AVL TREES

AVL TREES

- AVL tree is a self-balancing binary search tree invented by Adelson, Velsky and Landis in 1962

AVL TREES

- In an AVL tree, the heights of the two sub-trees of a node may differ by at most one.
- Due to this property, the AVL tree is also known as a height-balanced tree.
- The key advantage of using an AVL tree is that it takes **O(log n)** time to perform search, insert, and delete operations in an average case as well as the worst case because the height of the tree is limited to **O(log n).**

AVL TREES

- The structure of an AVL tree is the same as that of a binary search tree but with a little difference. In its structure, it stores an additional variable called the Balance Factor.

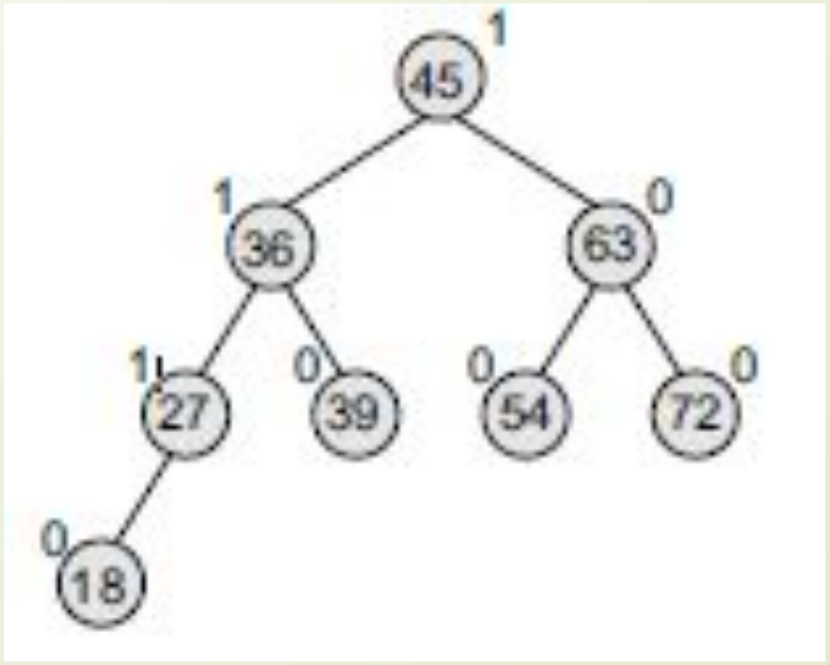
- Every node has a balance factor associated with it. The balance factor of a node is calculated by subtracting the height of its right sub-tree from the height of its left sub-tree.

  Balance factor = Height (left sub-tree) – Height (right sub-tree)

- A binary search tree in which every node has a balance factor of –1, 0, or 1 is said to be height balanced. A node with any other balance factor is considered to be unbalanced and requires rebalancing of the tree.

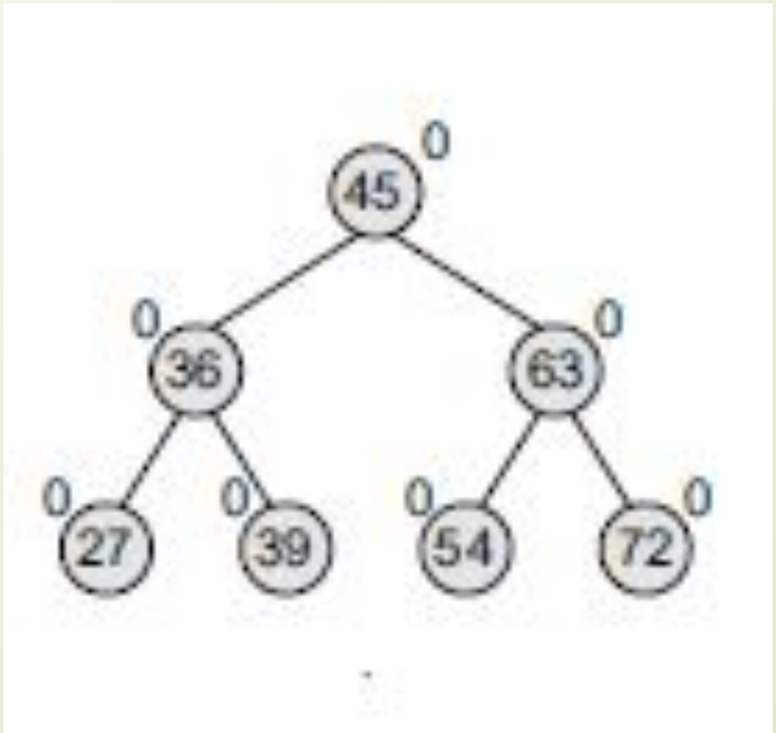
Left-Heavy Tree

- If the balance factor of a node is 1, then it means that the left sub-tree of the tree is one level higher than that of the right sub-tree. Such a tree is therefore called as a left-heavy tree.



