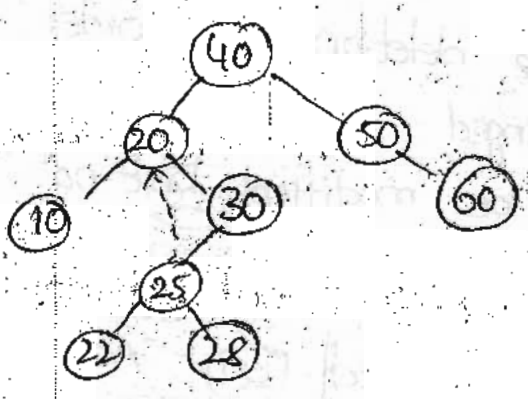
Eg



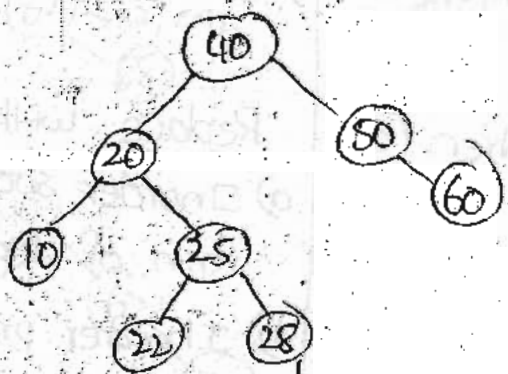
Case(i): Delete 30



Case(ii): ~~de~~ ~~de~~ consider



Delete 30

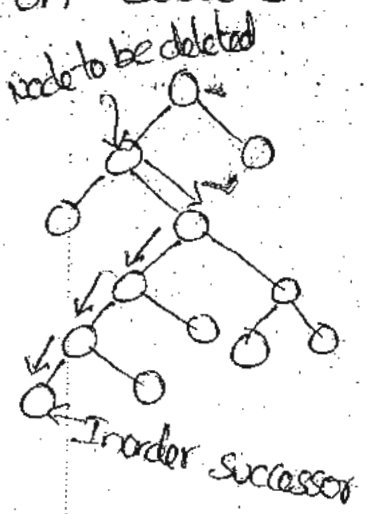


Case(iii):

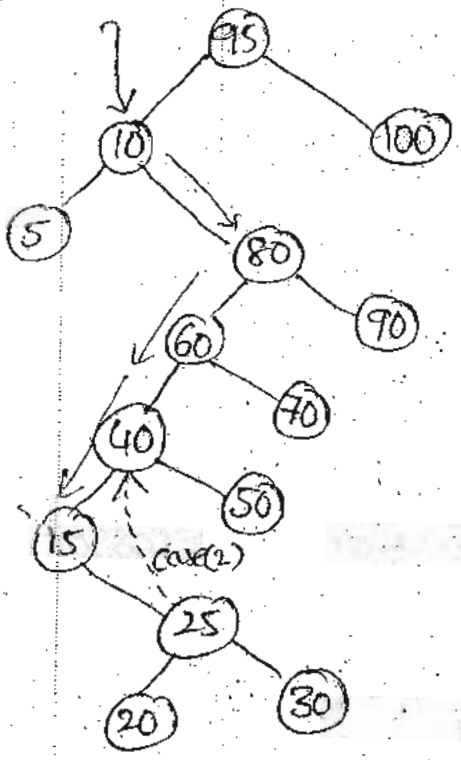
Logic for Inorder successor:

1. Jump right
2. Go on descend Left

Eg:

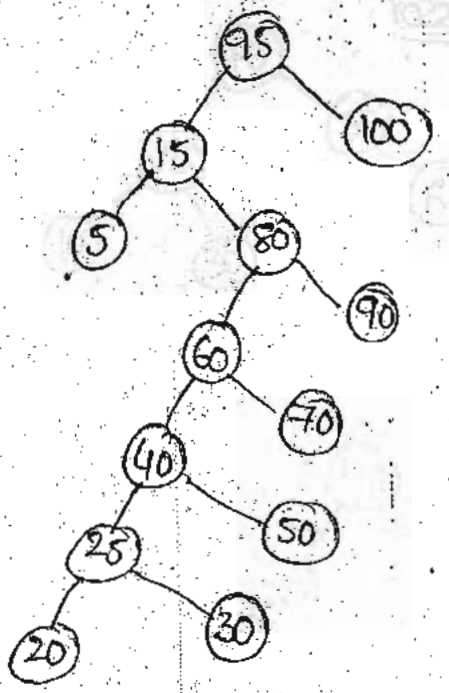


Eg:



Delete 10.

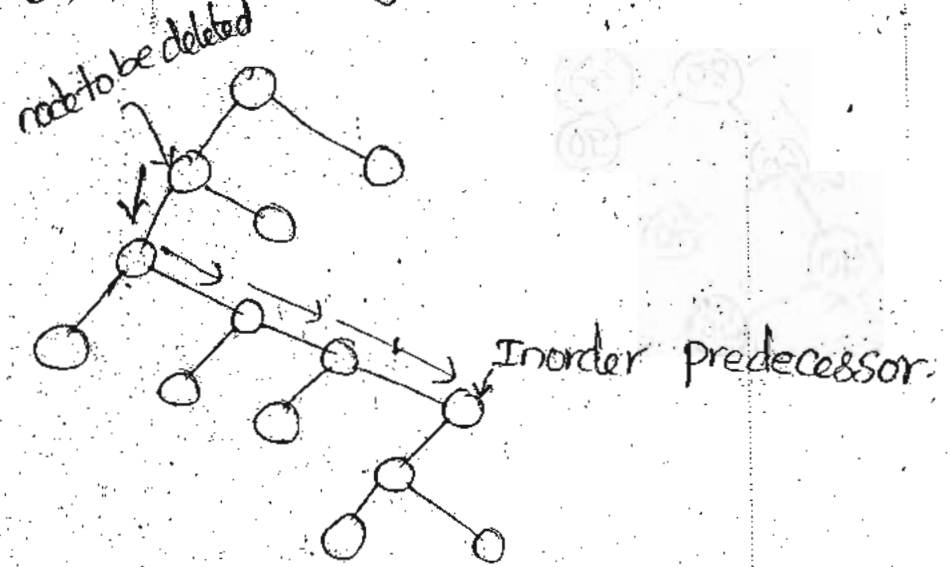
Inorder Successor for '10' is '15'



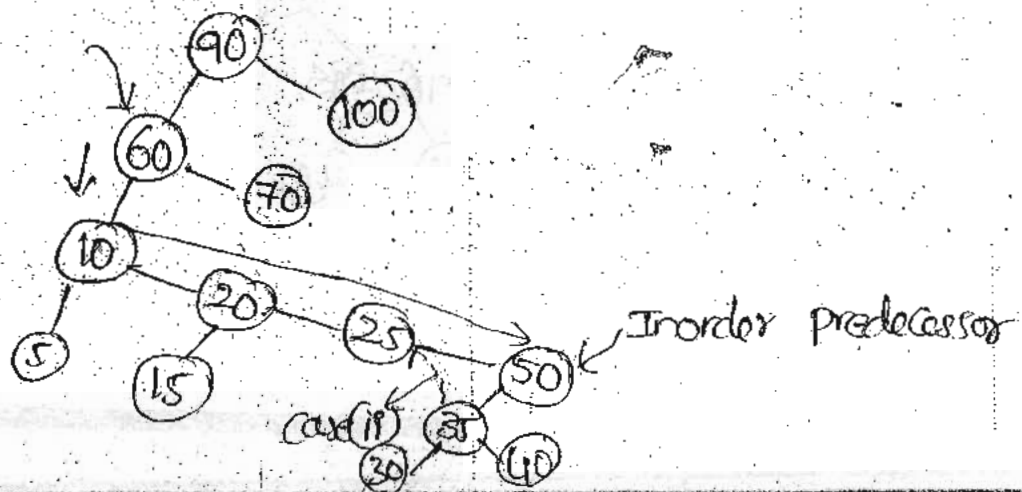
Logic for Inorder predecessor:

1. Jump left
2. Go on descend right

Eg:

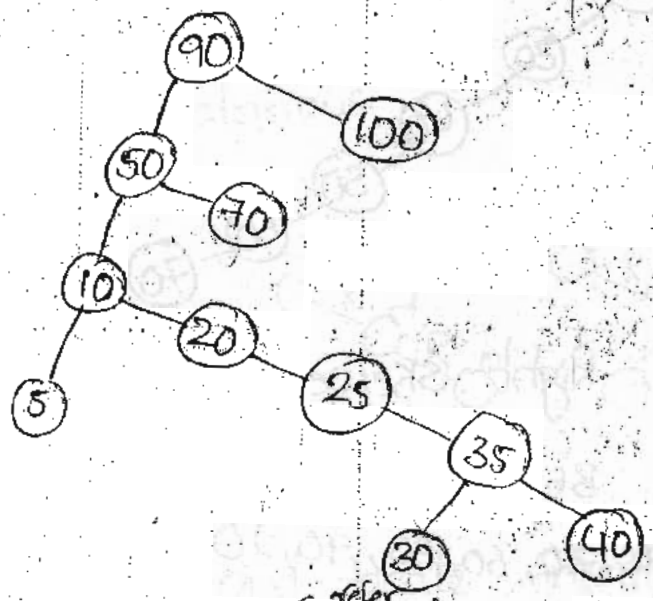


Eg:



Delete 60

Sol: Inorder predecessor for node 60 is node 50.



