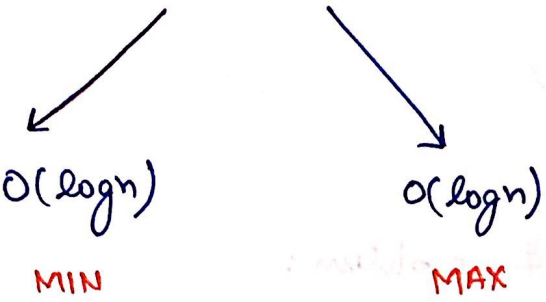
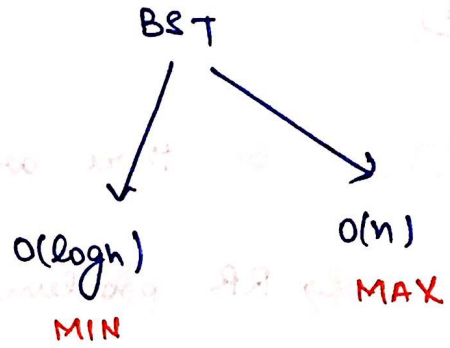
⇒ AVL Tree :-

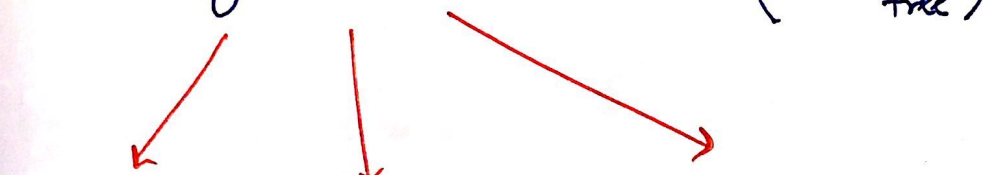
- Height balanced BST :





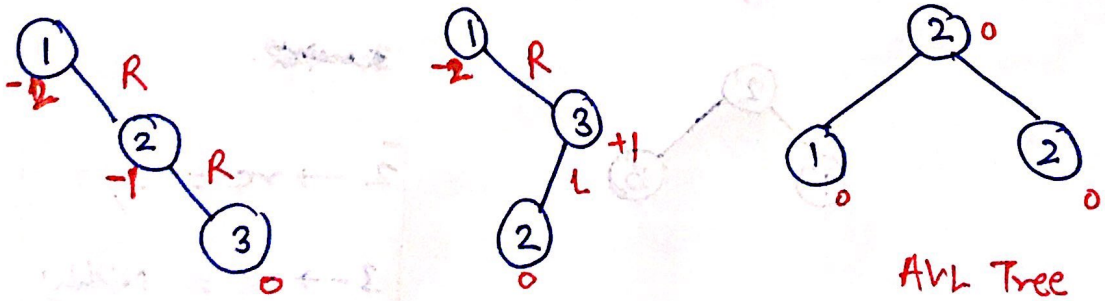
- In the given BST at every node balancing factor is (-1, 0, +1) then it is known as AVL Tree.

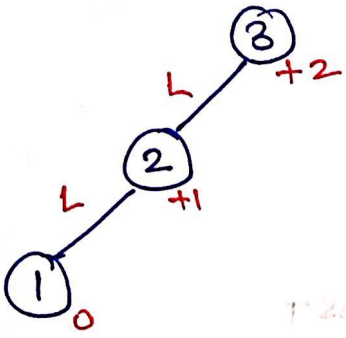
- Balancing Factor = Height (Left sub tree) − Height (Right sub tree)

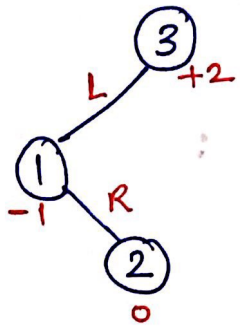


Draw BST and assign Balancing Factor :

n = 3 (1, 2, 3)







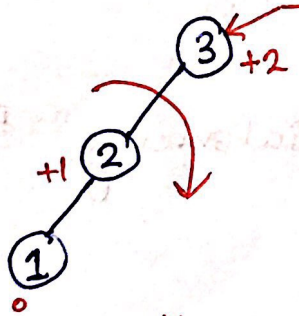
NOTE:- So there are 4 types of problem:

- ↳ RR problem
- ↳ RL problem
- ↳ LL problem
- ↳ LR problem

To remove this problem, rotation are done to balance trees.

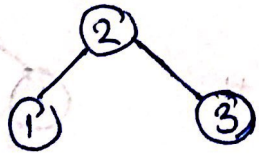
Rotation :-

i) LL- problem :-



So Right rotation is applied on (3) node.

O(1)

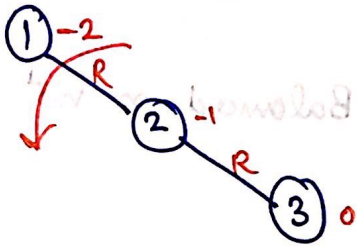


$2 \rightarrow rc = 3;$

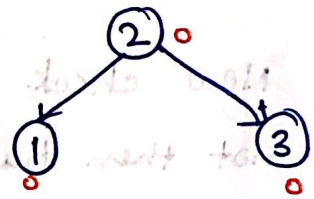
$3 \rightarrow lc = NULL;$

- So, in right rotation parent node fall down the right child of its child.

(ii) RR - problem :-



Left rotation ⟹



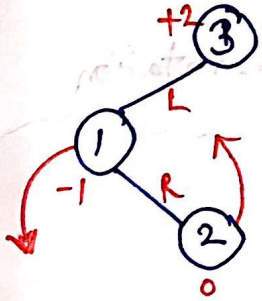
O(1)

$2 \rightarrow lc = 1$

$1 \rightarrow rc = NULL$

- So, in Left rotation parent node fall down to left child of its child.

(iii) LR - problem :-

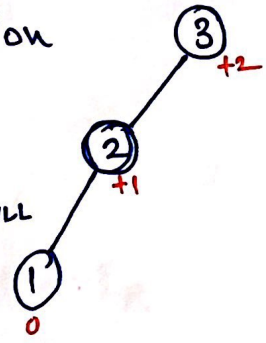


LRotation ⟹

$2 \rightarrow lc = 1$

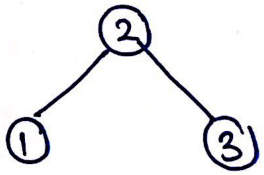
$1 \rightarrow rc = NULL$

O(1)

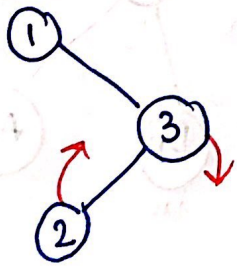


R-Rotation ⟹

O(1)



(iv) RL - problem :-

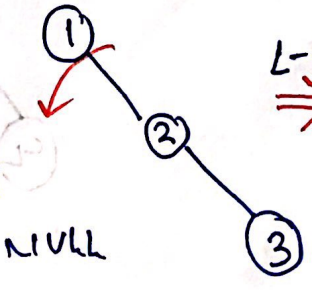


R-rotation ⟹

$2 \rightarrow rc = 3$

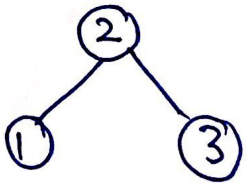
$3 \rightarrow lc = NULL$

O(1)



L-Rotatin ⟹

O(1)



