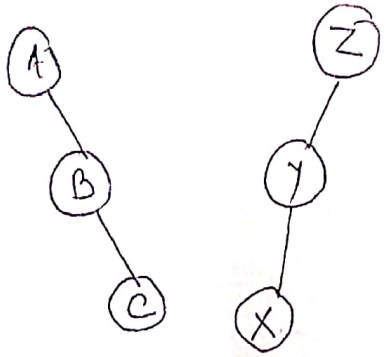
AVL Search Tree:

Consider that the elements A, B, and C to be inserted into a BST. Then BST results in a right skewed and again the insertion Z, Y X turns out to be a left skewed. The disadvantage of a skewed BST is that worst case complexity of a search is $O(n)$. The disadvantage can be eliminated if the BST is balanced properly. If BST is balanced then time complexity becomes $O(\log_2 n)$ in the worst case.



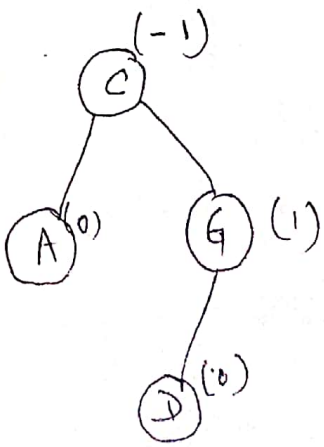
skewed BST.

One of the popular balanced tree was introduced in 1962 by Adelson-Velskii and Landis and was known as AVL trees.

Definition: An empty binary tree is an AVL tree. A non empty binary tree T is an AVL tree iff given T^l and T^r to be the left and right subtree of T and $h(T^l)$ and $h(T^r)$ to be the heights of subtree T^l and T^r respectively, T^l and T^r are AVL tree and $|h(T^l) - h(T^r)| \leq 1$.

$h(T^l) - h(T^r)$ is known as Balance Factor (BF) and for AVL tree the balance factor of a node can be $-1, 0$ or $+1$.

An AVL Search tree is a BST which is AVL tree. AVL search tree like BST are represented using linked list. However, every node registers its balance factor as shown.



Insertion in AVL tree:

Inserting an element into an AVL tree in its first phase is similar to that of BST. However, if after insertion of the element, the balance factor of any node in the tree is affected, i.e. it renders the BST unbalanced, we resort to technique called rotations to restore the balance of BST.

In order to perform rotations it is necessary to identify a specific node A where BF(A) is neither 0, +1 or -1, and which is the nearest ancestor to the inserted node on the path from the inserted node to the root. This implies that all nodes on the path from the inserted node to A will have their balance factor to be either 0 or 1 or -1. The rebalancing are classified as below, based on the position of the inserted node with reference to A.