AVL Search Trees: (refer Kruse)

→ Devised by Adelson, Velskii, Landis } Russian mathematicians (1962)

→ Height Balanced Trees

Definition:

BST with balance factor $bf(N) = H_L - H_R = \begin{cases} +1 \\ 0 \\ -1 \end{cases}$

H_L - Height of left subtree

H_R - Height of right subtree

→ $bf(N) = \pm 2$ ← violated node = Ancestor

Need to go for AVL Trees:

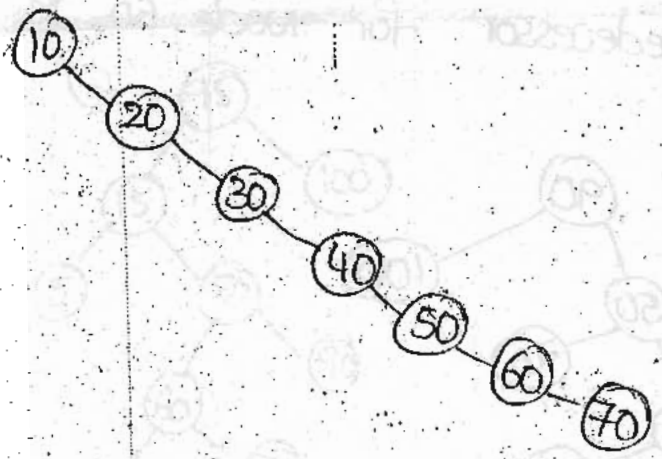
create BST

10, 20, 30, 40, 50, 60, 70

Sol: Search for 70 takes 7 attempts

for 7 nodes → 7 attempts

for 'n' nodes → O(N) attempts

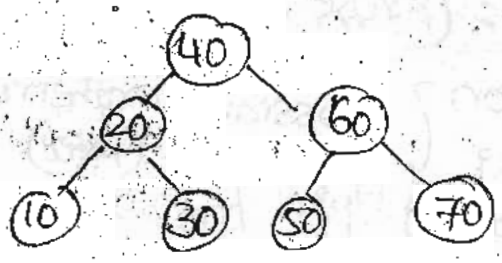


Height = Max. depth

Right skewed BST

Eg: create BST

40, 20, 30, 60, 50, 70, 10



for 70 node → 3 attempts

for 7 nodes ⟹ 3 attempts

$= \log_2 7 \approx \log 8 = 3$

- So skewed BST becomes bushy BST (full) by rotations

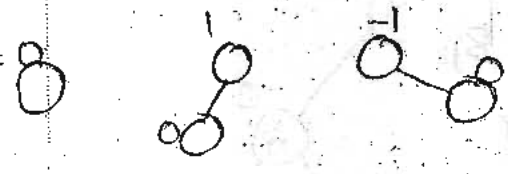
Goal of AVL Trees:

- minimise search times

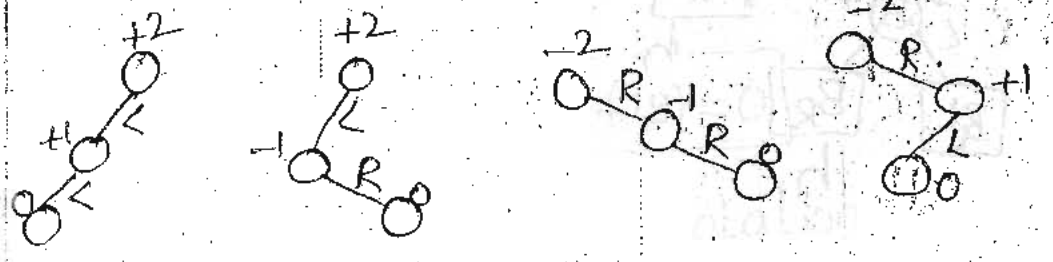
$O(N) \longrightarrow O(\log N)$

- It can be achieved by using rotations.

Valid: No need to apply rotations.

bf = 

Invalid: Apply rotations.



Nomenclature:

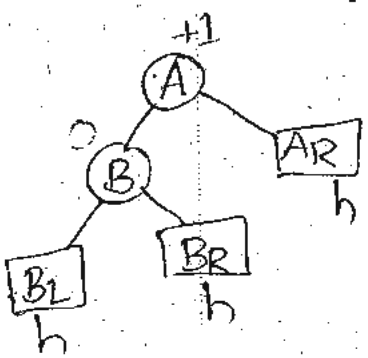
1. Insert a node and locate Ancestor (A)
2. Traverse from (A), towards the newly inserted node
3. consider only TWO hops from (A) to give nomenclature of LL, LR, RR, RL.

Rotations:

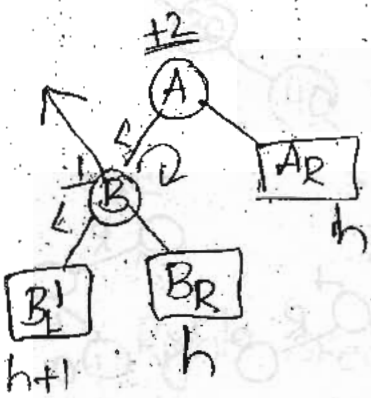
① **LL Rotation**: single rotation

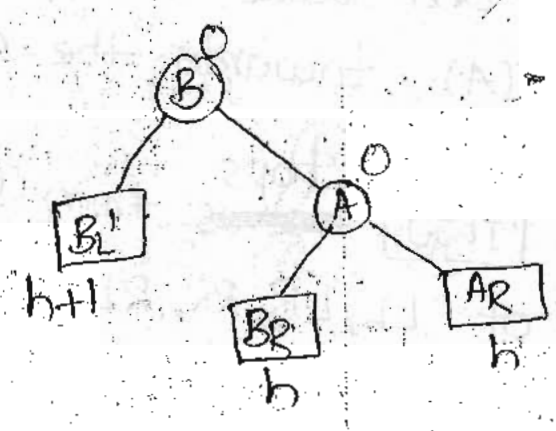
○ Node
□ subtree

a) Initial



(b) Insert a node in B_L

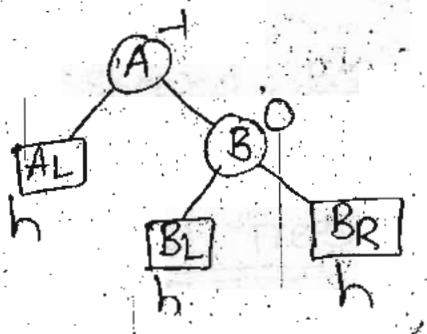




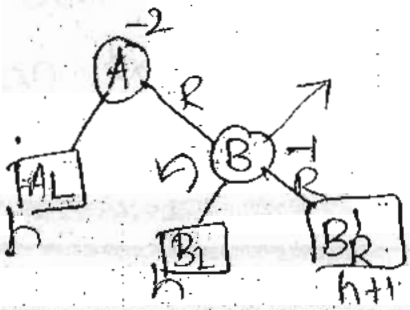
$\therefore bf(A) = bf(B) = 0$

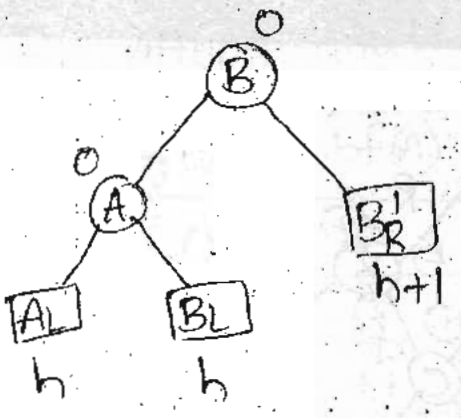
(2) RR rotation: Single rotation

(a) Initial



(b) Insert a node in B_R





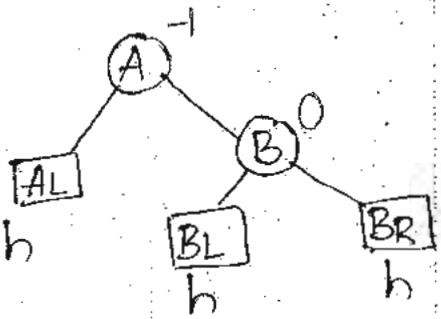
$\therefore$ bf(A) = bf(B) = 0

③ RL rotation: Double rotation

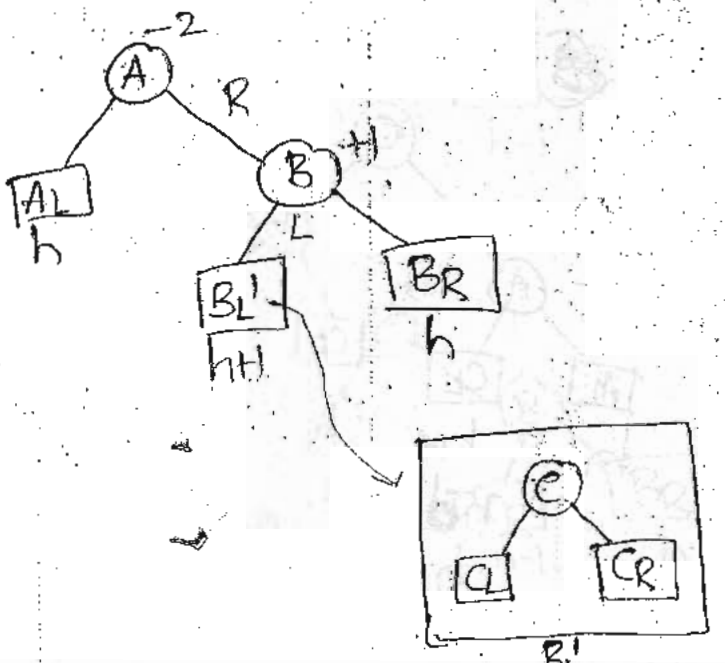
RL
LL
RR

=0

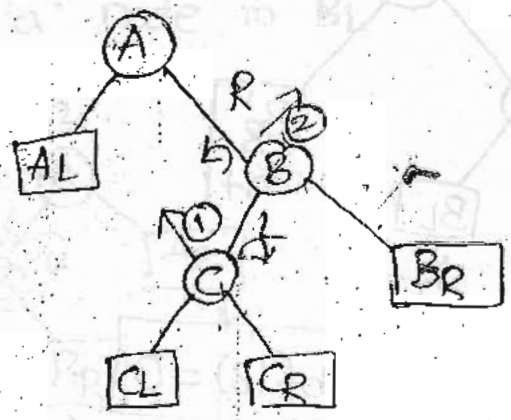
ⓐ Initial



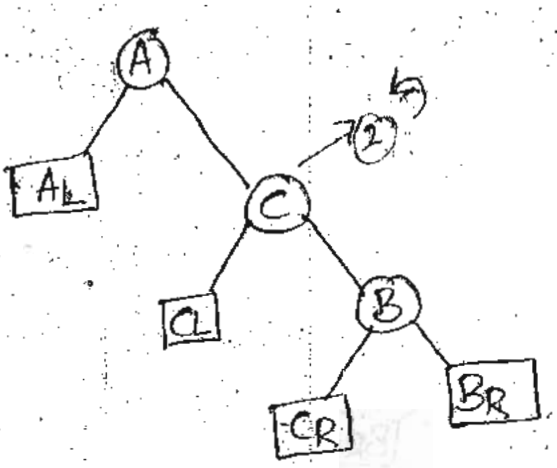
ⓑ Insert a node in B_L



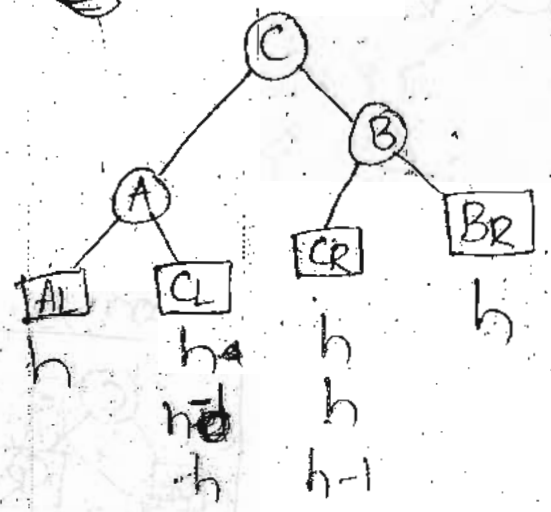
i.e.



↓ LL



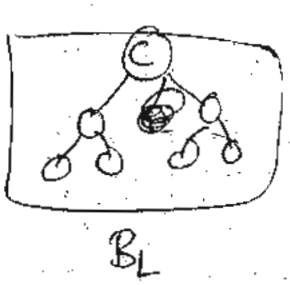
↓ RR



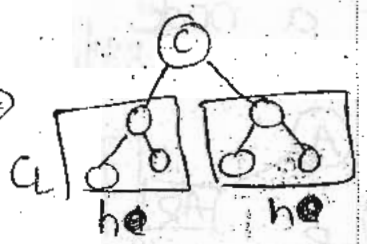
Before rotation	After rotation →	
bf(C)	bf(B)	bf(A)
0	0	0
+1	−1	0
−1	0	+1

} bf(C) = 0

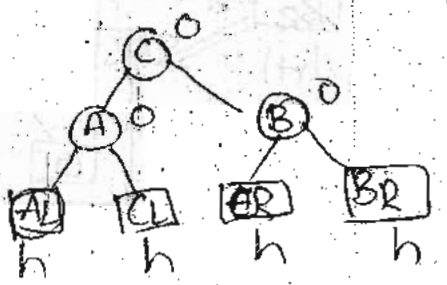
Sol: 1) bf(C) = 0



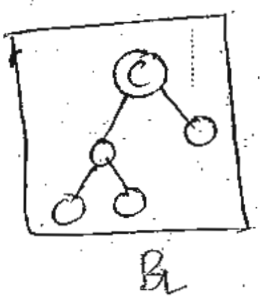
⇒



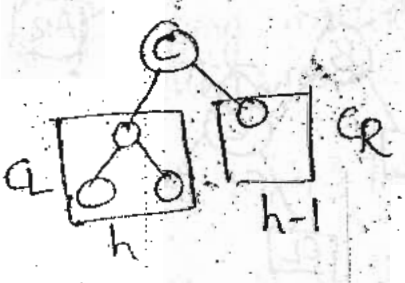
∴ bf(B) = bf(A) = 0

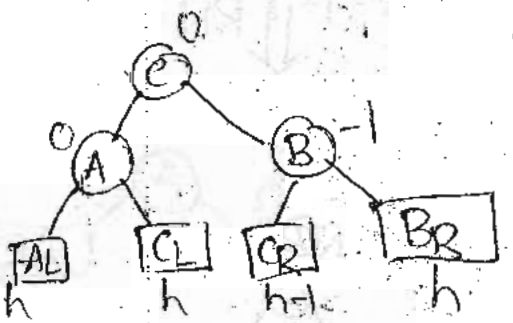


2) bf(C) = 1



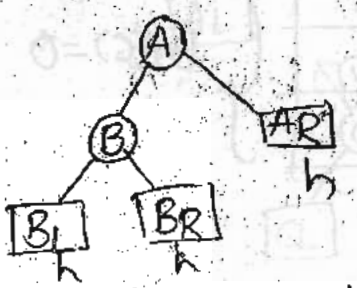
⇒

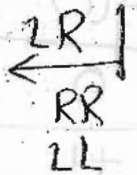




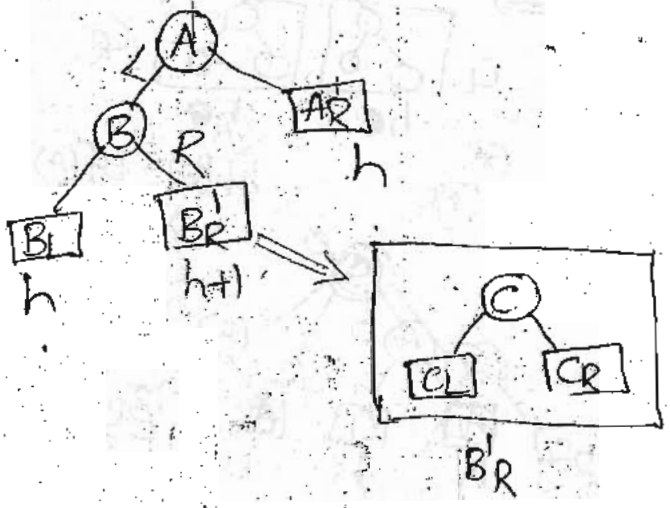
4) LR Rotation: Double rotation

a) Initial

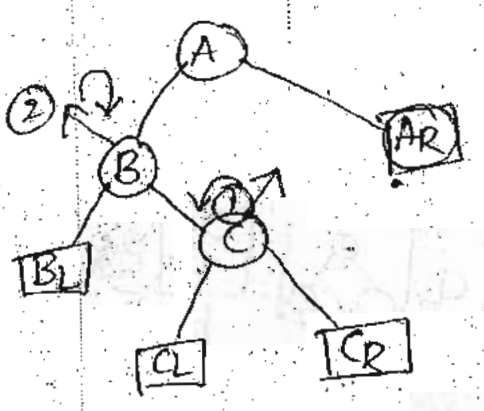




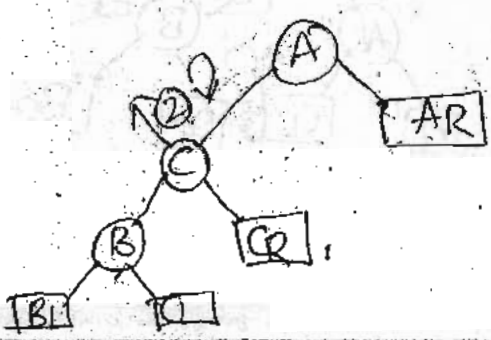
b) Insert a node in BR



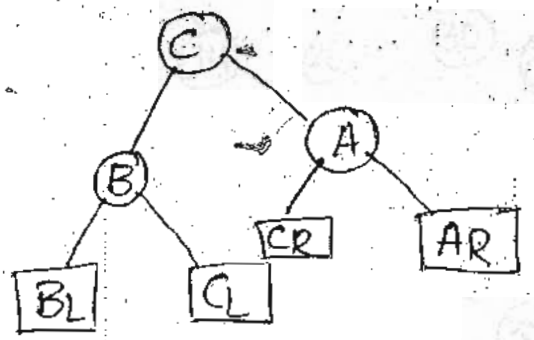
i.e.