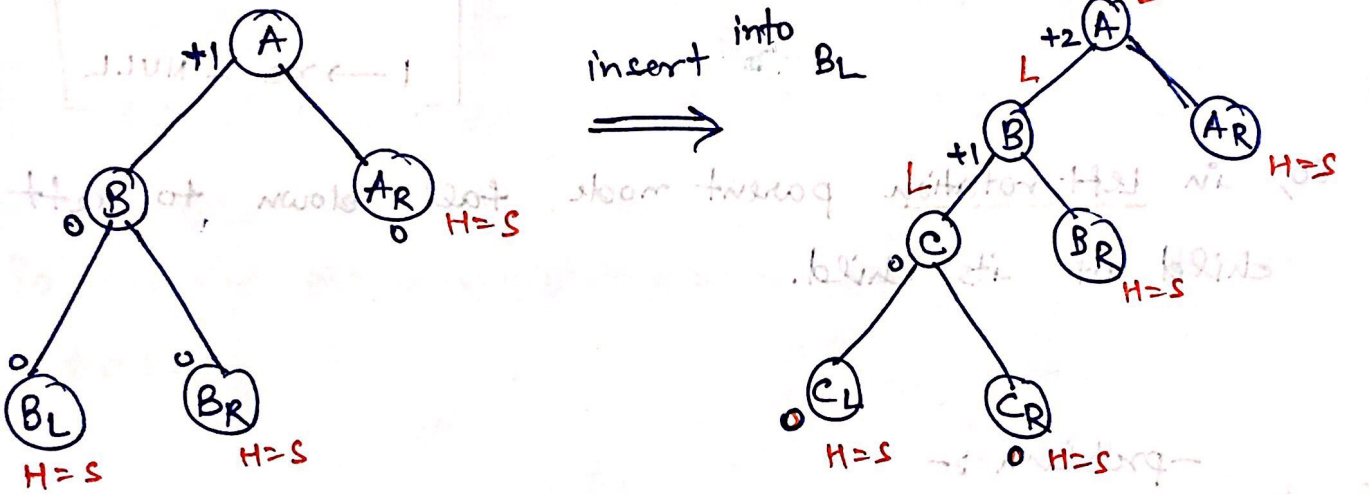
Insertion in AVL Tree :-

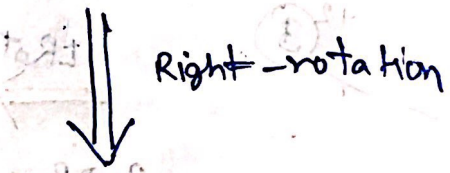
LL-problem:-

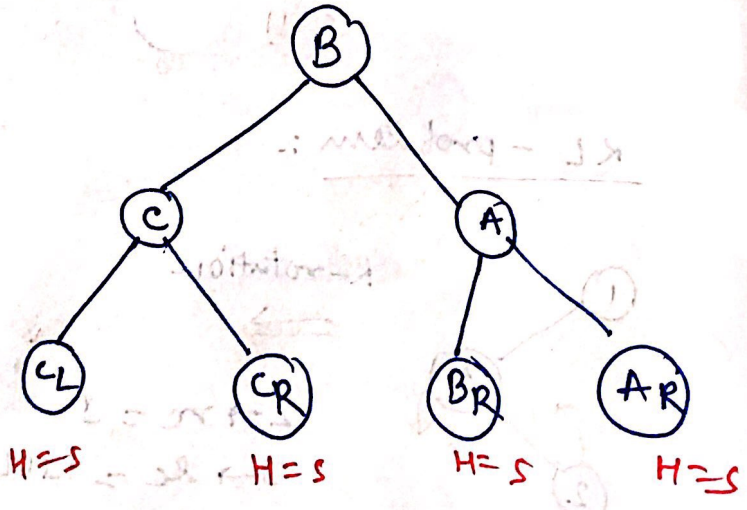
(i) Find place or position where new node has to inserted (Similar to BST) O(logn)

(ii) Now create a node and link to that position. O(1)

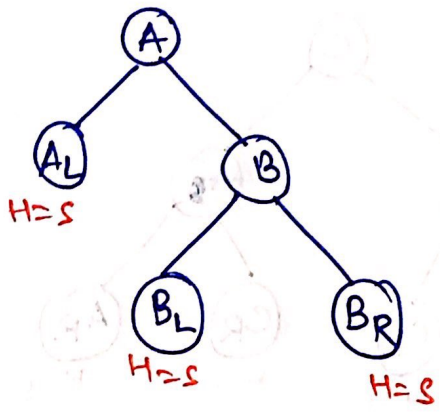
(iii) Now check whether tree is Balanced or not, if not then balanced. O(logn)