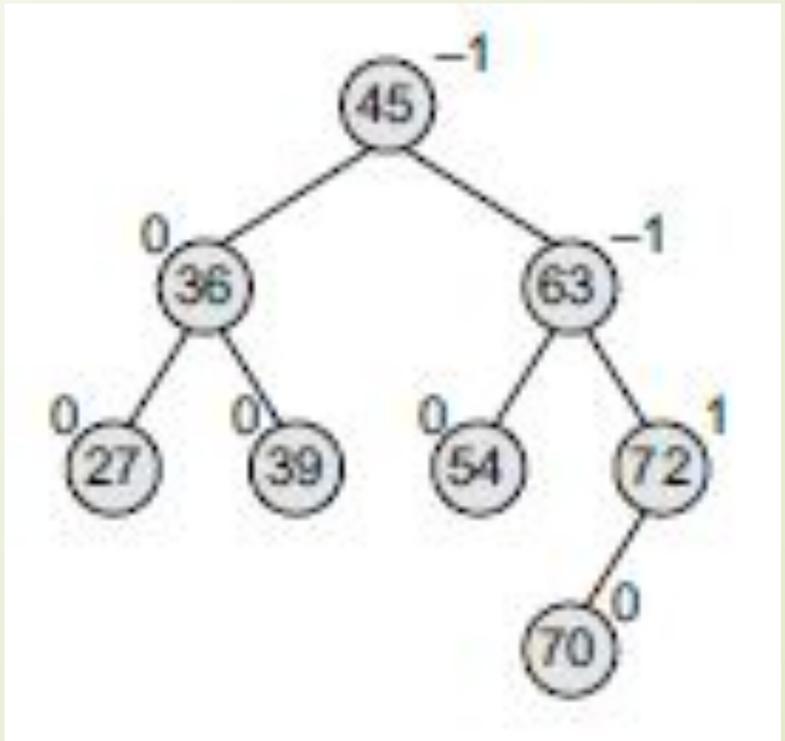
Balanced **AVL TREES**

- If the balance factor of a node is 0, then it means that the height of the left sub-tree (longest path in the left sub-tree) is equal to the height of the right sub-tree.



Right-Heavy tree

- If the balance factor of a node is –1, then it means that the left sub-tree of the tree is one level lower than that of the right sub-tree. Such a tree is therefore called as a right-heavy tree.



Operations on AVL Trees

Searching for a Node in an AVL Tree:

- Searching in an AVL tree is performed exactly the same way as it is performed in a binary search tree.
- Due to the height-balancing of the tree, the search operation takes O(log n) time to complete.
- Since the operation does not modify the structure of the tree, no special provisions are required.

Inserting a New Node

Inserting a New Node in an AVL Tree:

- Insertion in an AVL tree is also done in the same way as it is done in a binary search tree.
- But the step of insertion is usually followed by an additional step of rotation. Rotation is done to restore the balance of the tree.
- However, if insertion of the new node does not disturb the balance factor, that is, if the balance factor of every node is still –1, 0, or 1, then rotations are not required.
- During insertion, the new node is inserted as the leaf node, so it will always have a balance factor equal to zero. The only nodes whose balance factors will change are those which lie in the path between the root of the tree and the newly inserted node.

Inserting a New Node

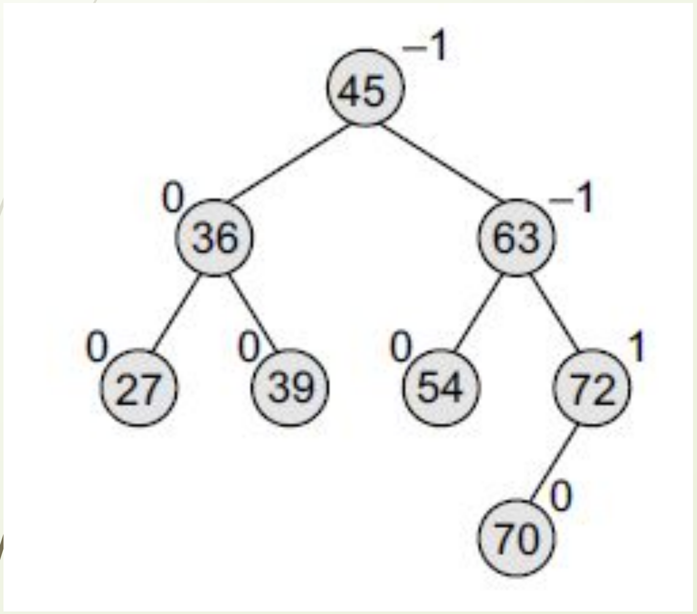
Inserting a New Node in an AVL Tree:

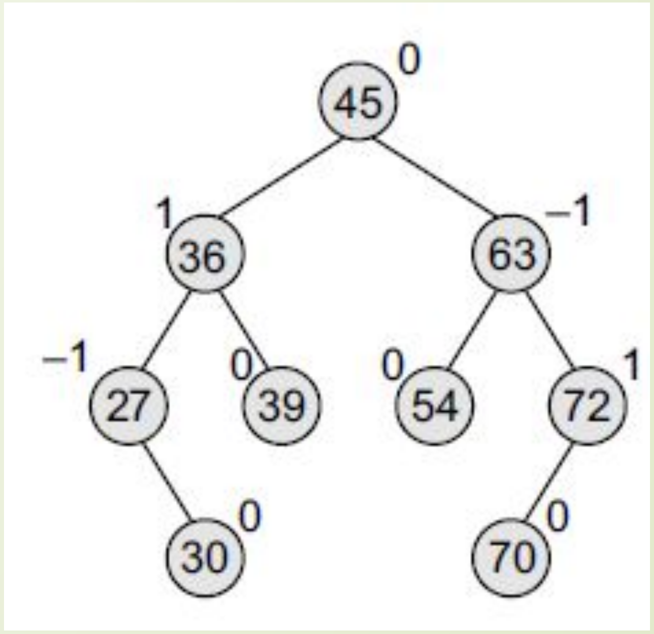
The possible changes which may take place in any node on the path are as follows:

- Initially, the node was either left- or right-heavy and after insertion, it becomes balanced.
- Initially, the node was balanced and after insertion, it becomes either left-heavy or right-heavy.
- Initially, the node was heavy (either left or right) and the new node has been inserted in the heavy sub-tree, thereby creating an unbalanced sub-tree. Such a node is said to be a ***critical node***.