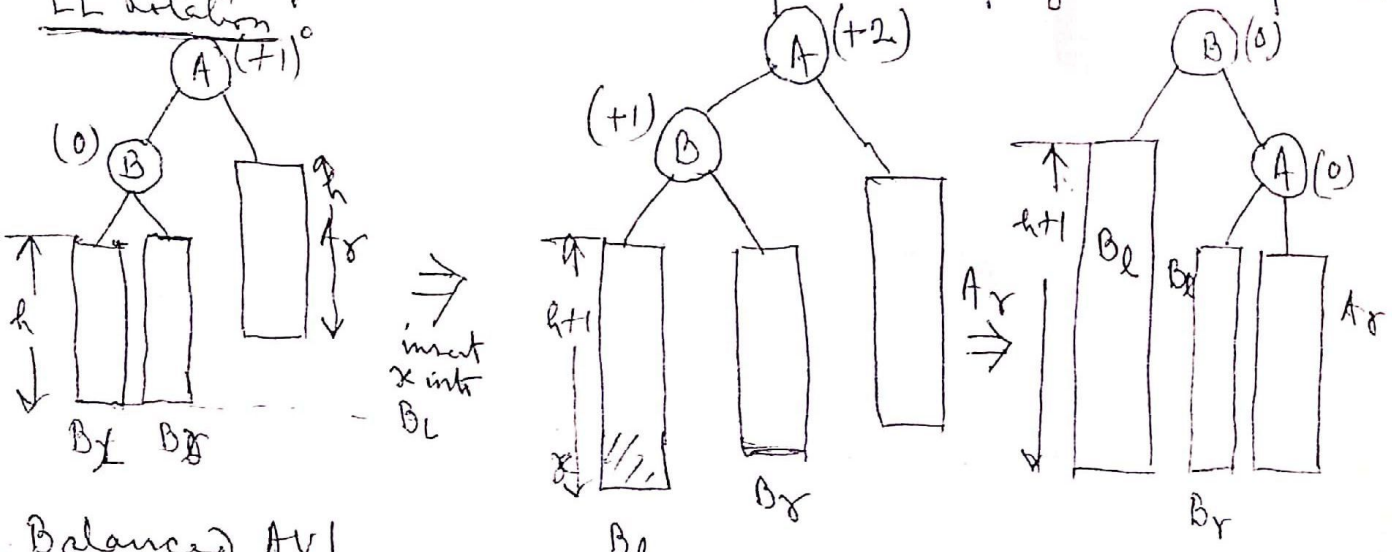
LL rotation : Inserted node is in left subtree of left subtree of node A

RR rotation : Inserted node is in right subtree of right subtree of node A

LR rotation : Inserted node is in right subtree of left subtree of node A

RL rotation : Inserted node is in left subtree of right subtree of node A

LL Rotation :