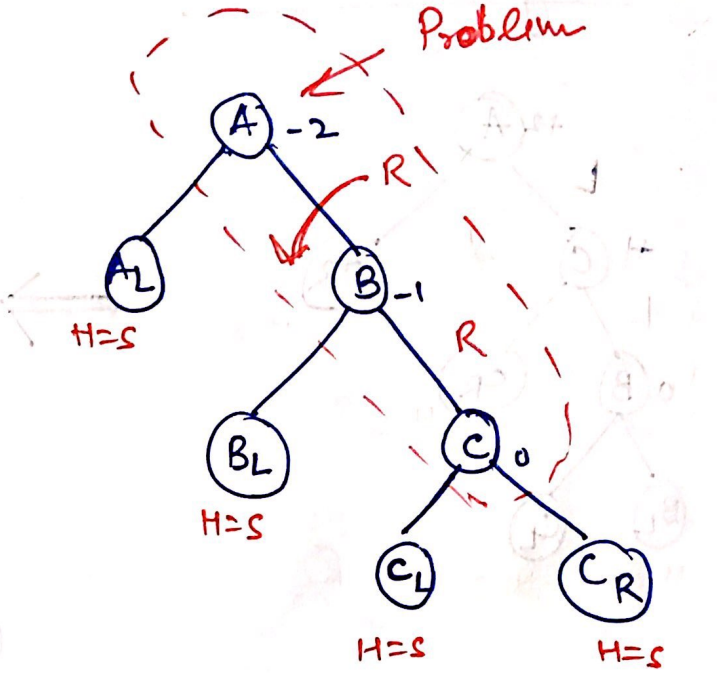




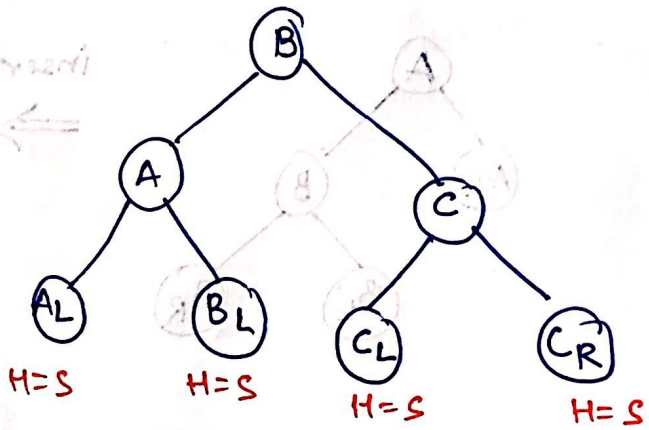
RR - problem :-



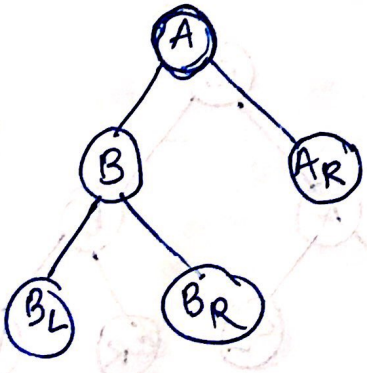
insert into B_R ⟹



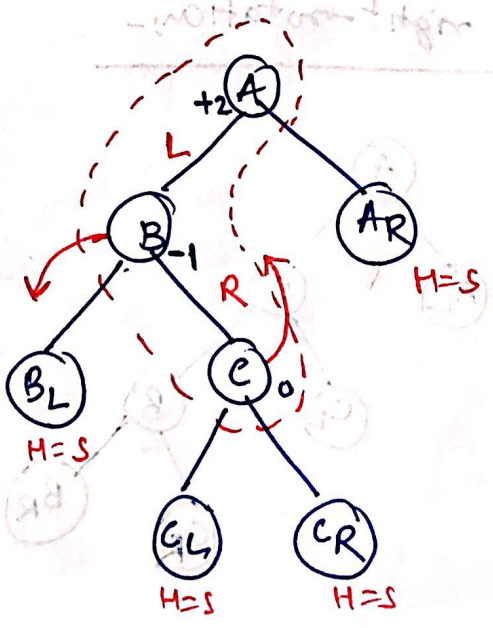
⟹ left - rotation



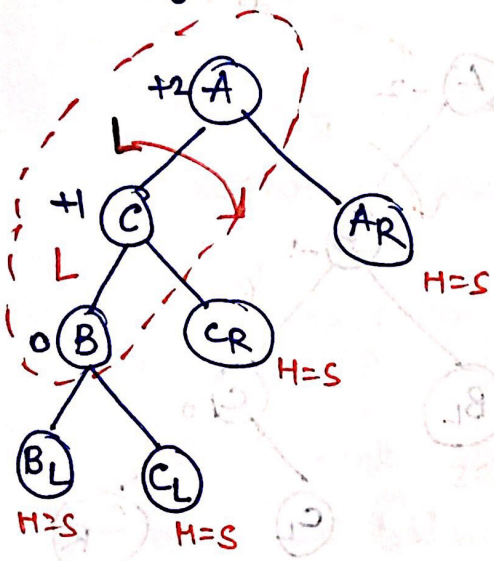
LR - problem :-



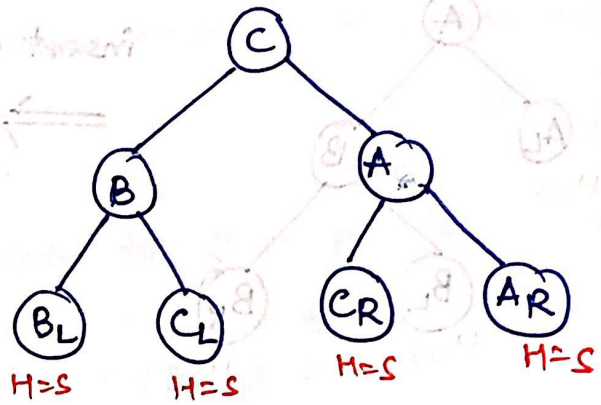
Insert — B_R ⟹



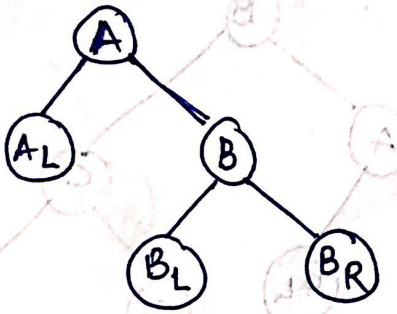
Apply left-rotation:-



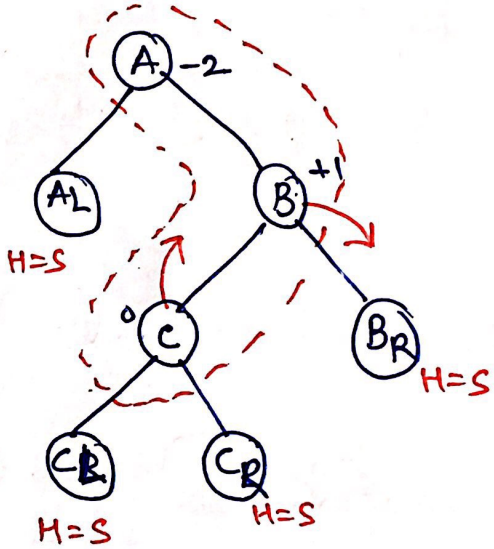
Apply Right-rotation:-



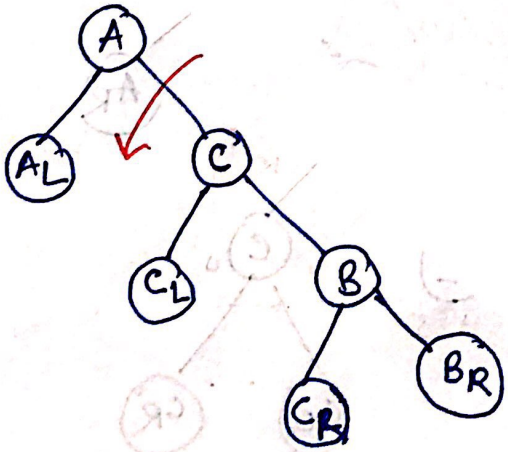
⇒ RL - problem :-



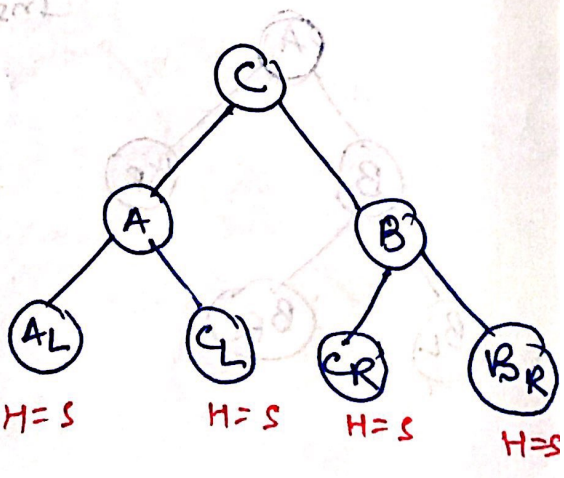
insert - B_L ⇒



Apply right-rotation:-

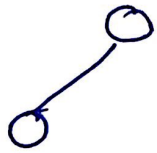


Apply left rotation :-



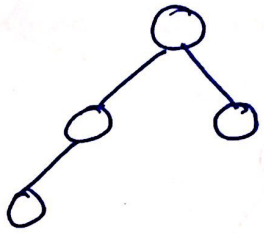
Q: what is the maximum height of AVL tree with 20 nodes.

h=1



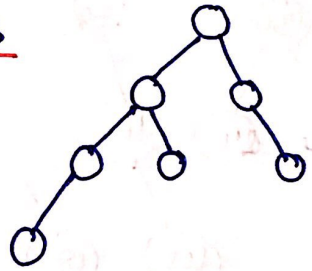
min 2-nodes

h=2



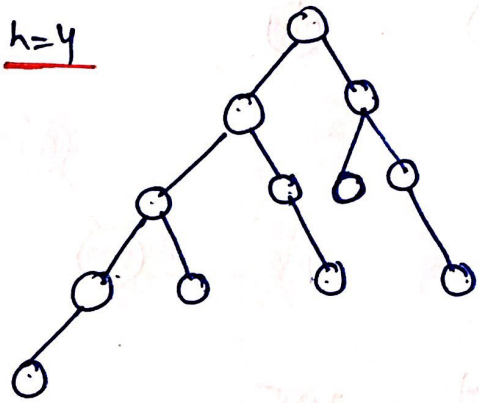
min 4-nodes

h=3



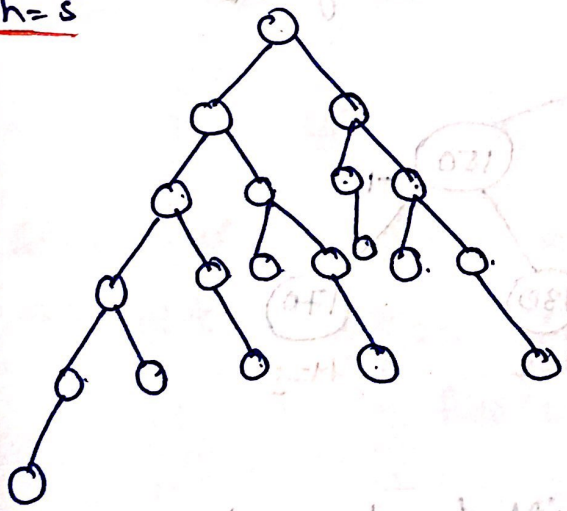
min 7-nodes

h=4



min 12-nodes

h=5



min 20 nodes

So,

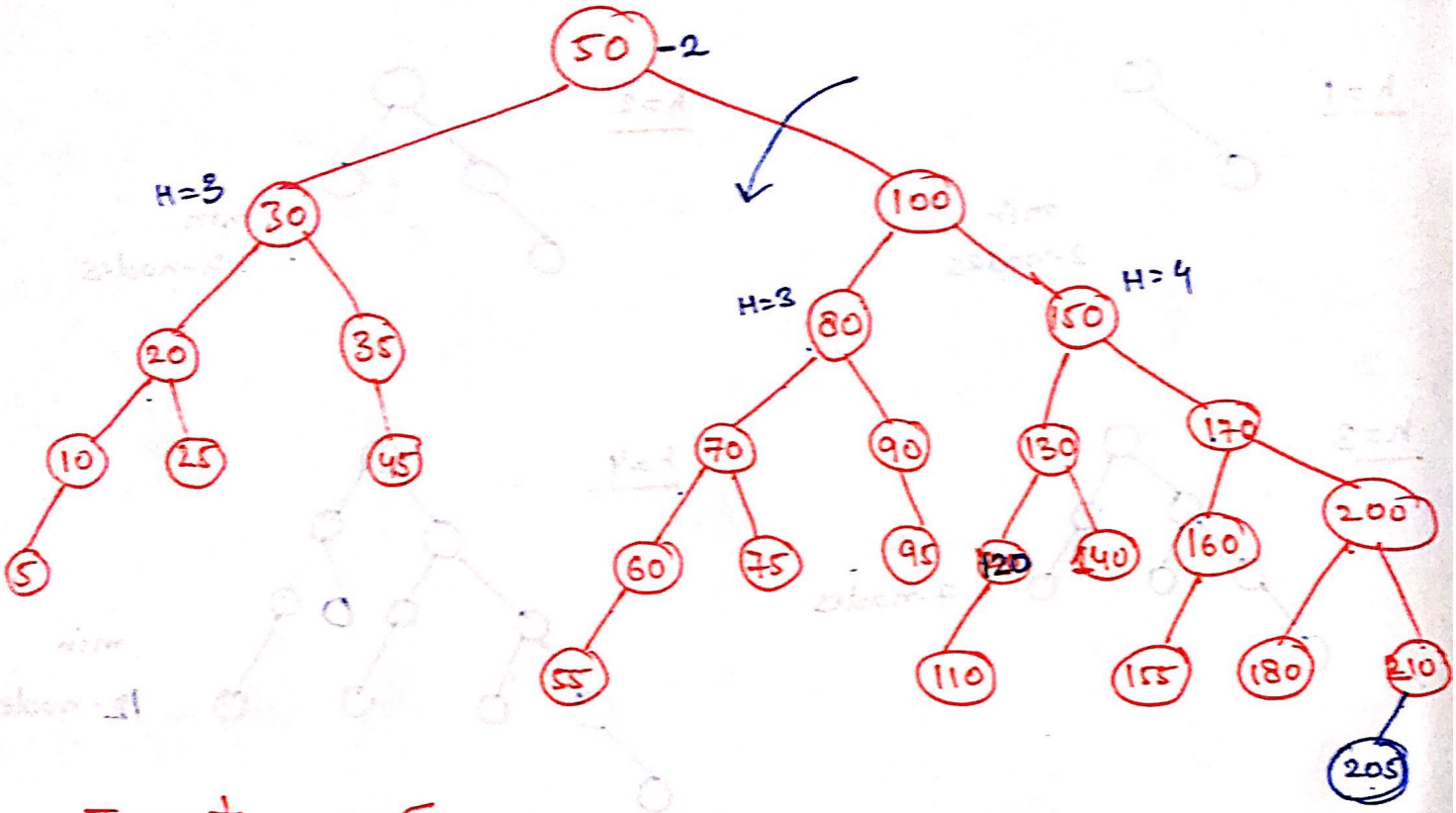
$$\text{min no. of nodes (MNN)}(H) = MNN(H-1) + MNN(H-2) + 1 \quad \text{if } H>2$$