Inserting a New Node

Insert 30



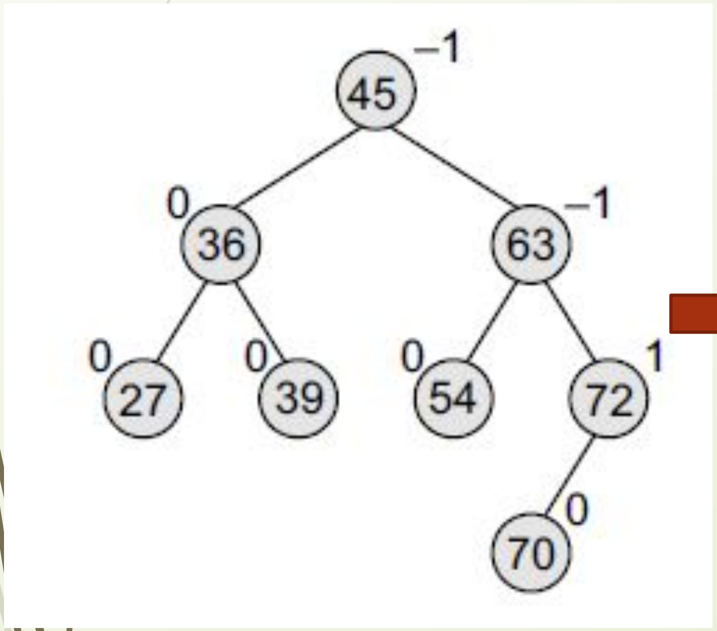


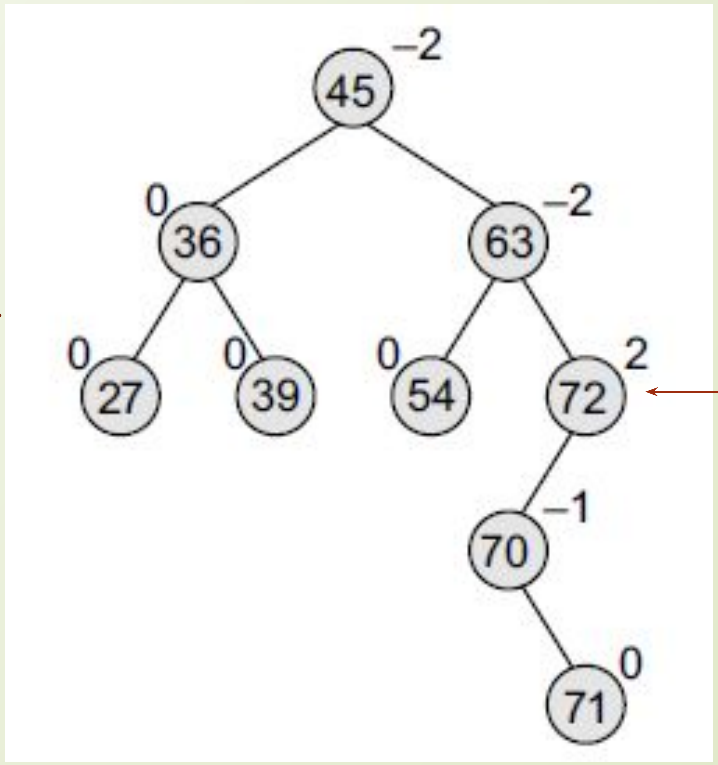
Balanced

Balanced

Inserting a New Node

Insert 71





Critical No

Balanced

Not Balanced

Rotations in AVL tree

- ***LL rotation:***

  The new node is inserted in the left sub-tree of the left sub-tree of the critical node.

- ***RR rotation:***

  The new node is inserted in the right sub-tree of the right sub-tree of the critical node.

- ***LR rotation:***

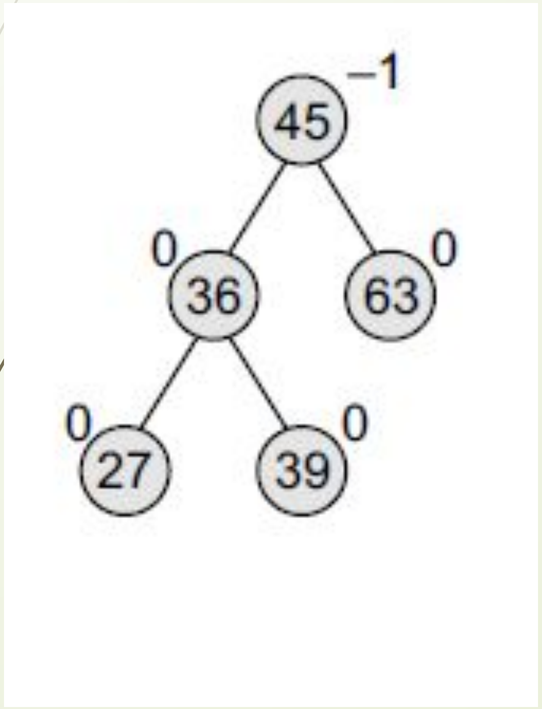
  The new node is inserted in the right sub-tree of the left sub-tree of the critical node.