Balanced AVL Search tree

B_L Unbalanced AVL Search tree after insertion

Balanced AVL Search tree after rotation

B_L → left subtree of B

B_R → right subtree of B

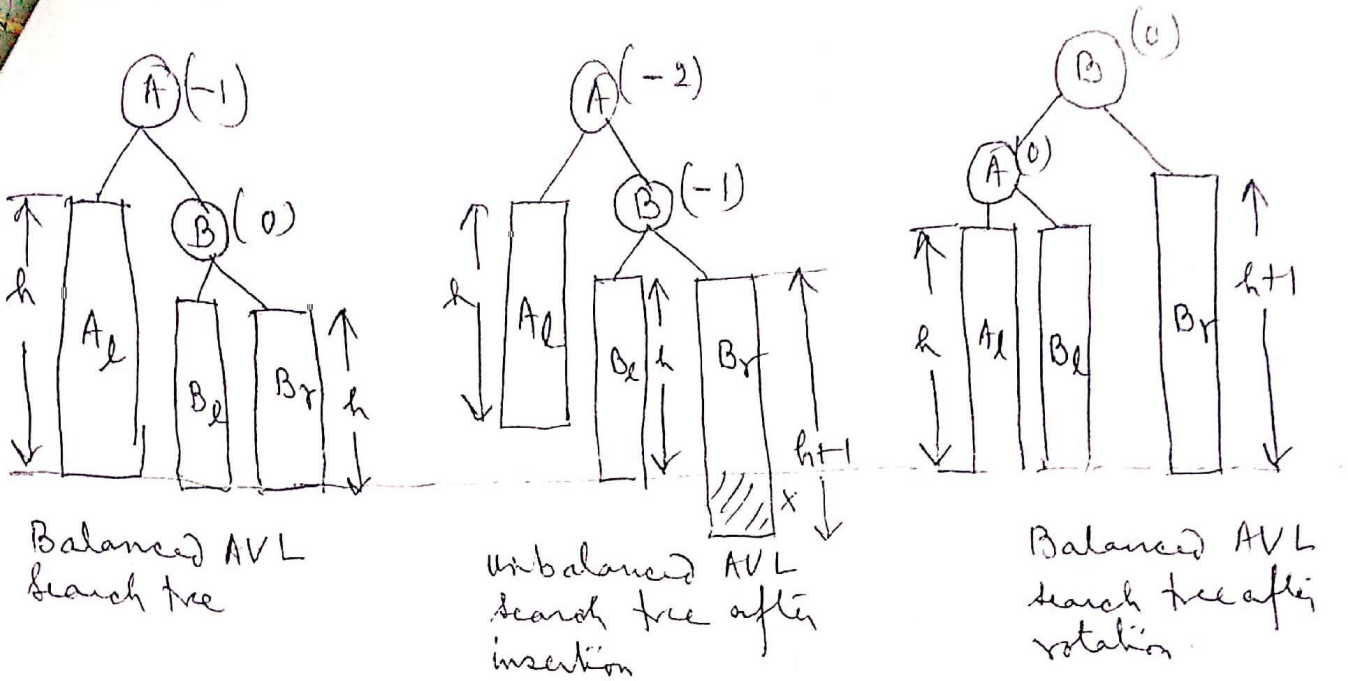
A_R → right subtree of A

h → ~~Height~~ height

The new element X is inserted in the left subtree of left subtree of A, the closest node whose BF(A) becomes +2 after insertion.

In order to rebalance the ~~subtree~~ Search tree, it is related to allow B to be the root with B_L and A to be its right subtree and right child, and B_R and A_R are to be the left and right subtrees of A.

rotation:



Balanced AVL search tree

Unbalanced AVL search tree after insertion

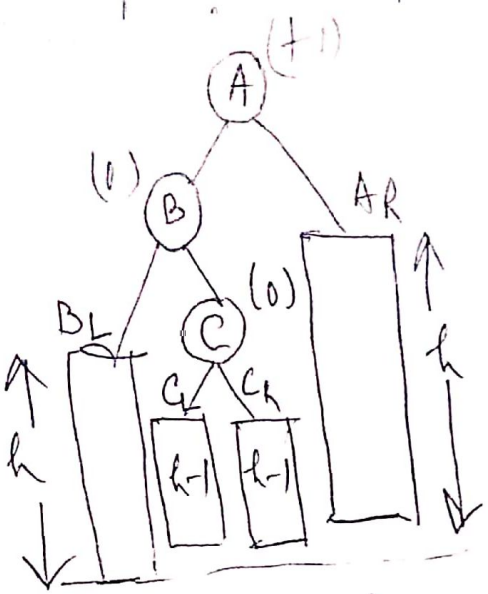
Balanced AVL search tree after rotation

LR rotation:

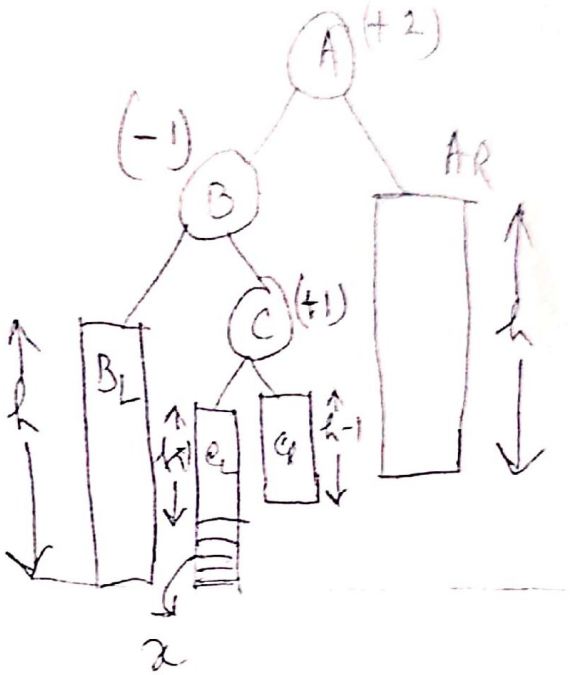
The balancing of LR and RL methodologies are similar in nature but are mirror images of one another. In this case the BF values of nodes A and B after balancing are dependent on the BF value of node C after insertion.