$$= 2 \quad \text{if } H=1$$

$$= 4 \quad \text{if } H=2$$

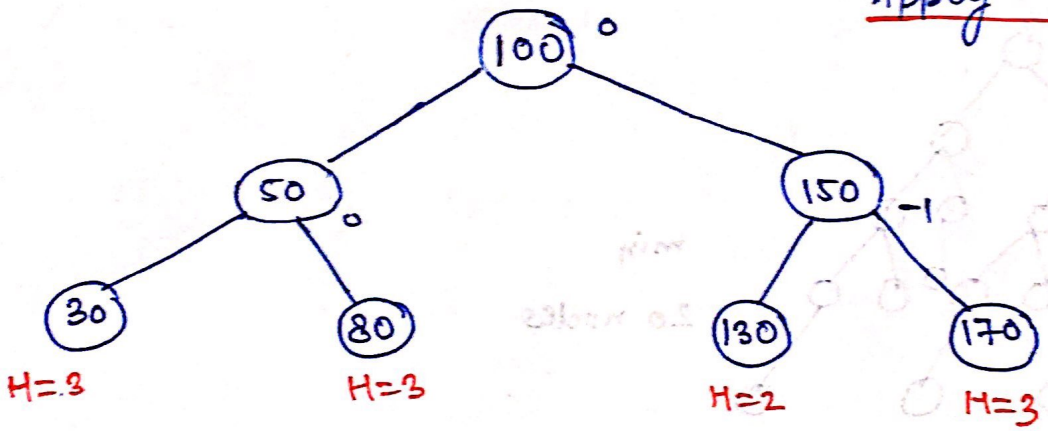
(height)

Q. Consider the following AVL tree :



Insert - 205.

Apply LLH rotation:-

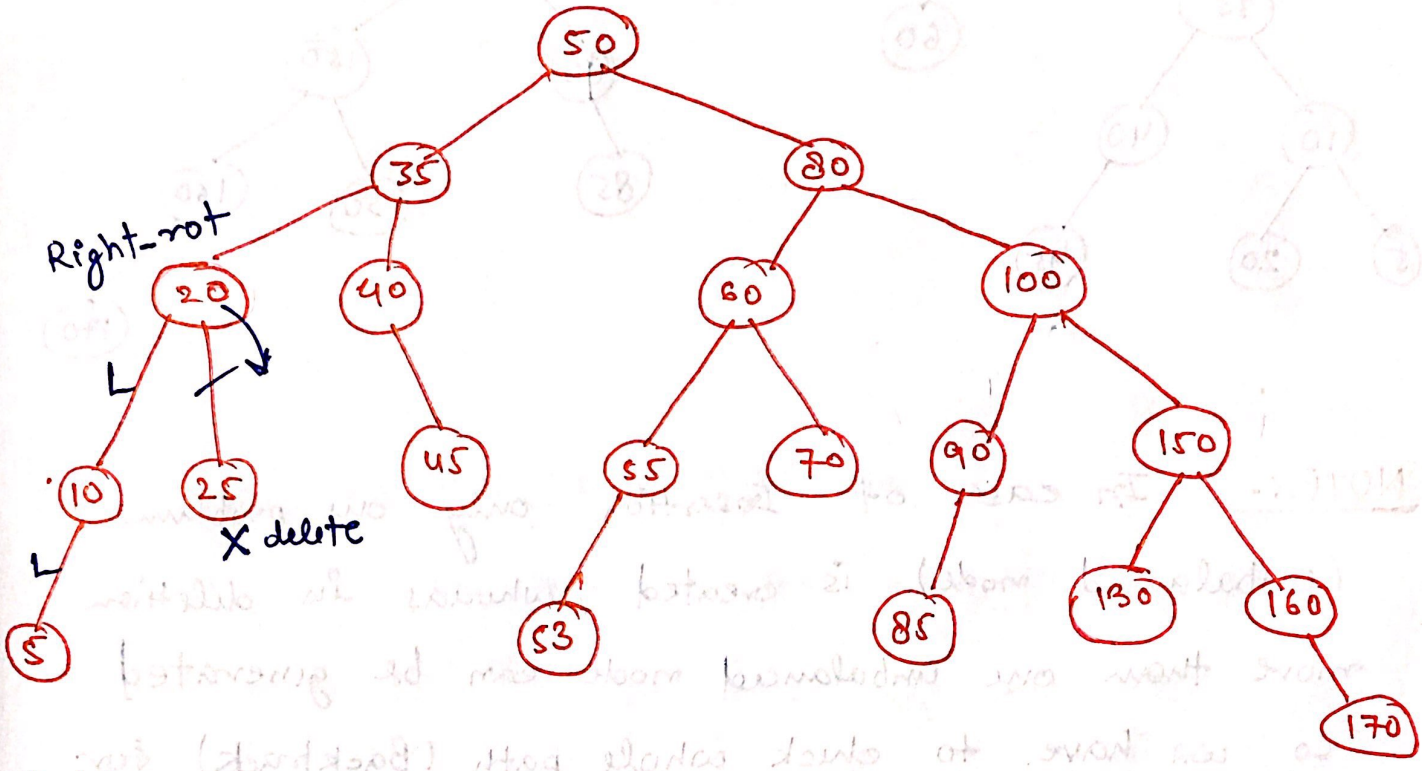


NOTE :- - In AVL Tree insertion will be done at NULL place

- Inserting a element into n element AVL tree will take O(logn) Every case
- To create AVL tree for n elements will take O(n logn). everycase

- In AVL tree after insertion is over, while backtracking in that path (Balancing) : maximum 1 problem can happen (maximum 2 rotation).