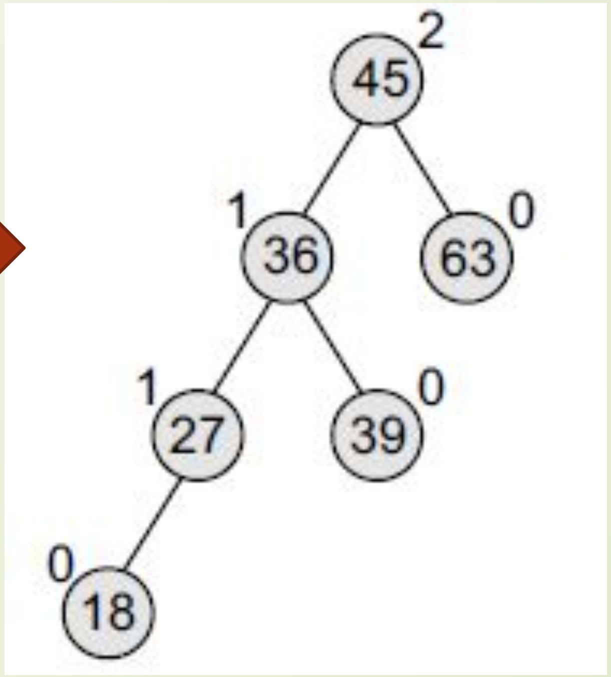
- ***RL rotation:***

  The new node is inserted in the left sub-tree of the right sub-tree of the critical node.