If BF(C) = 0 after insertion then BF(A) = BF(B) = 0 after rotation.
If BF(C) = -1 after insertion then BF(A) = 0, BF(B) = 1 after rotation
If BF(C) = 1 after insertion then BF(A) = -1, BF(B) = 0, after rotation

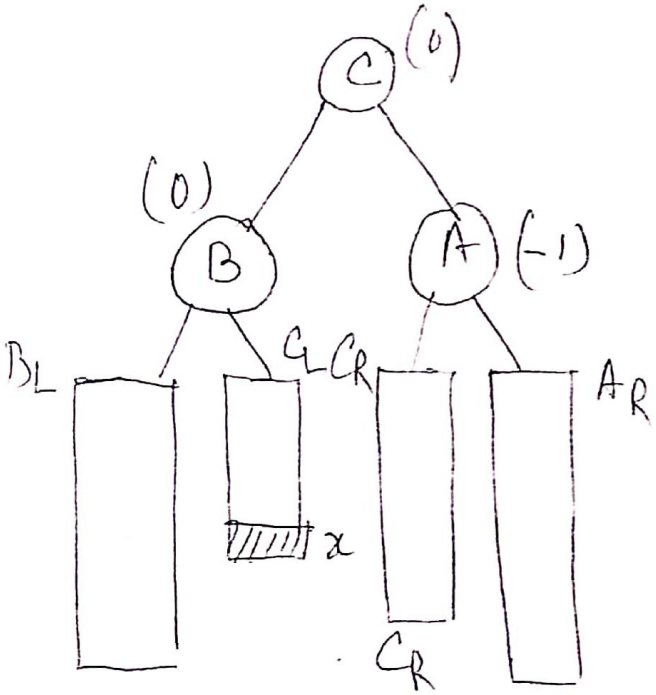
Amongst the rotation LL and RR rotations are called as single rotation and LR and RL are known as double rotations, since LR is accomplished by RR followed by LL rotation and RL can be accomplished by LL followed by RR rotation. The time complexity of an insertion operation in an AVL tree is given by $O(\log n) = O(\log_2 n)$.



Balanced AVL search tree



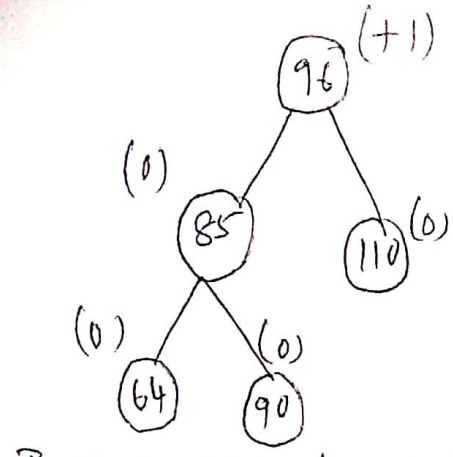
Unbalanced AVL search tree after insertion



Balanced AVL search tree after LR rotation

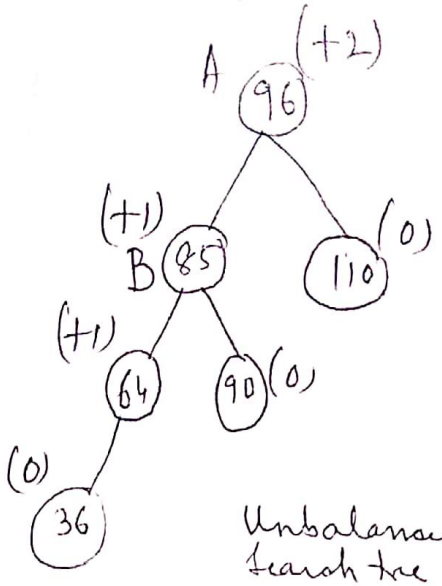
C_L : Left subtree of C
C_R : Right subtree of C.