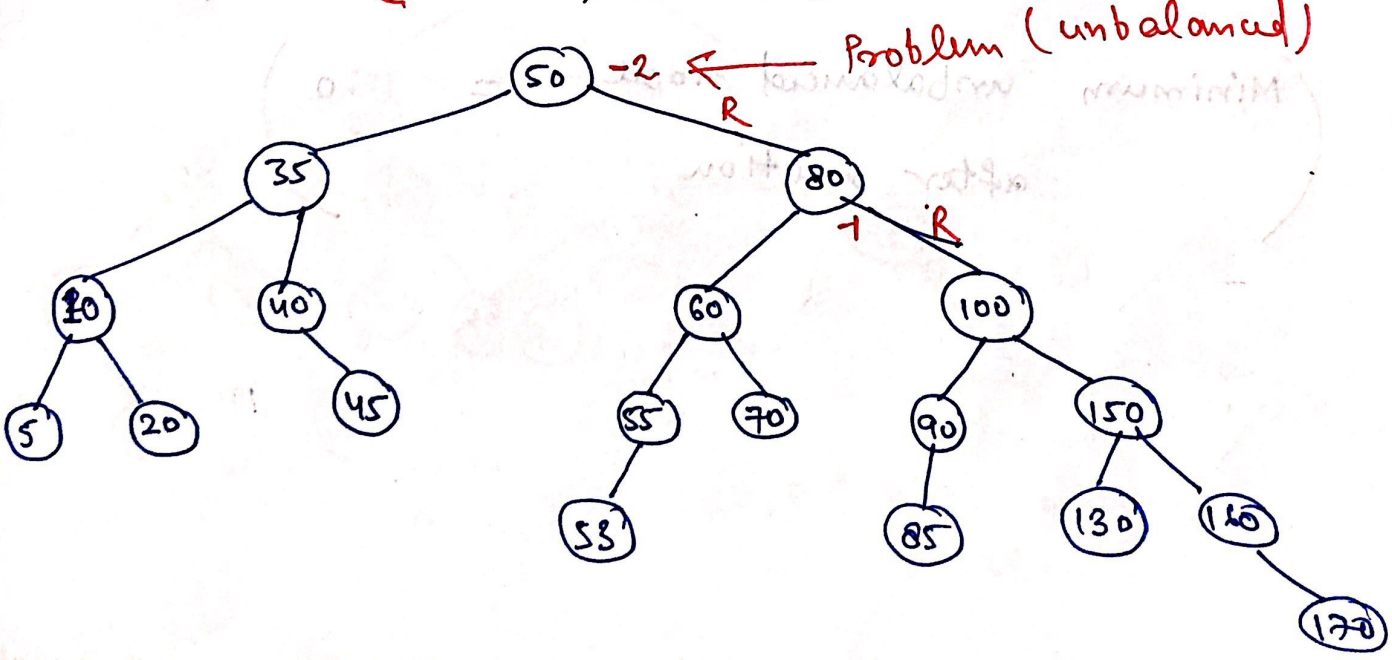
Deletion in AVL tree:-



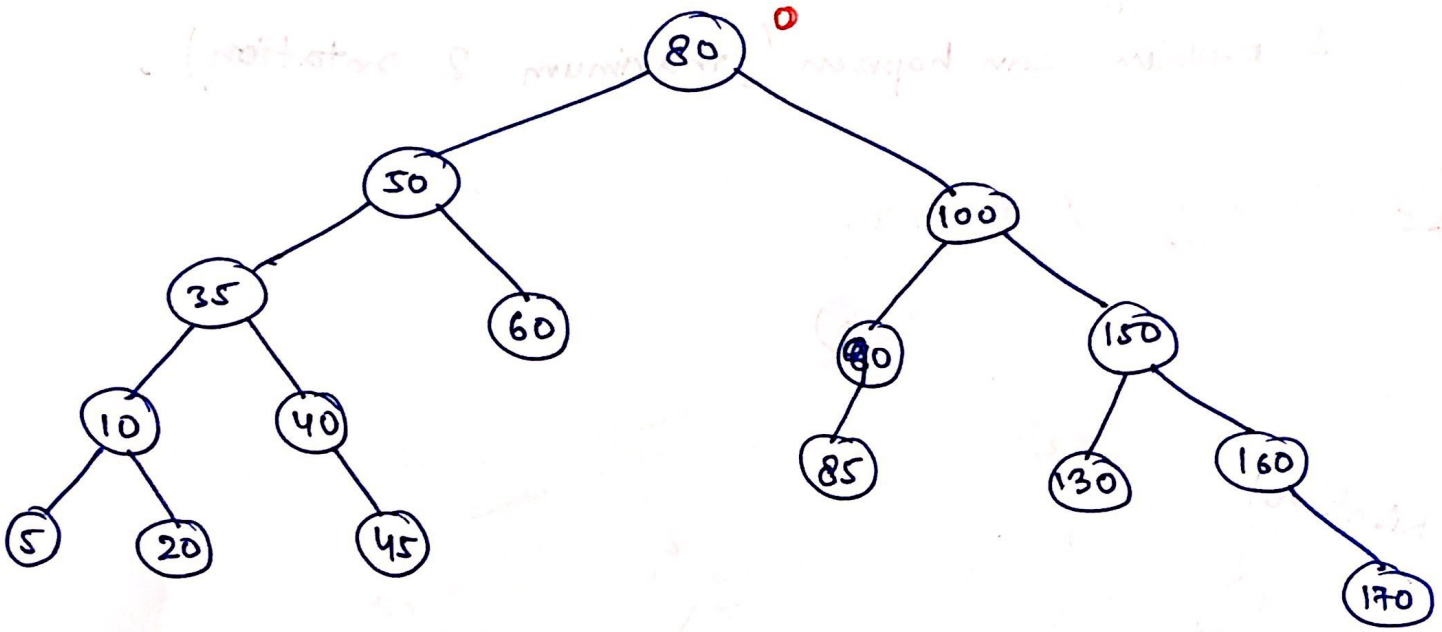
Delete node 25:

Since 25 node has no child, so just simple delete this node:

```
20 → rc = NULL;
free(25);
```



Apply Left-rotation on (50) :-

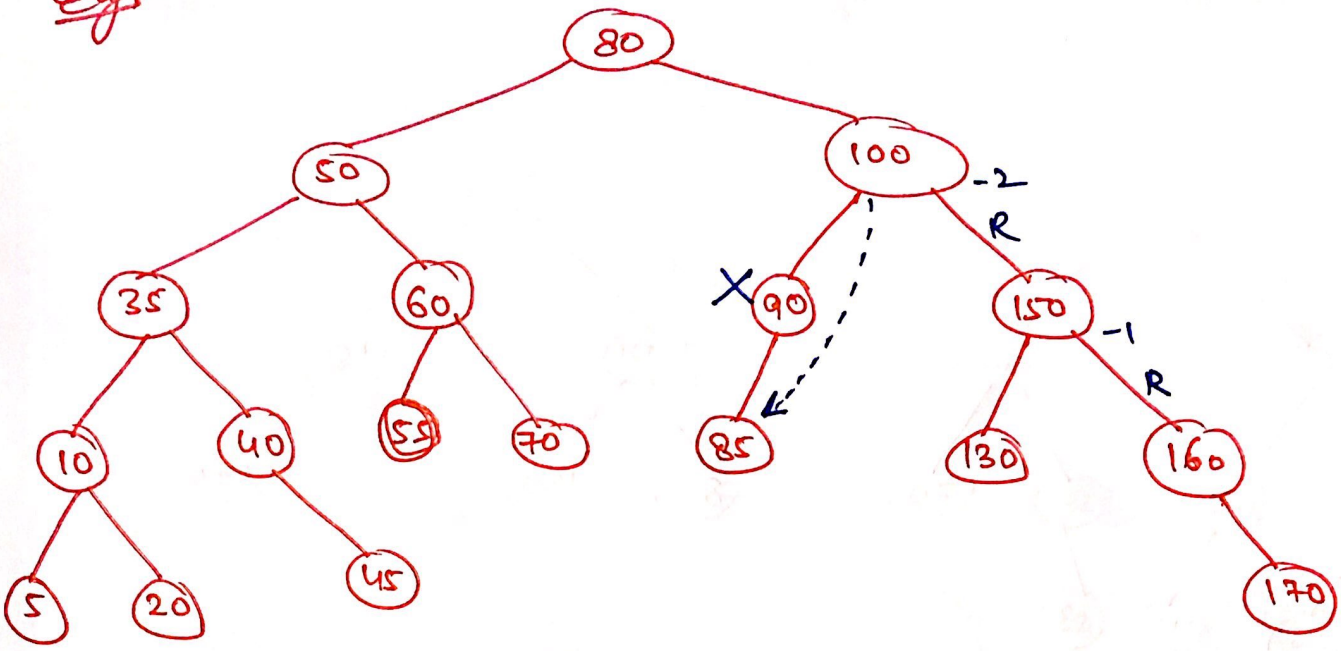


NOTE:- In case of Insertion only one problem (unbalanced node) is created, whereas in deletion more than one unbalanced node can be generated so we have to check whole path (Backtrack) for unbalanced node.

(Maximum unbalanced node possible in Deletion = $O(\log n)$)

(Minimum unbalanced node after deletion = 0)

Eg:-

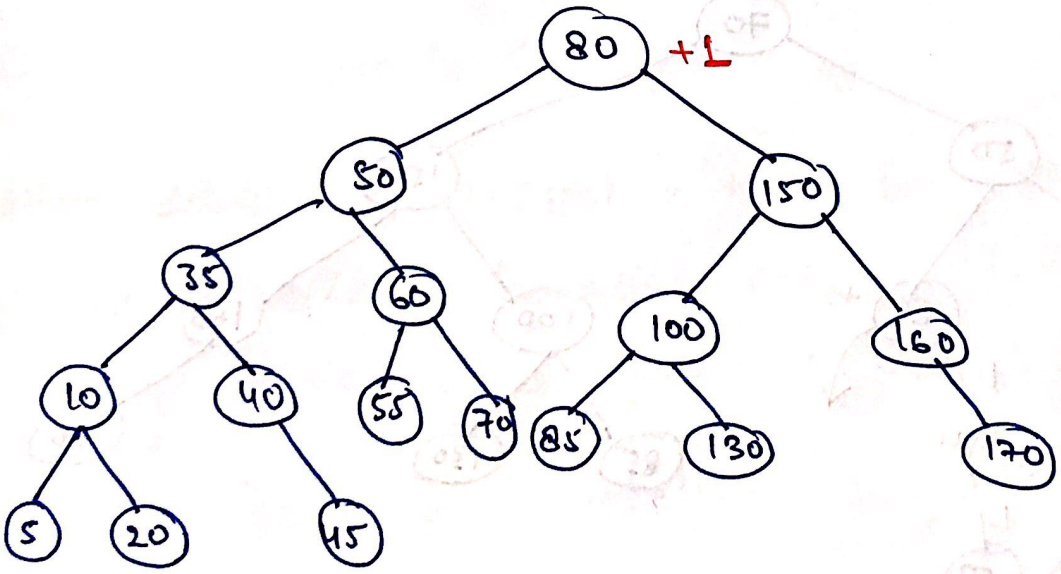


Delete node 90:

Since node 90 has one child so :

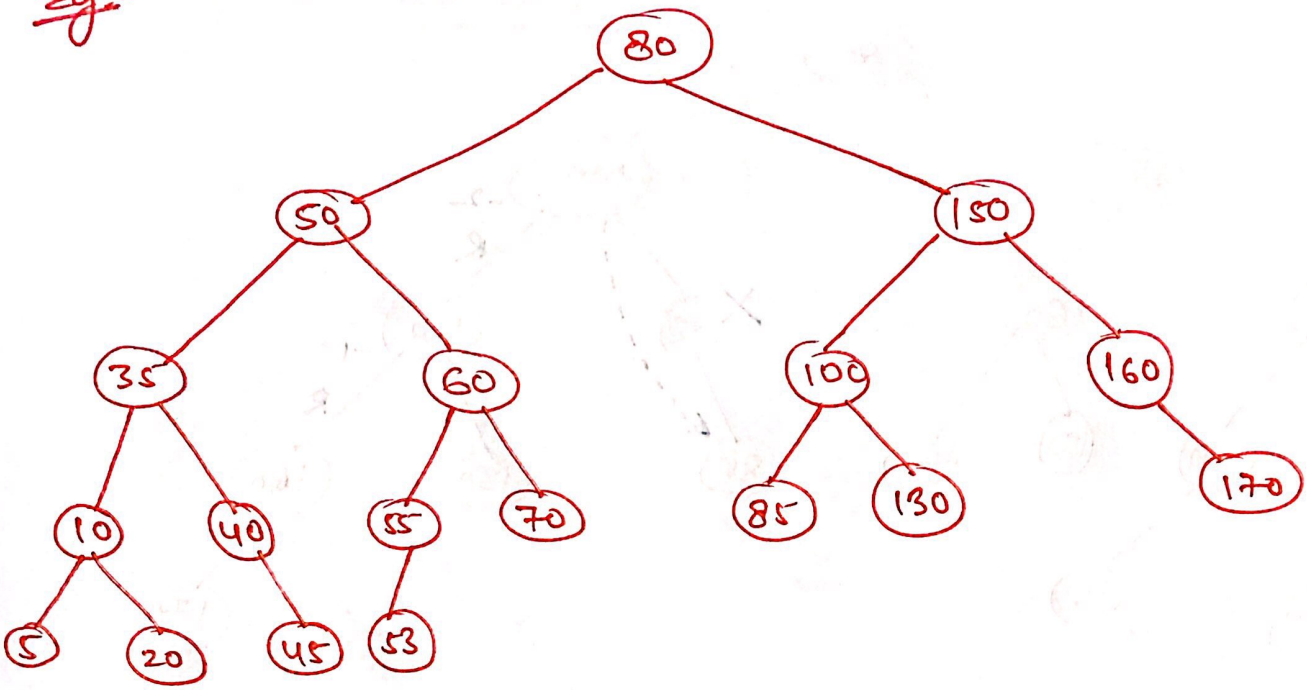
$$\left[\begin{array}{l} 100 \rightarrow lc = 90 \rightarrow lc; \\ free(90); \end{array}\right]$$

Now, BF(100) = −2 and BF(150) = −1, which is RR problem so apply left rotation.



BF(80) = +1 which is balanced.

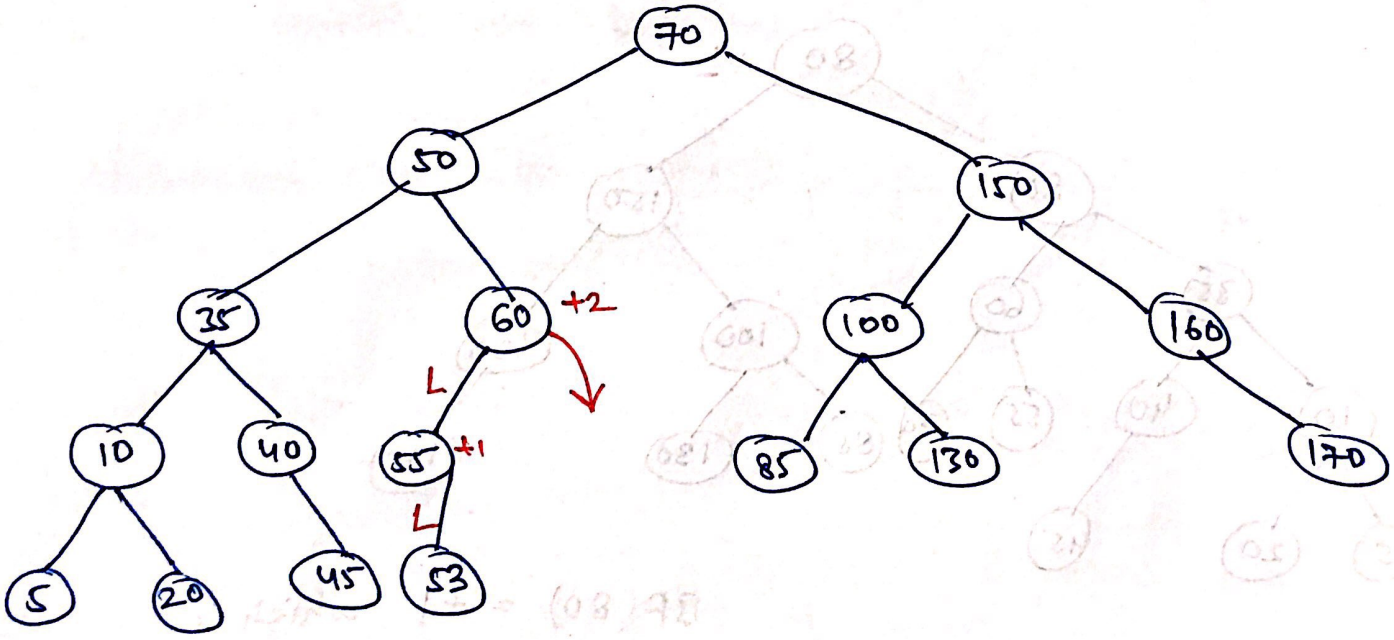
Eg:-



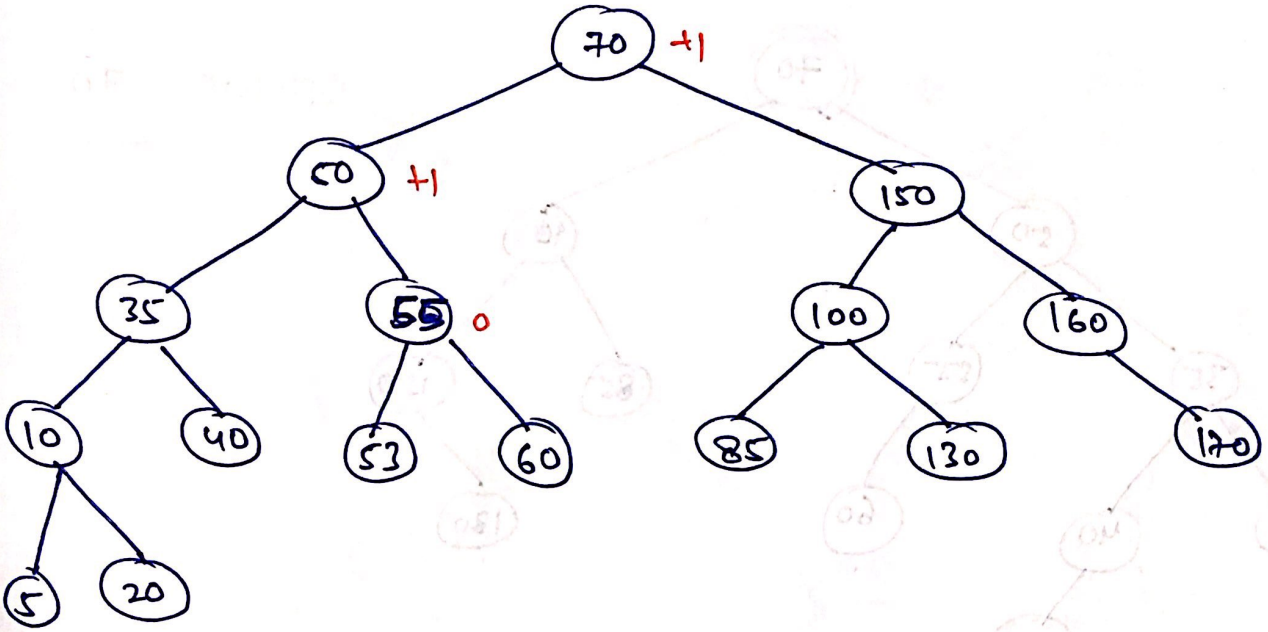
Delete 80 :