RR



↓↓ LL



Before rotation	After rotation →		
bf(C)	bf(B)	bf(A)	
0	0	0	bf(C)=0
+1	0	-1	
-1	+1	0	

16/8/11

Creation of AVL Tree:

Procedure:

1. Insert a node as per BST
2. configure
   if imbalance
   balance it by applying rotations.
   with the nodes

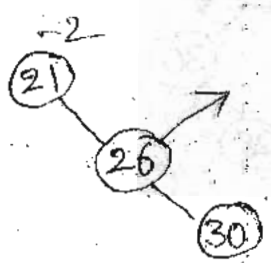
P) create AVL Tree

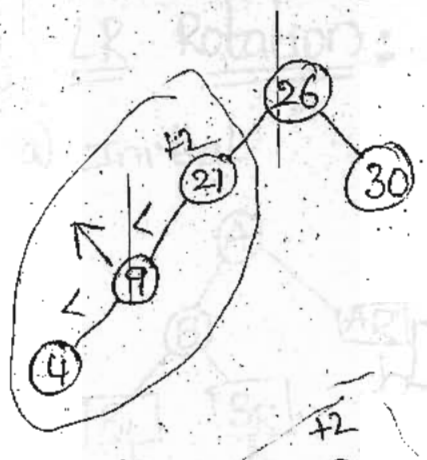
21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7

Sol):

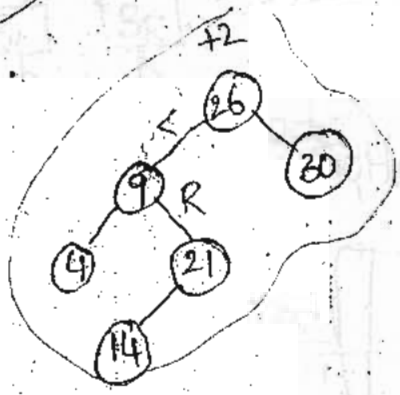
RR LL LR RL RL RL LL LR LR

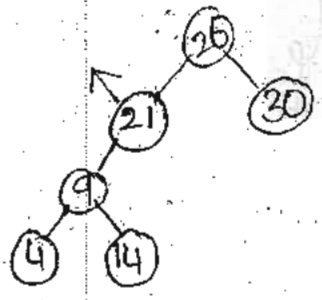
21, 26, 30 | 9, 4 | 14 | 28 | 18, 15 | 10 | 2 | 3 | 7 |