Example of LL rotation :- (96, 85, 110, 64, 90, 36)

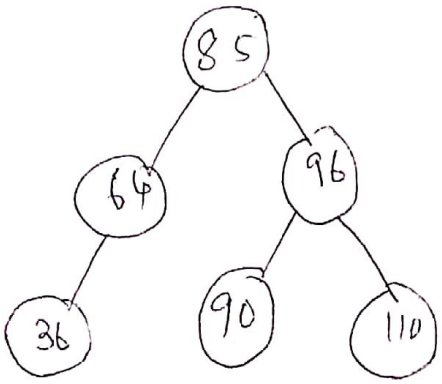


Initial AVL search tree

Insert 36.



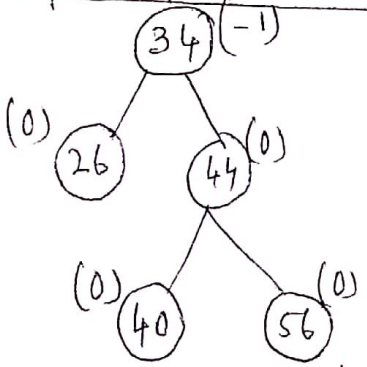
Unbalanced AVL search tree



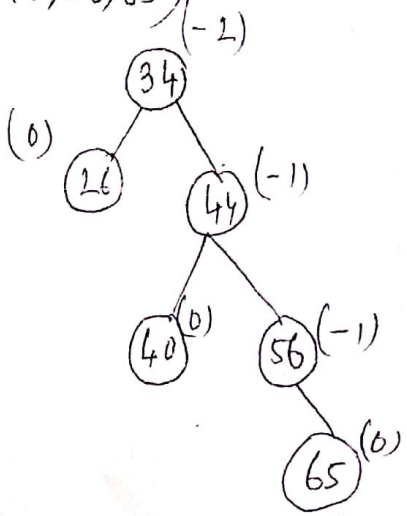
Balanced AVL search tree after rotation.

*Note: (90) cannot be in the right subtree of 64 as 90 > 85, Every node in left subtree of 85 should be less than 85.

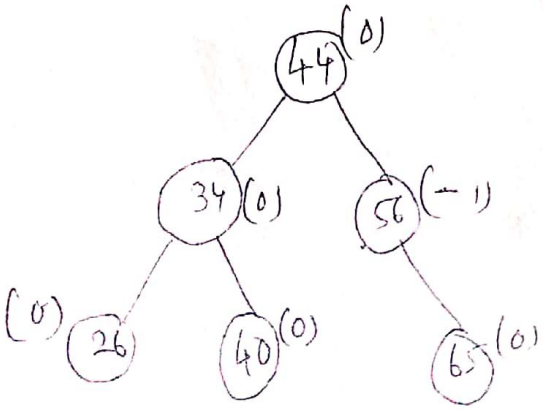
Example of RR rotation : (34, 26, 44, 40, 56, 65)



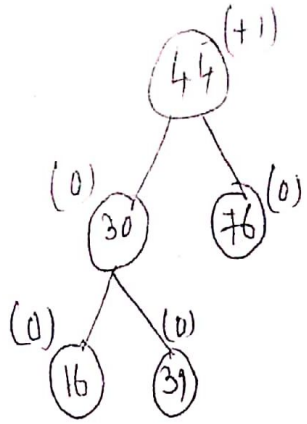
Initial AVL search tree (a)



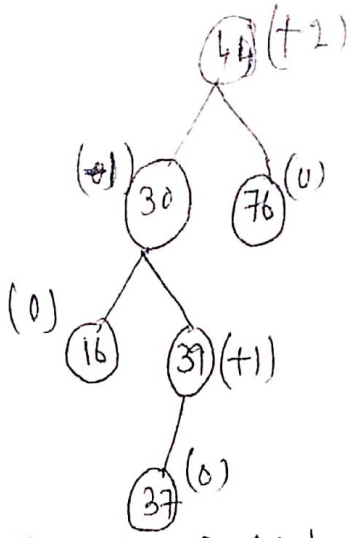
Unbalanced AVL search tree (b)