Since node 80 has 2 childs so we can substitute it with inorder succesor or predecessor. so we get two solutions:

(i) Predecessor substitution

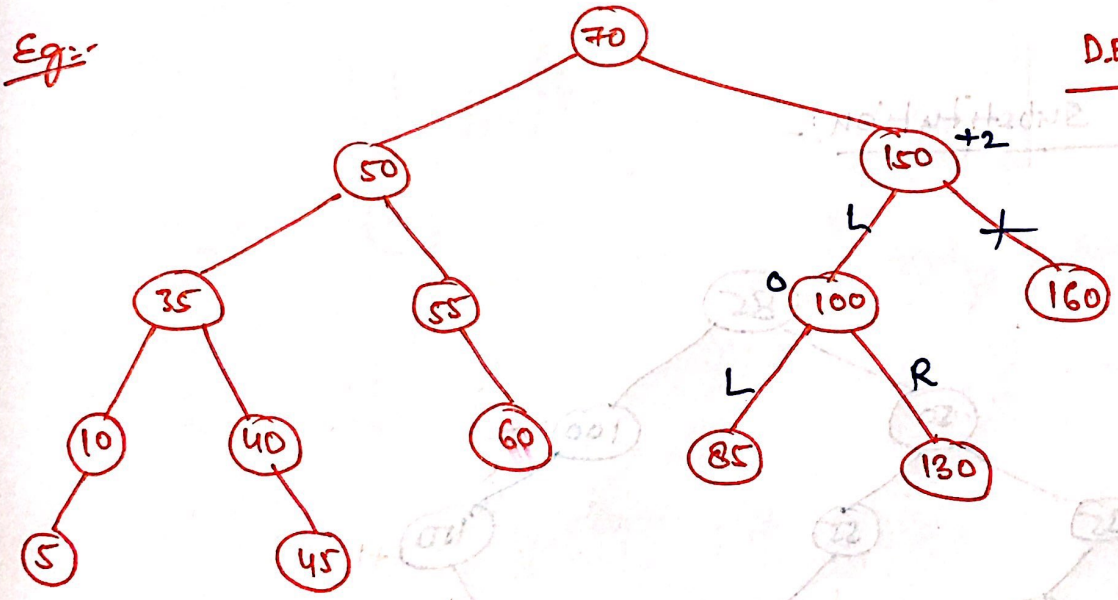


BF(60) is +2 and BF(55) is +1 so LL problem apply right-rotation:



Eg:-

DELETE -160



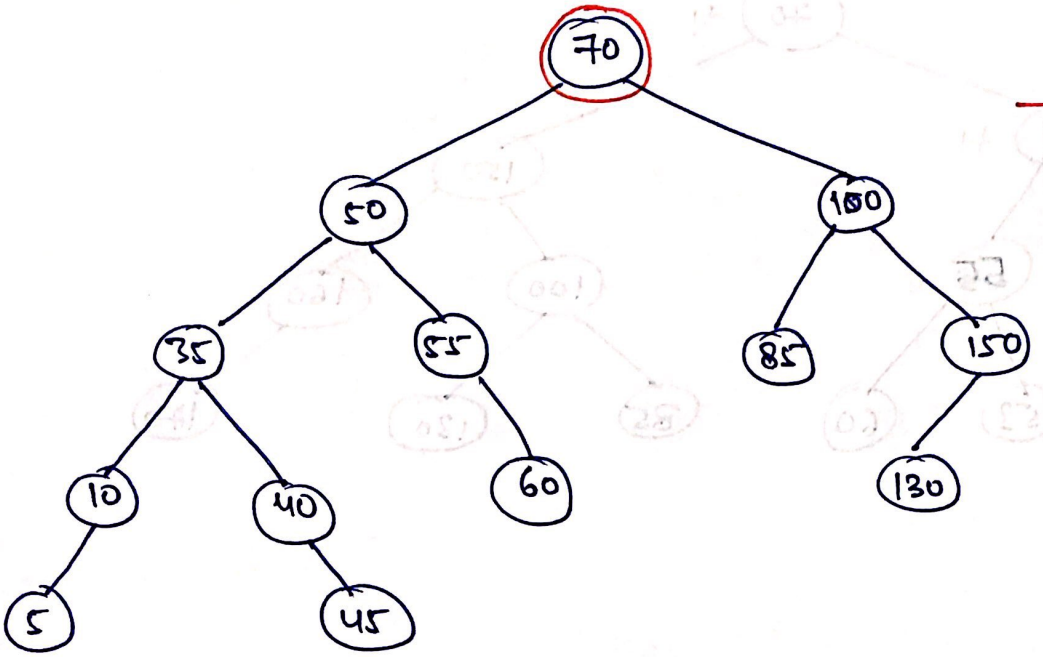
After deletion BF(150) = +2 but BF(100) = 0

so there is no specific problem but we can say there are two types of problem (LL & LR)

LL ⟶ 1 rotation to resolve

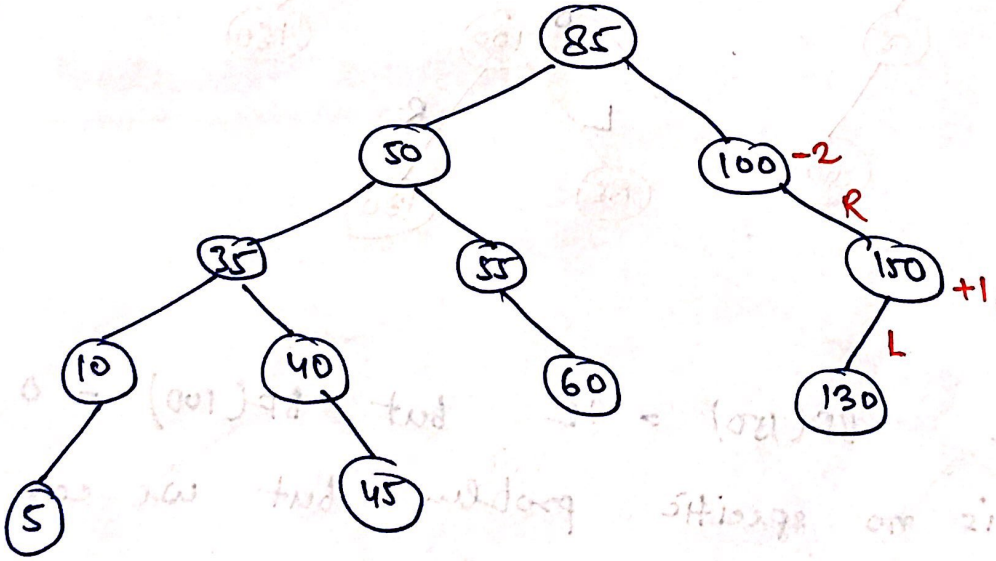
LR ⟶ 2 rotation " "

So we consider it as LL - problem and apply right rotation only.