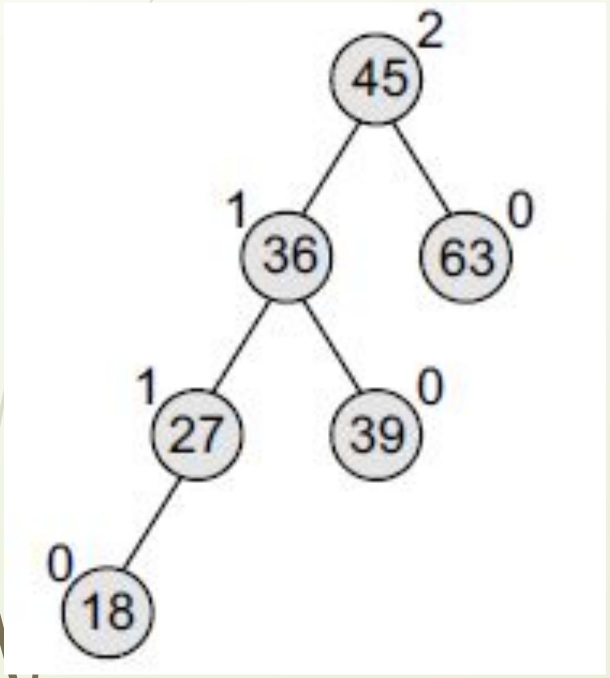
LL rotation in AVL tree

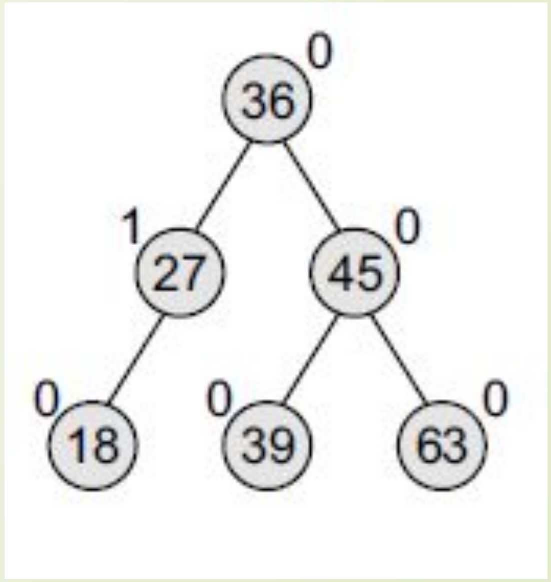
Insert 18





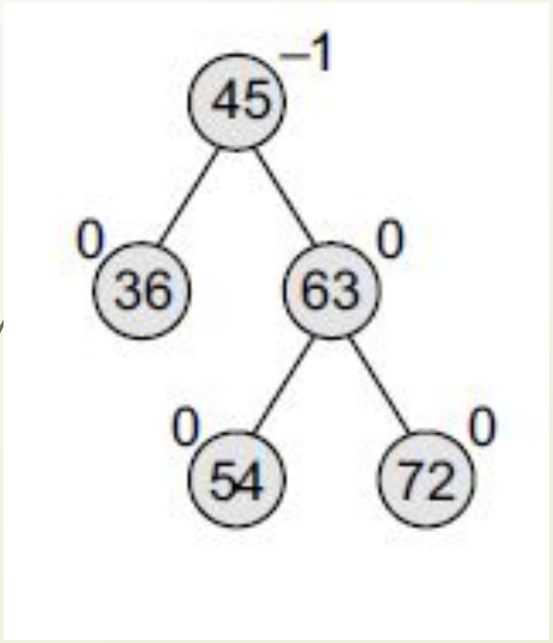
LL Rotations in AVL Tree

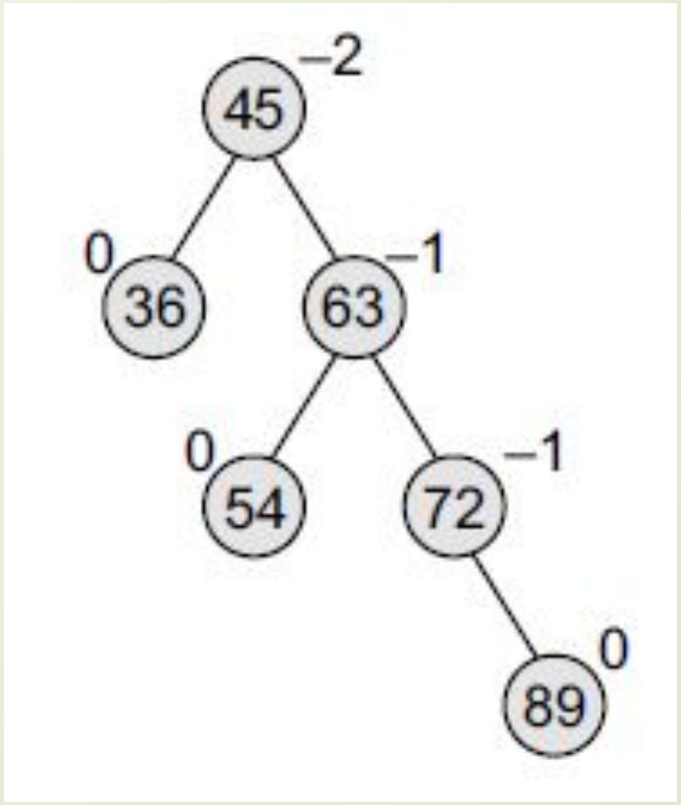




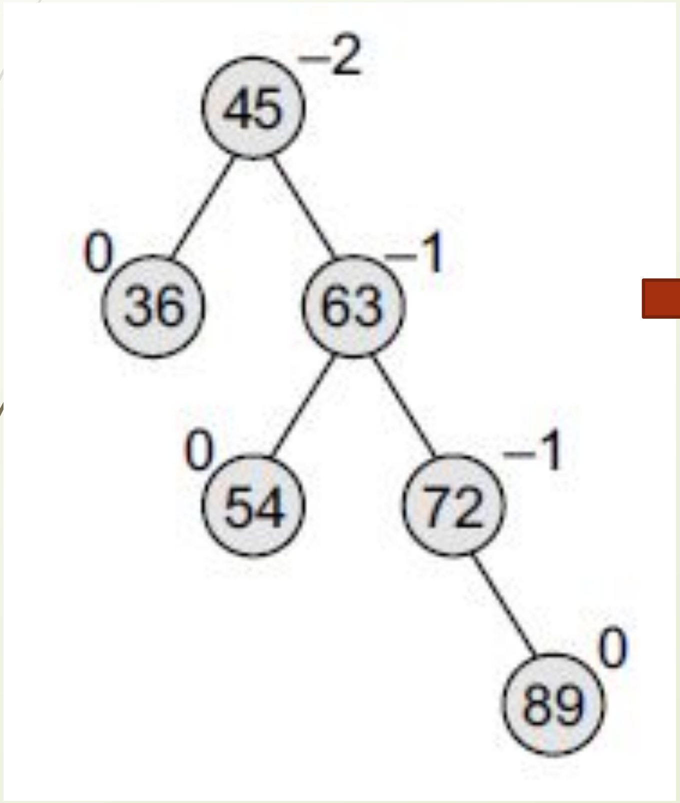
RR rotation in AVL Tree

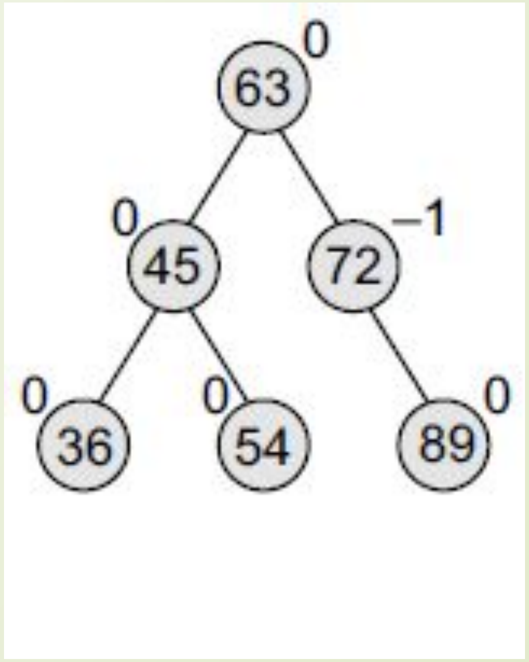
Insert 89