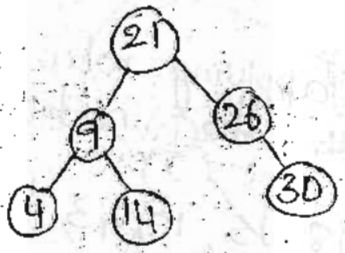




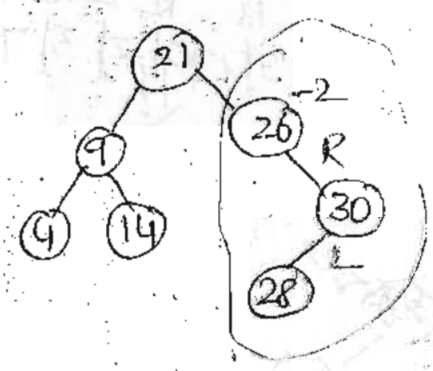
Insert: 14

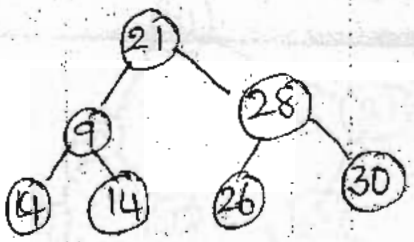




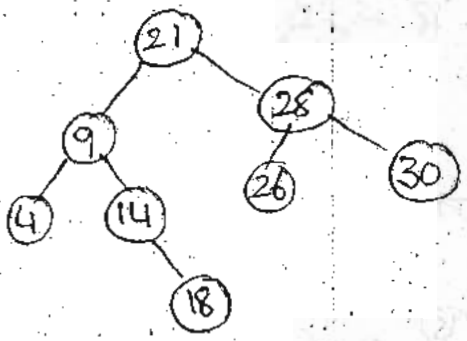


Insert: 28

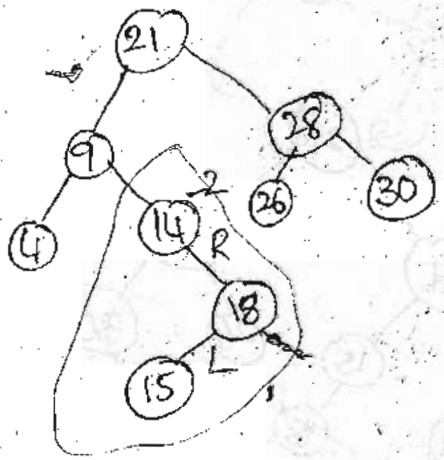


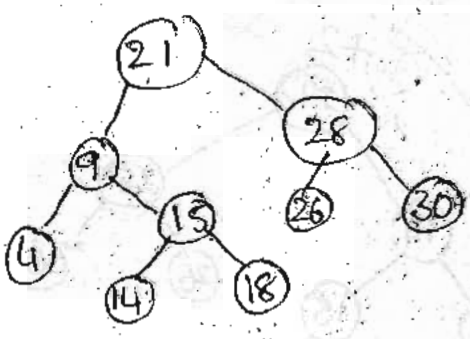


Insert : 18

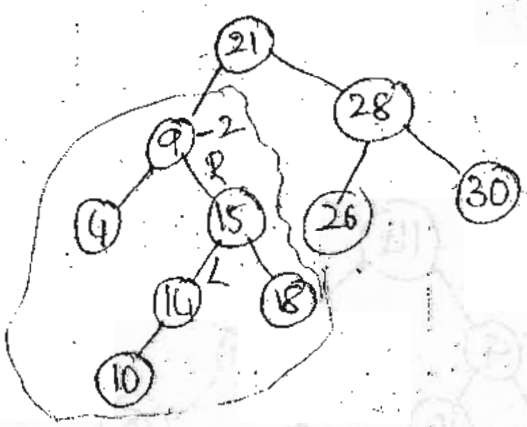


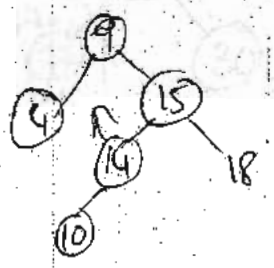
Insert : 15

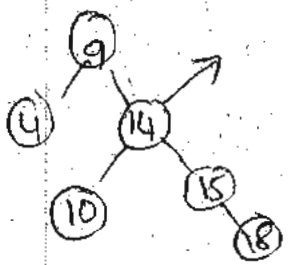


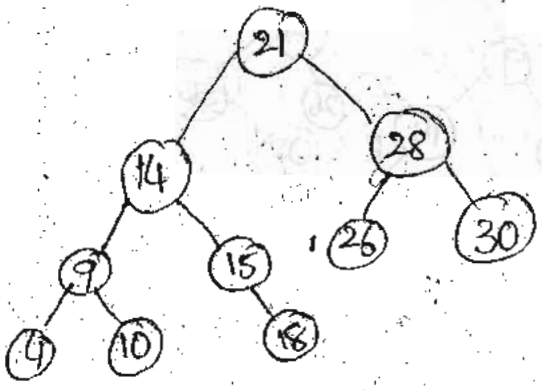


Insert : 10

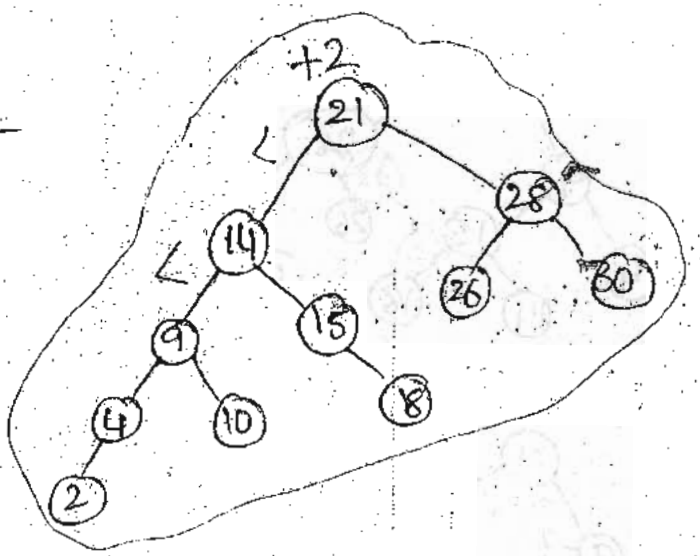


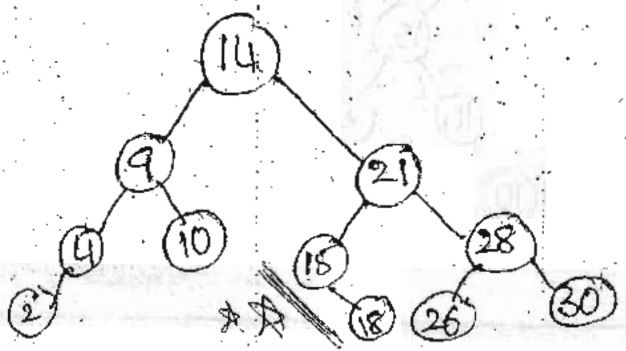

9
4
15
14
18
10


9
4
14
10
15
18

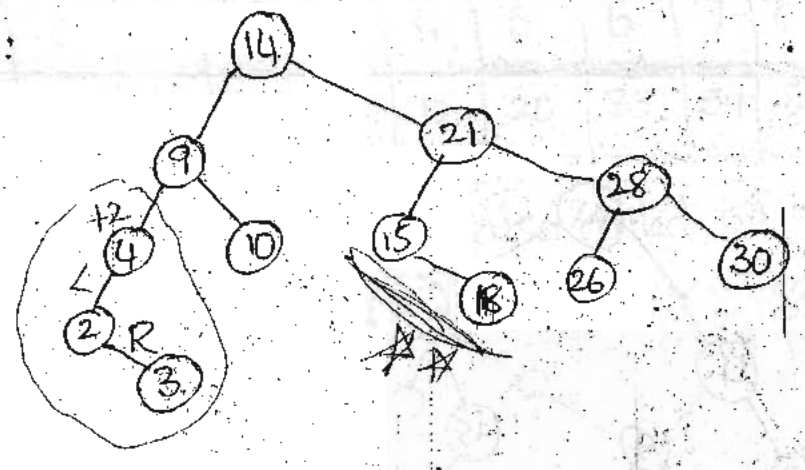

21
14
28
9
15
26
30
4
10
18

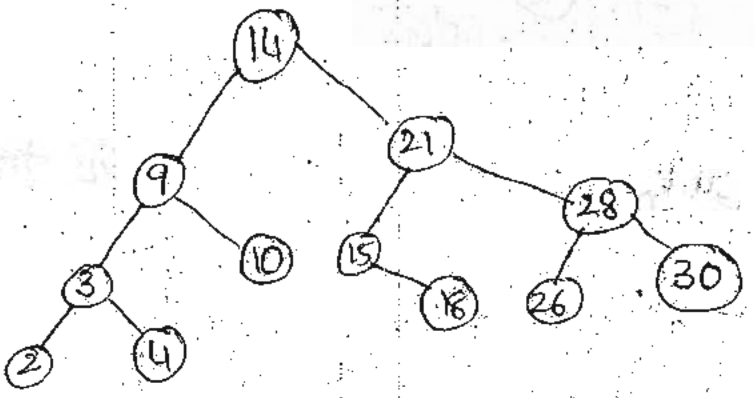
Insert: 2


+2
21
28
14
26
30
9
15
4
10
8
2

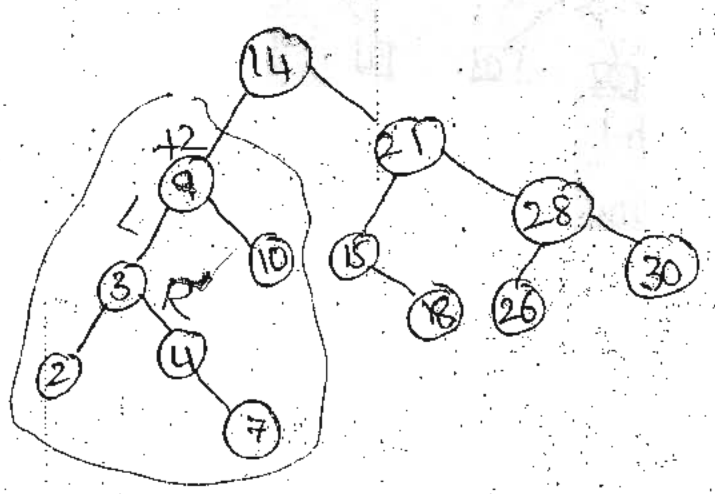

14
9
21
4
10
15
28
2
18
26
30

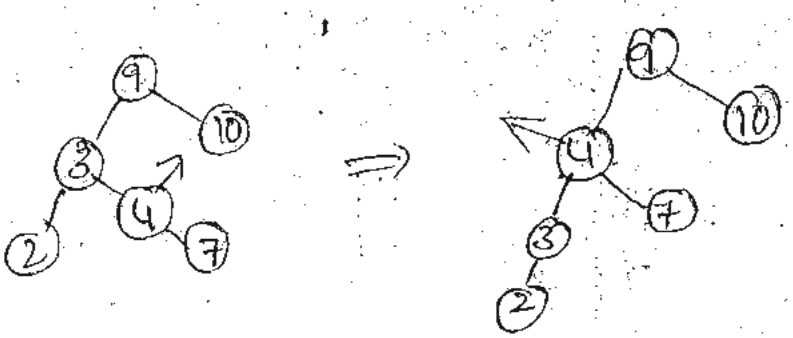
Insert 3:

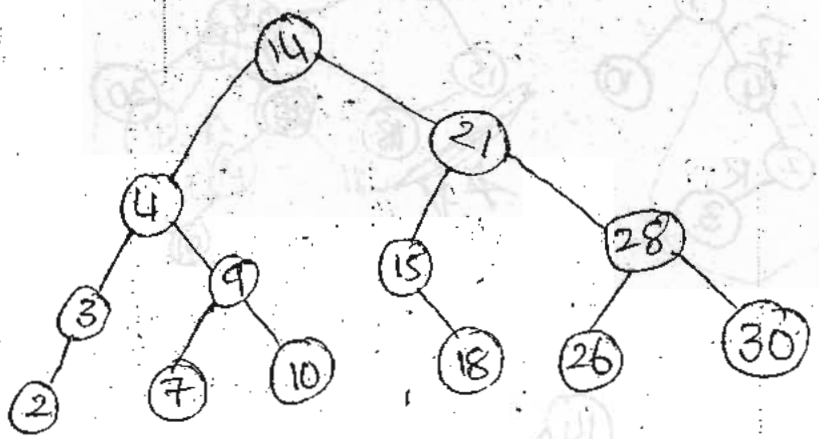




Insert 7:

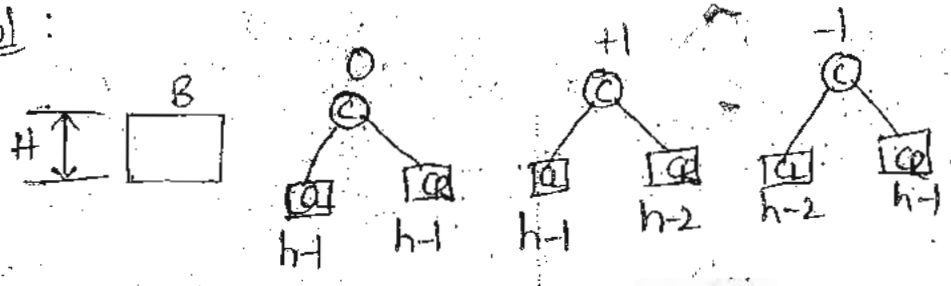






b) calculate min no. of nodes in AVL tree of height h=8 (level starts at 0)

Sol:

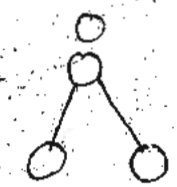


L=0 → 0

N(0)=1

L=1 ⇒ L=0

L=1

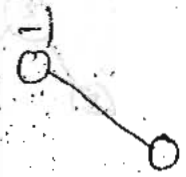


N(1)=3

X

This is not considered

N(1)=2



N(1)=2

$$N(H)_{min} = \begin{cases} 1 & L=0 \\ 2 & L=1 \\ 1+N(H-1)+N(H-2) & L>1 \end{cases}$$

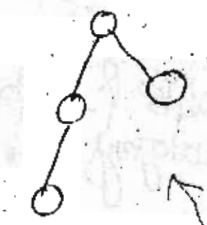
looks like fibonacci series

L=H=D	0	1	2	3	4	5	6	7	8
N(H) min	1	2	4	7	12	20	33	54	88

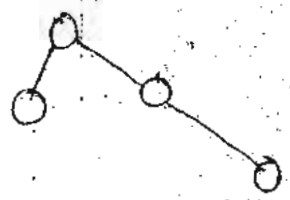
Eg: the following AVL trees also called as fibonacci heaps

collection of trees not heapsort heap

1) L=0, L=1, L=2



(or)



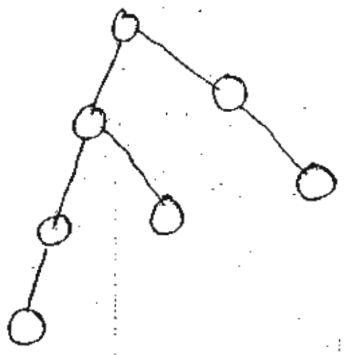
N(2) = 4

Left biased

N(2) = 4

Right biased

2) L=0, L=1, L=2, L=3



N(3) = 7

(or)

N(3) = 7

AVL deletion:

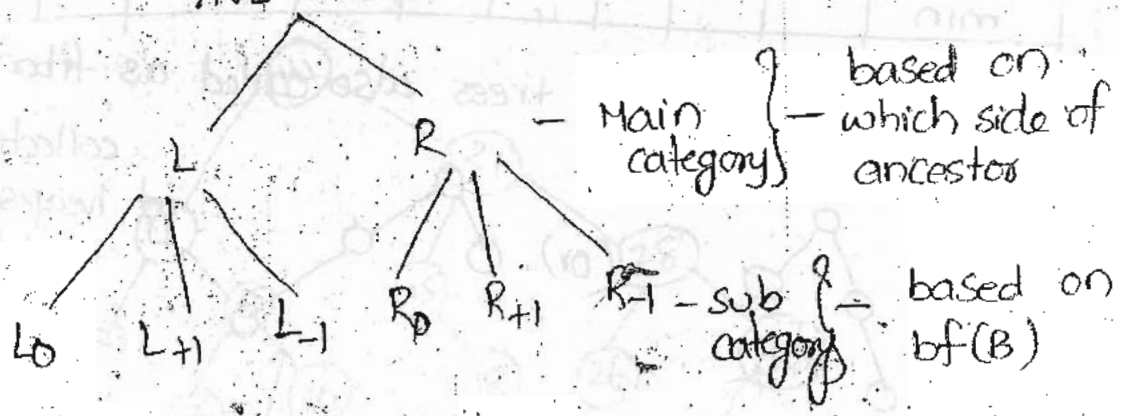
Note:

- A single insertion may cause the maximum no. of two rotations
- A single deletion may cause the maximum no-of multiple rotations (worst case)

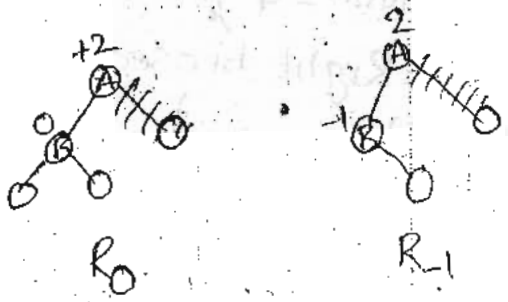
$O(\log_2 n)$

- Every insertion and every deletion may not cause rotation

AVL Deletion

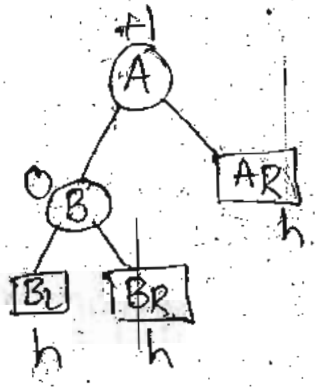


Eg:

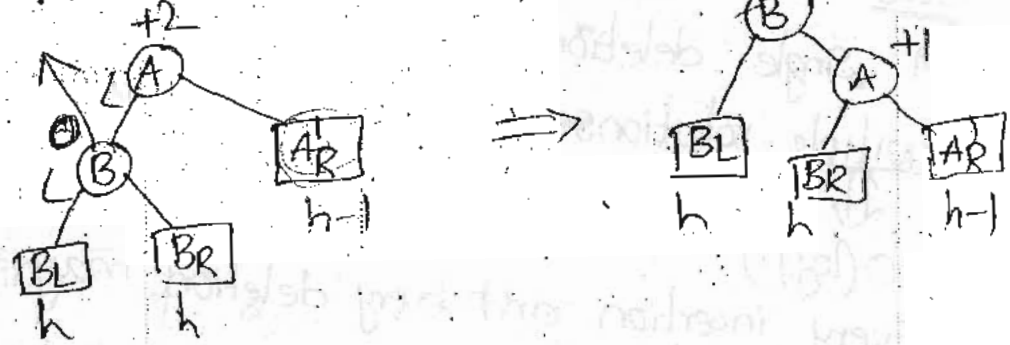


R_0 :

a) Before deletion

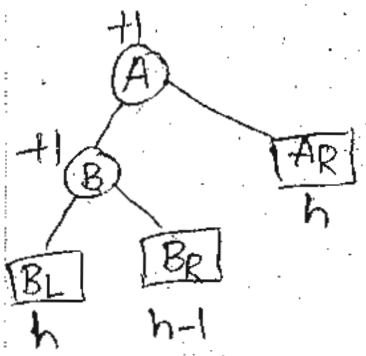


b) Deleting a node in A_R

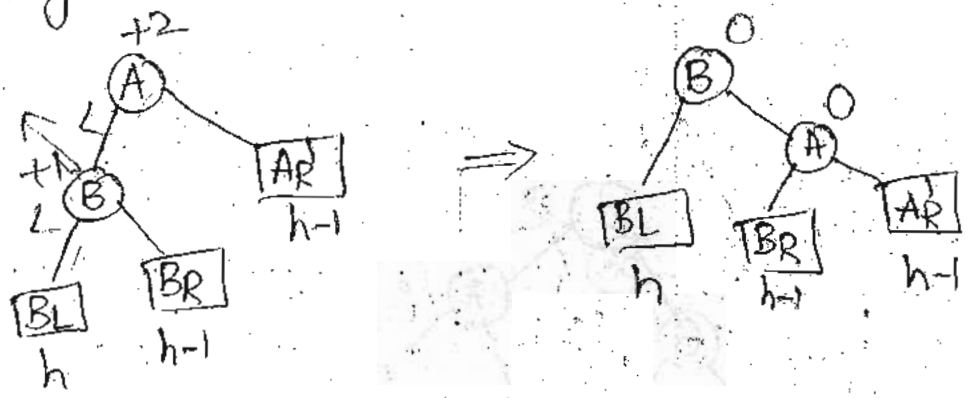


R_1 :

a) Before deletion

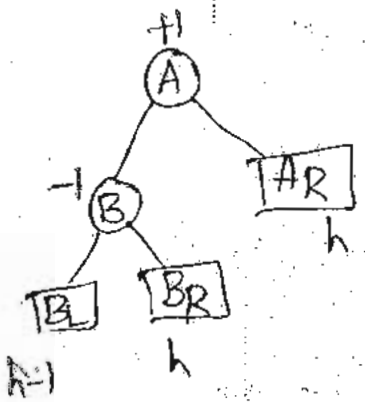


b) Deleting a node in AR

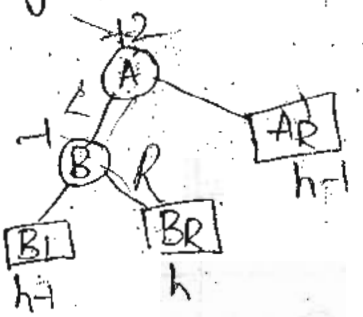


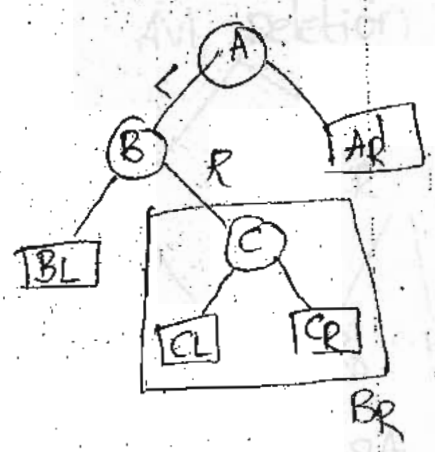
R_{-1} :