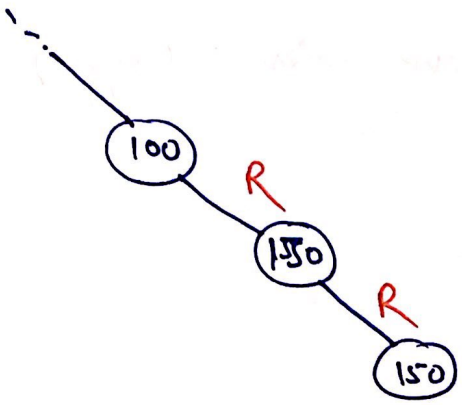
DELETE 70

Successor Substitution:-

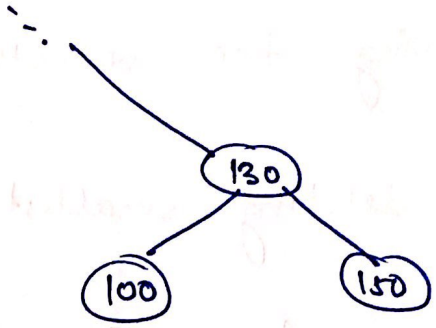


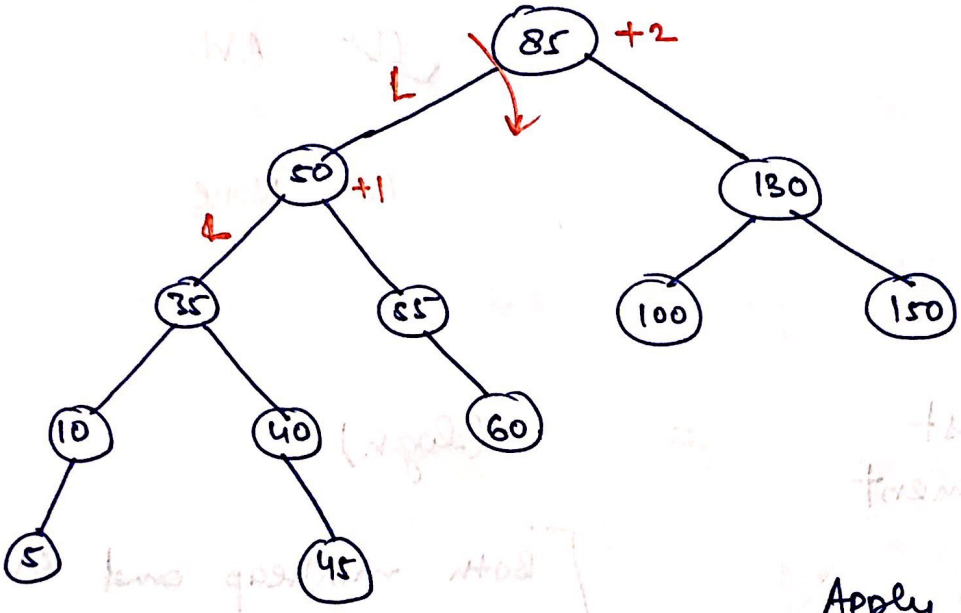
BF(100) = -2 and BF(150) = +1 so RL problem.

So Apply Right rotation:

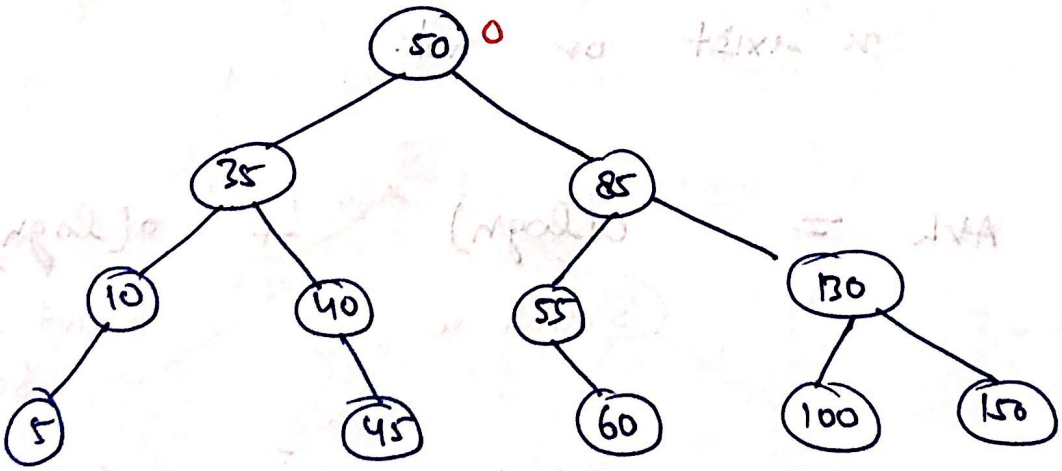


Apply Left rotation





Apply right rotation:



Q: In which of the following data structure following two operations will done with O(logn) time:

(i) deleting smallest element

(ii) insert x if not exist

(a) Minheap (b) AVL

(c) a & b (d) None

Deleting smallest element = O(logn)

[Both minheap and AVL]

But in insert x we have to check whether x exist or not.

so, AVL = O(logn) + O(logn)

search x — Insert & backtrack

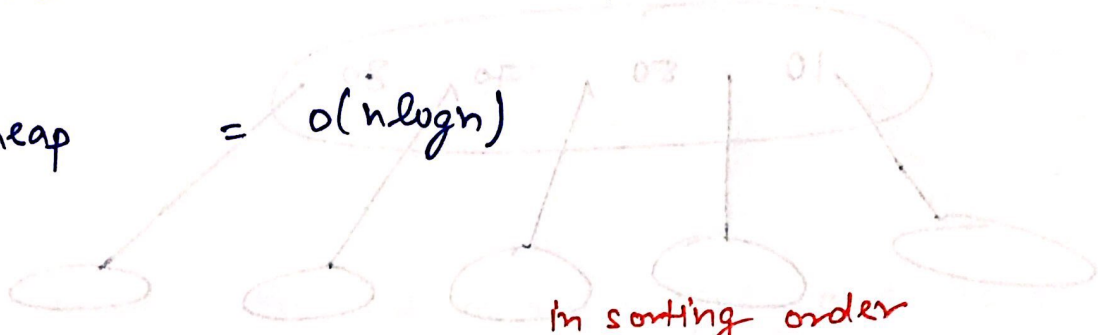
Minheap = O(n) + O(logn)

search x — insert

Q. For sorting minheap & AVL, which is better?

AVL = O(n) (Inorder print)

Minheap = O(nlogn)



in sorting order

Q. For printing m elements b/w L and H nodes of AVL tree time complexity :

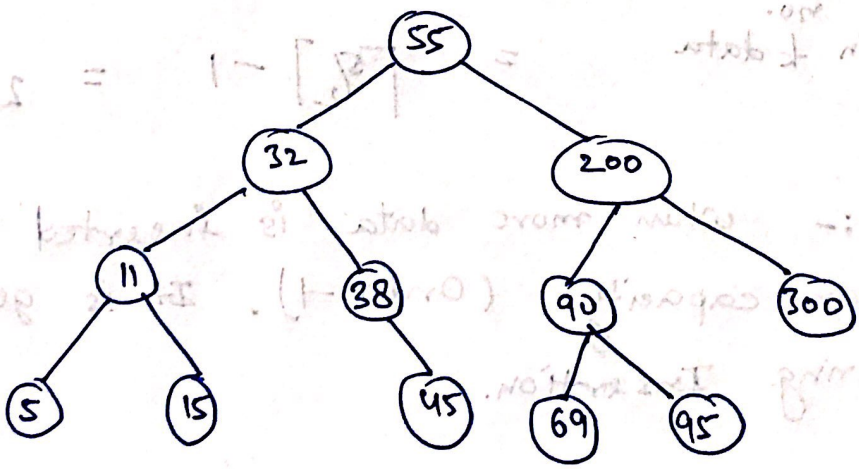
$T(n) = O(\log n) + m$

searching L node in AVL

printing Inorder

Q. Create AVL tree :

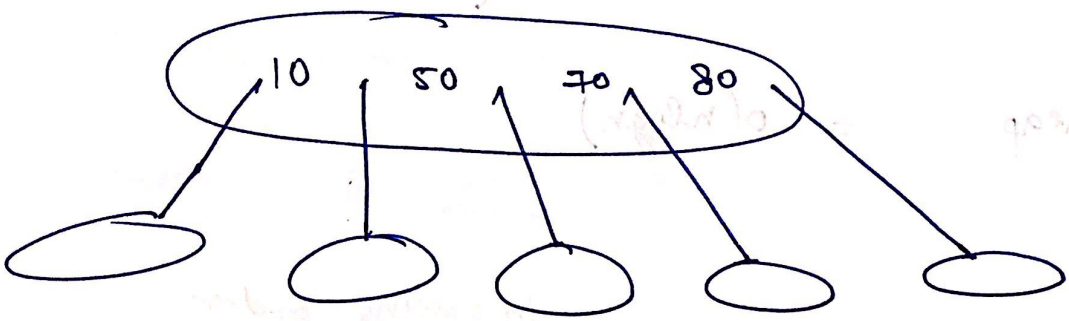
55 95 11 200 5 38 300 69 15 32 45 90



4- L Rotation

2- R Rotation

Balanced - Tree (BTree):-



- Balanced - tree
- Search - tree
- Atmost 'n' childs depends on order

order : maximum no. of child

so, if order = 5

Max - child = 5

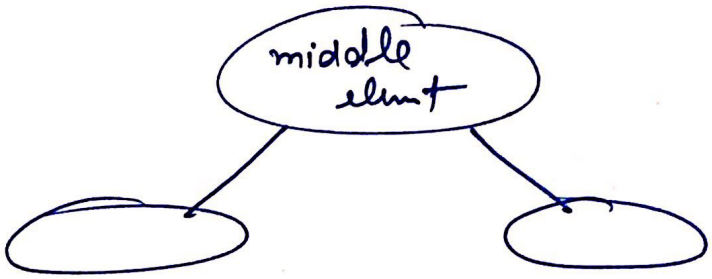
Min - child = $\lceil 5/2 \rceil = 3$

Max no. of data = (5) - 1 = 4

Min no. of data = $\lceil 5/2 \rceil - 1 = 2$

- Overflow :- when more data is inserted (more than capacity (Order-1). It is generated during Insertion.

(Overflow ⟶ node splitting)



- Underflow :- when data is deleted and no. of elements left inside node is less than minimum capacity.

(Underflow ⟶ node combine)