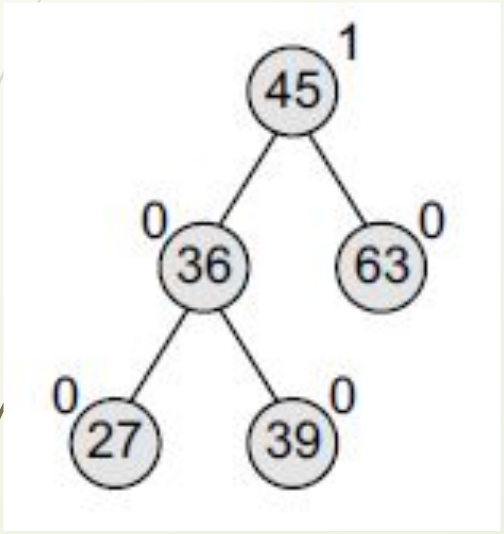
RR Rotations in AVL Tree

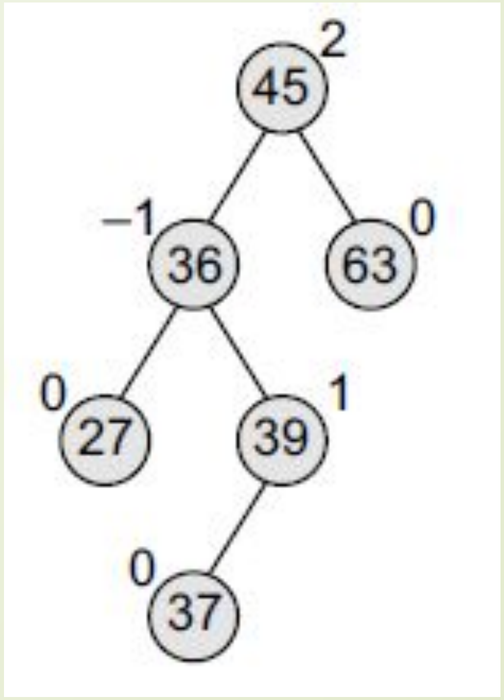




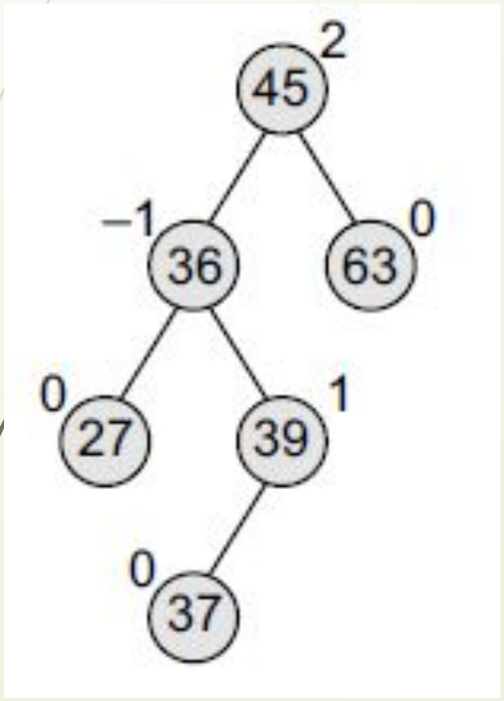
LR Rotation in AVL Tree

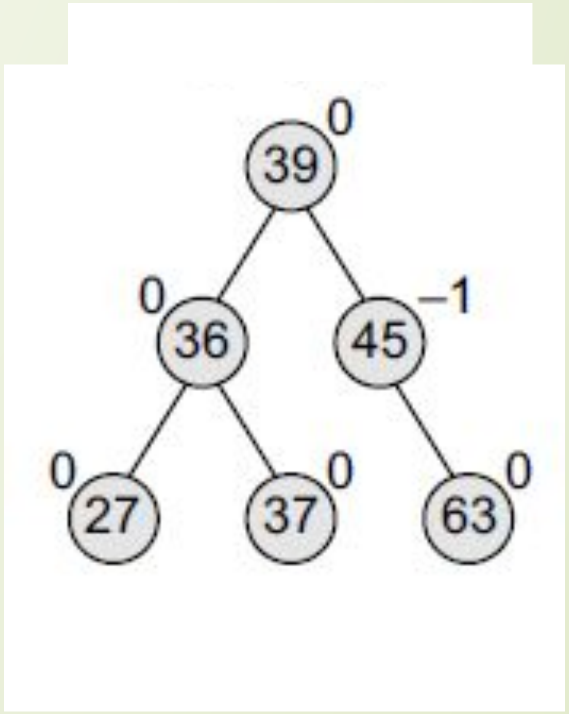
Insert 37





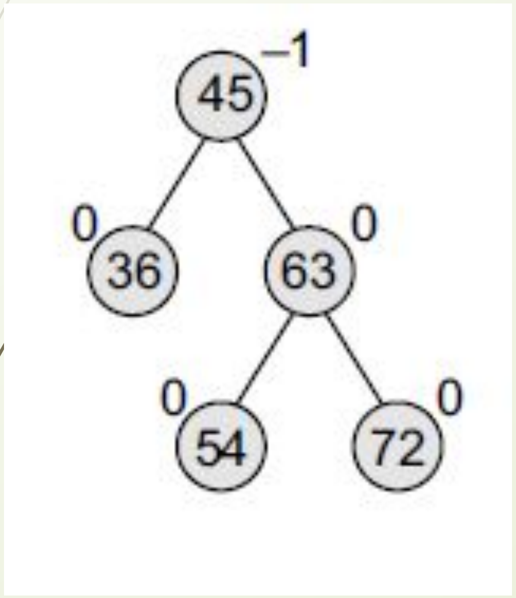
LR Rotation in AVL Tree

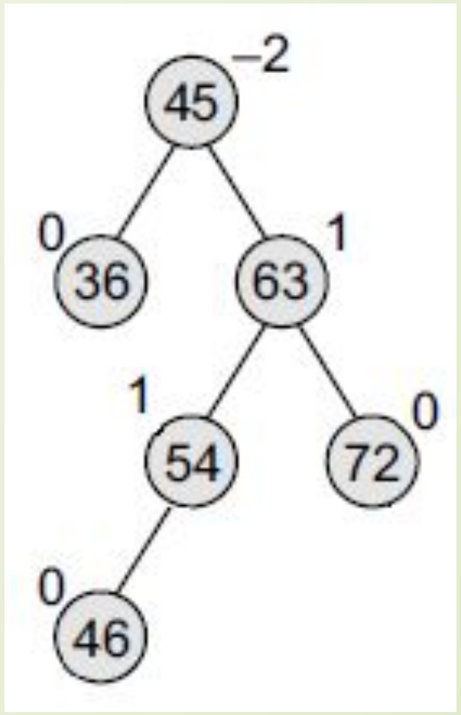




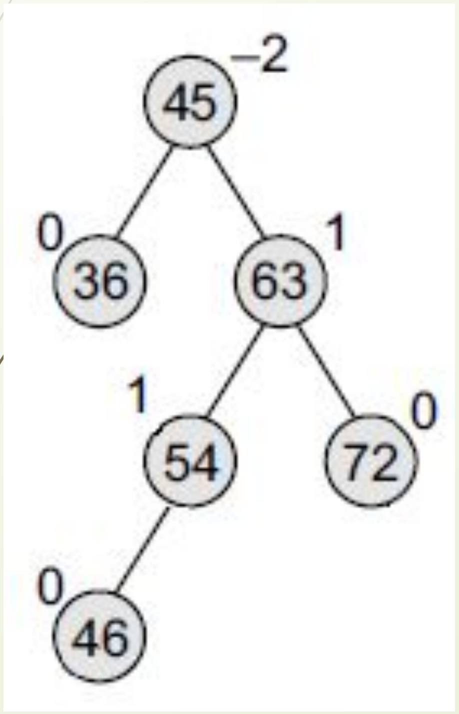