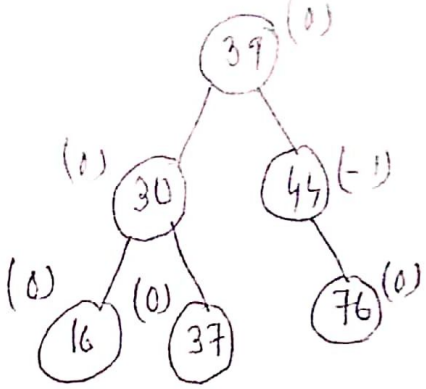
Balanced AVL search tree after RR [illegible]



Initial AVL search tree

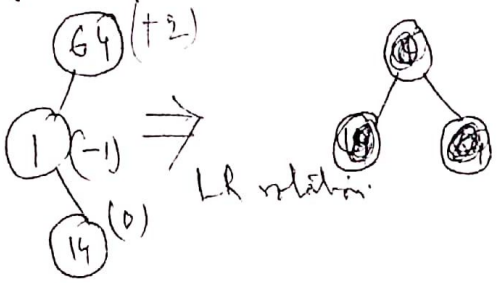


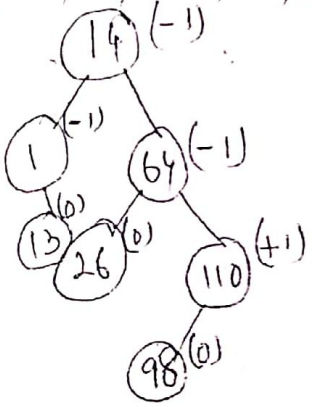
Unbalanced AVL search tree



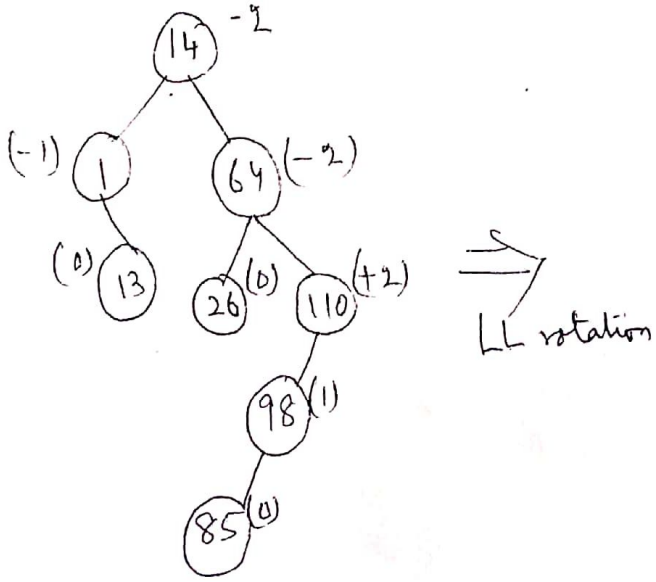
Balanced AVL search tree after rotation (LR)

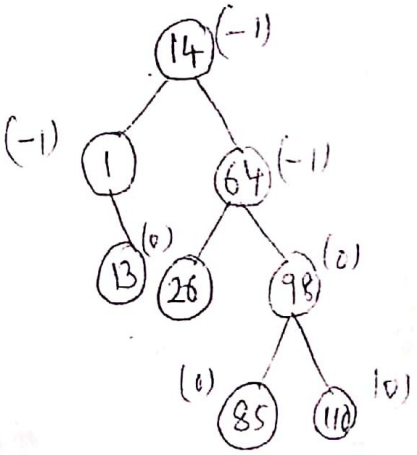
Example:- Construct an AVL search tree by inserting the following elements in the order of their occurance: 64, 1, 14, 26, 13, 110, 98, 85





Insert 85:





Note: The node 110 is the immediate predecessor of node which BF turned to (+2). Hence the new node inserted is in the left subtree of the left subtree of node 98.