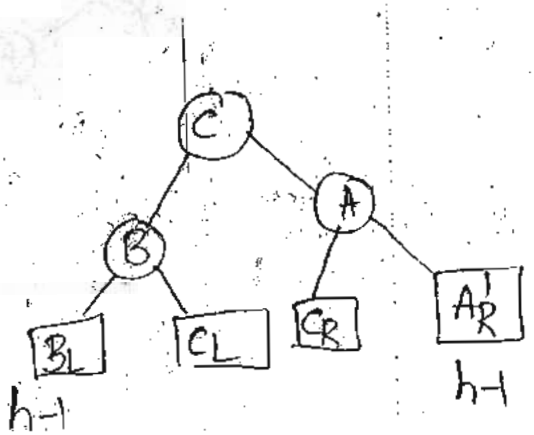
a) Before deletion



b) Deleting a node in AR







(7)

→

	Similar to	bf(B)	bf(A)	Remark 100% same
R_0	LL	−1	+1	NO
R_{+1}	LL	0	0	Yes
R_{-1}	LR	bf(C) < +1, 0, −1		

	Similar to	bf(B)	bf(A)	Remark 100% same
L_0	RR	+1	−1	NO
L_{+1}	RL	bf(C) < +1, 0		
L_{-1}	RR	0	0	Yes

17/8/11

AVL deletion:

Procedure:

1. Delete a node as per BST
2. configure
   If imbalance
   then balance it by applying rotations.

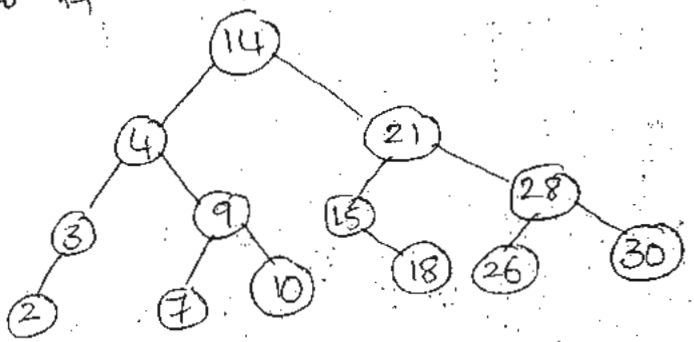
multiple rotations

Eg: Delete the following nodes

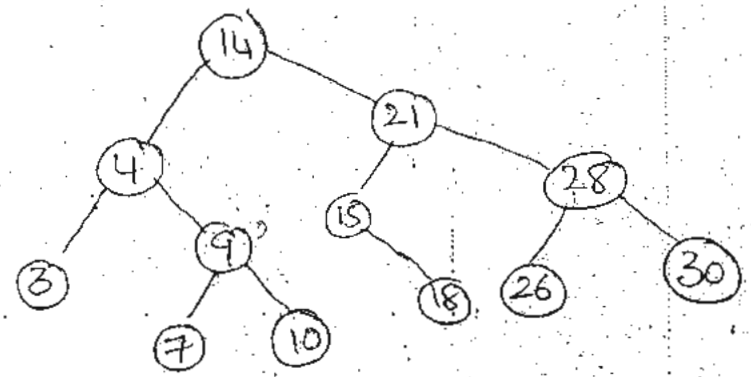
2, 3 | 10 | 18, 4, 9 | 14, 7, 15

L0 R1 L1

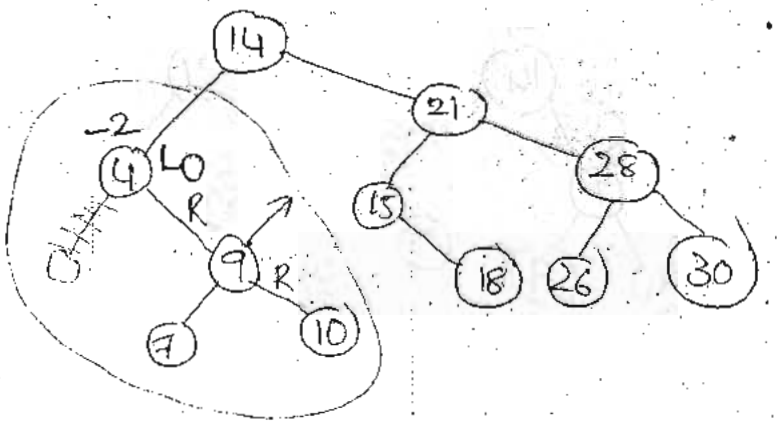
Sol:

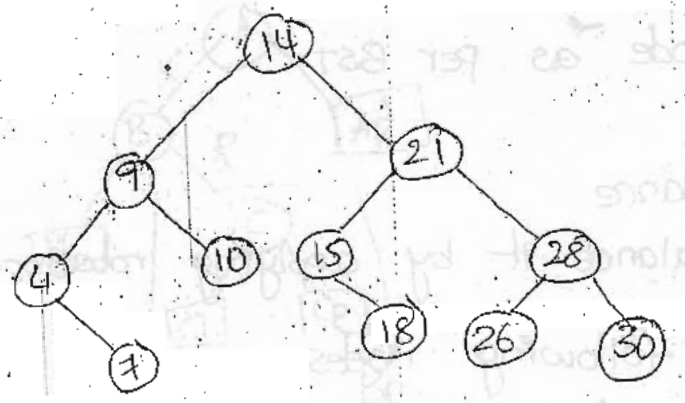


Delete 2:

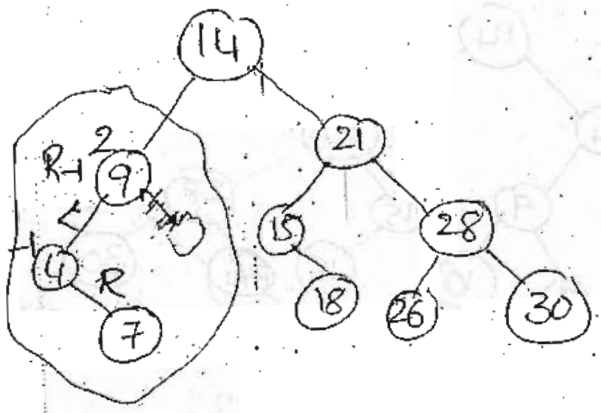


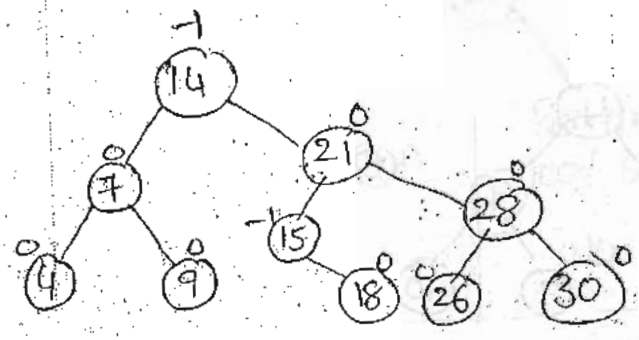
Delete 3:



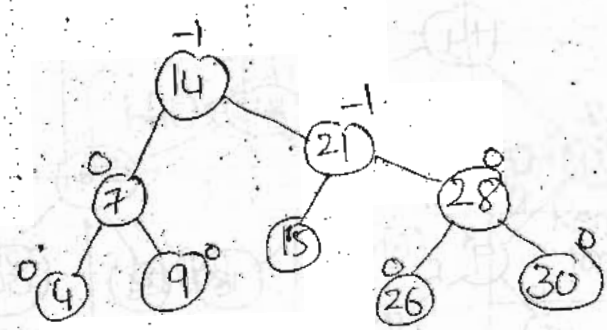


Delete 10:

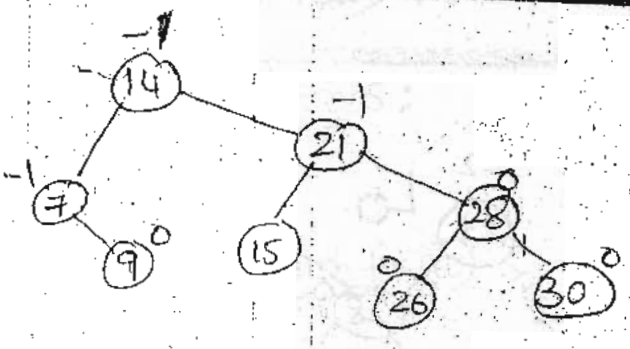




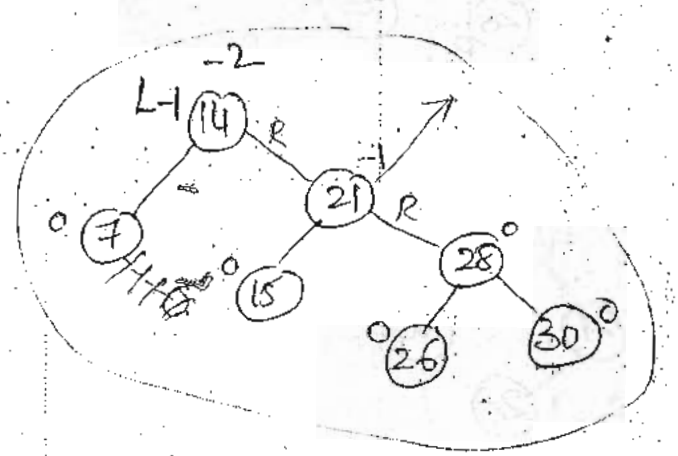
Delete 18:

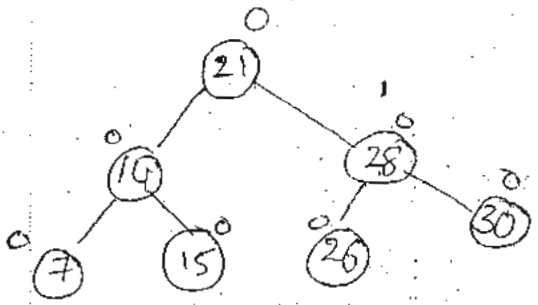


Delete 4:

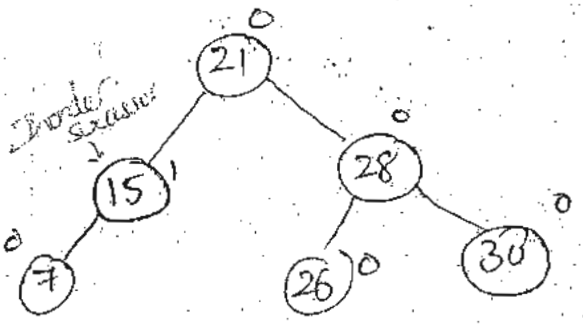


Delete 9:

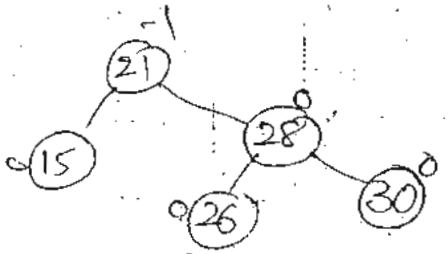




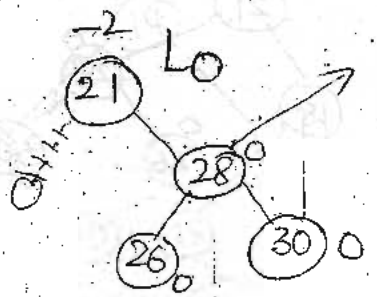
Delete 14:

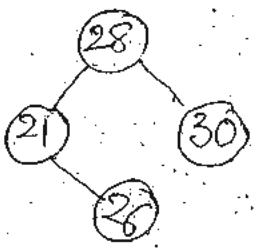


Delete 7:



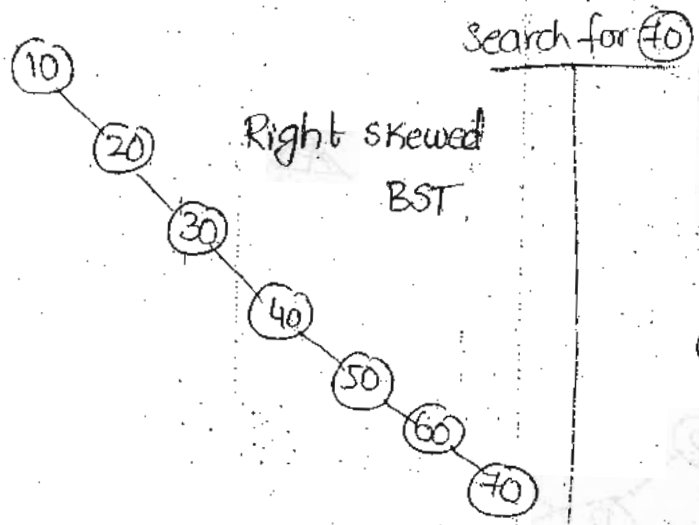
Delete 15:





splay trees :- (weiss)

Need to go for splay trees :

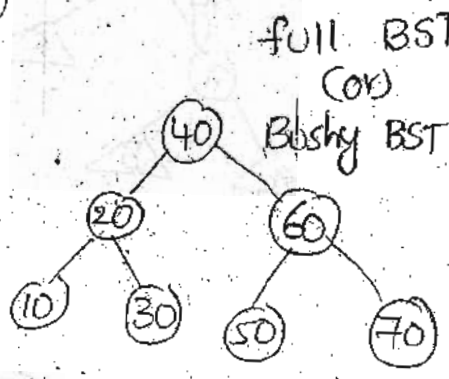


1 time ⟶ 7 attempts

splay @ root

999 times x 1 attempt ⟶ 999 attempts/probes

1000 times ⟶ 1006 probes



1 time ⟶ 3 attempts

1000 times x 3 ⟶ 3000 attempts

Note:

In the world of IT, when a record is retrieved, it has been observed that the same record is going to be accessed for SEVERAL TIMES.

So splaying that record towards root requires the following rotations

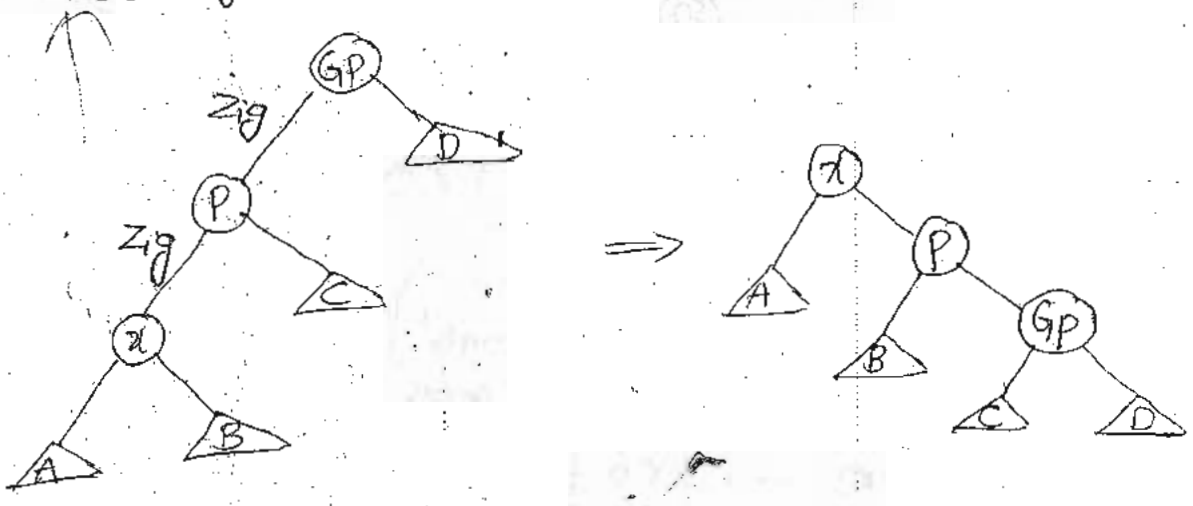
i) Zig

ii) Zig-Zig

iii) Zig-Zag

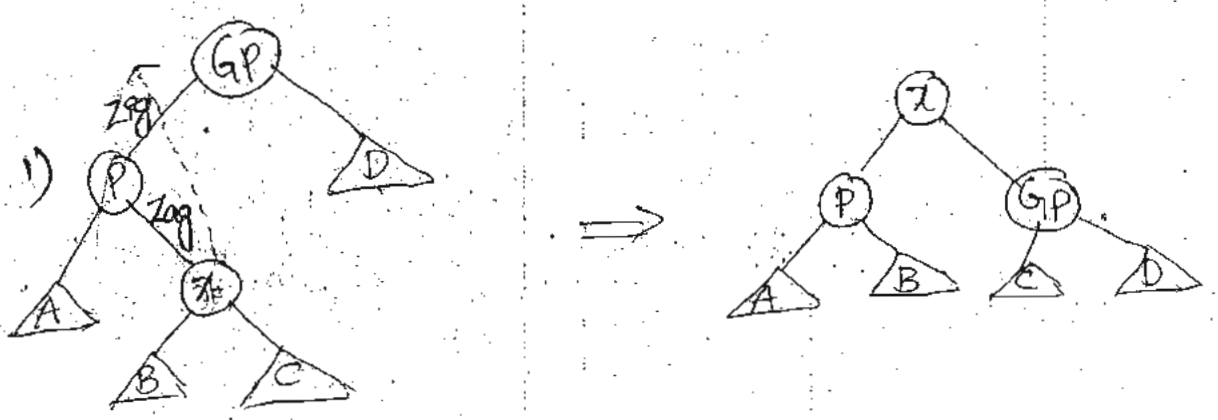
I. Zig:

II. Zig-Zig:



III. Zig-Zag:

1)



2)