RL Rotation in AVL Tree

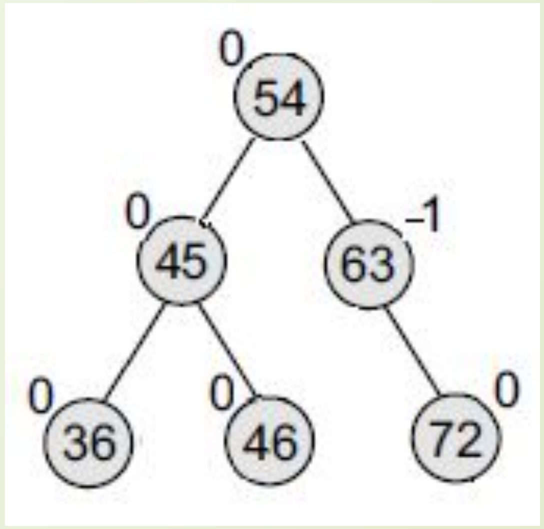
Insert 46





RL Rotation in AVL Tree



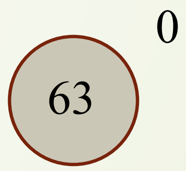


Example to Construct an AVL Tree

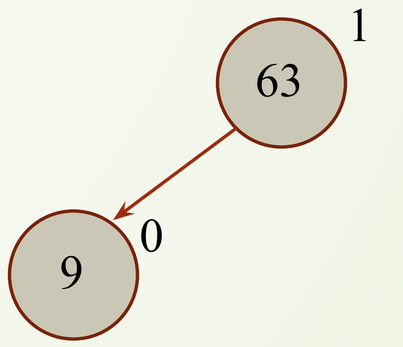
- Construct an AVL tree by inserting the following elements in the given order.

  63, 9, 19, 27, 18, 108, 99, 81.

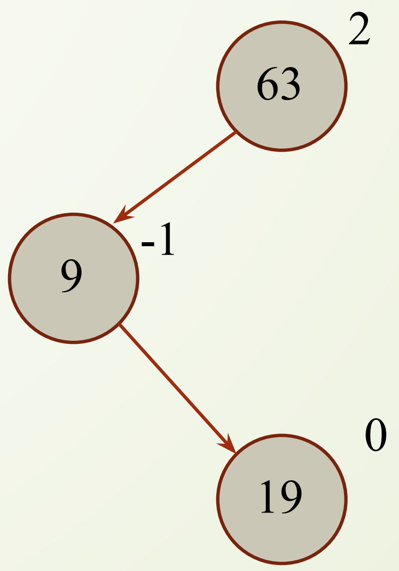
63, 9, 19, 27, 18, 108, 99, 81



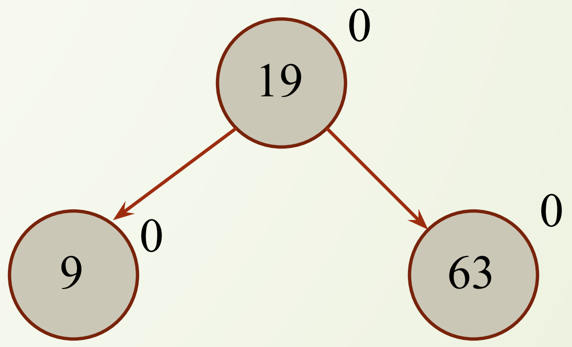
63, 9, 19, 27, 18, 108, 99, 81



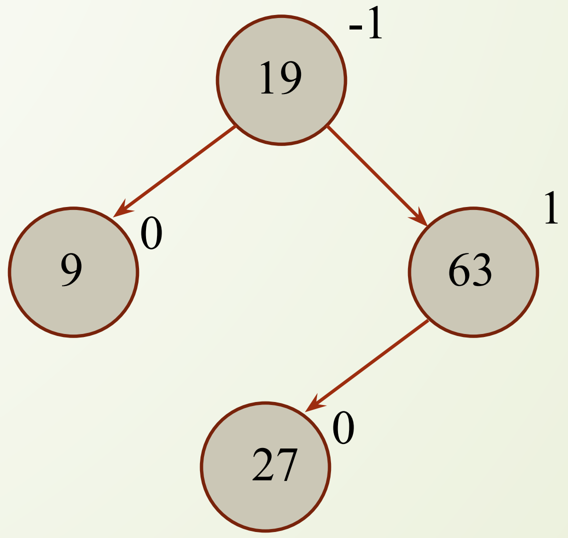
63, 9, 19, 27, 18, 108, 99, 81



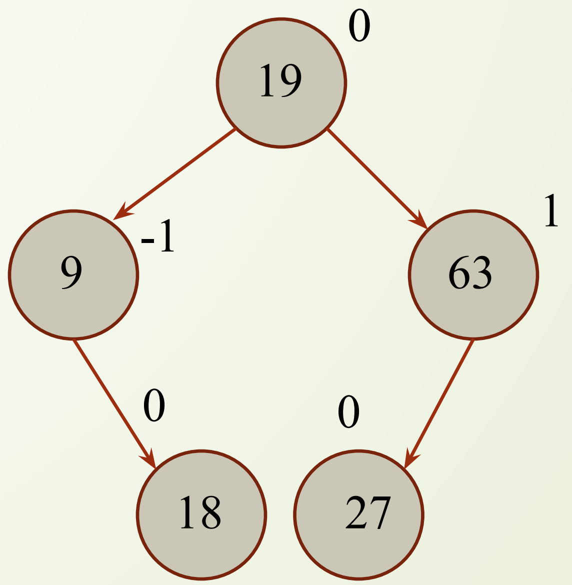
63, 9, 19, 27, 18, 108, 99, 81



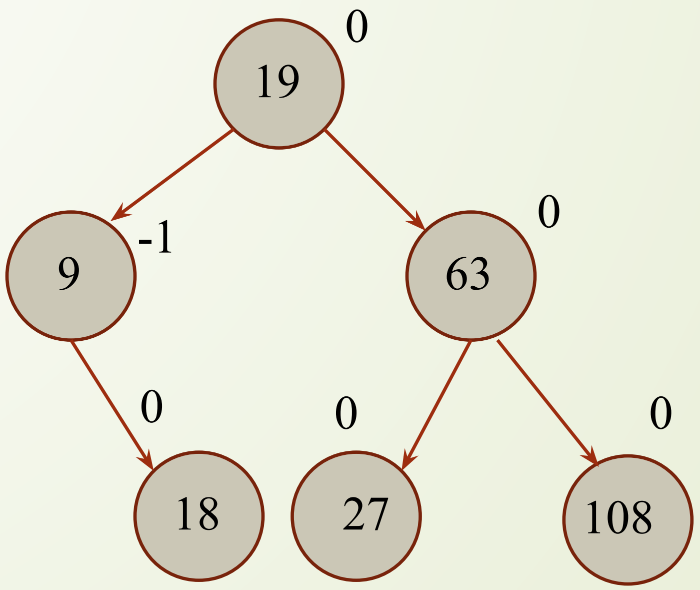
63, 9, 19, 27, 18, 108, 99, 81



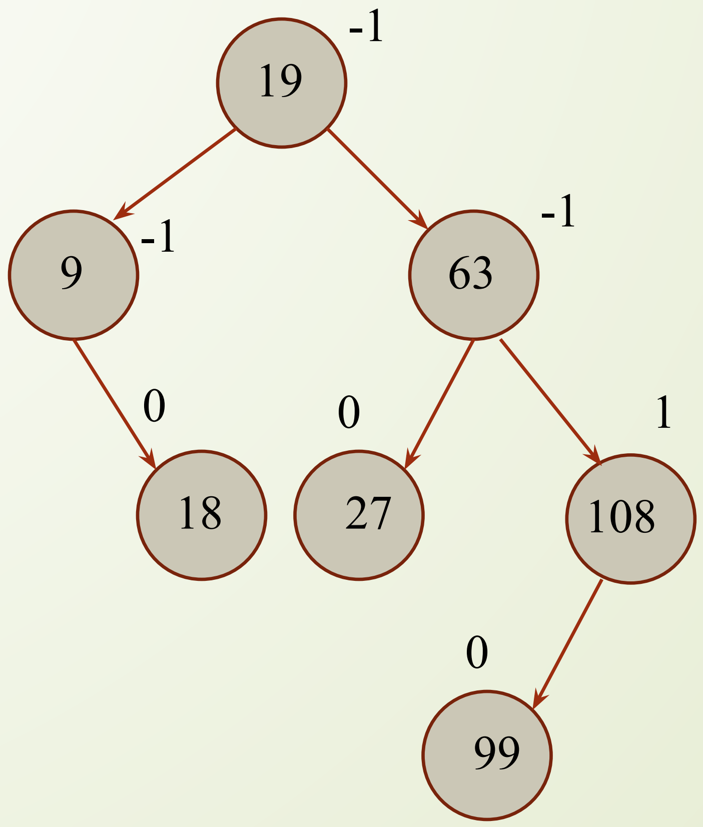
63, 9, 19, 27, 18, 108, 99, 81



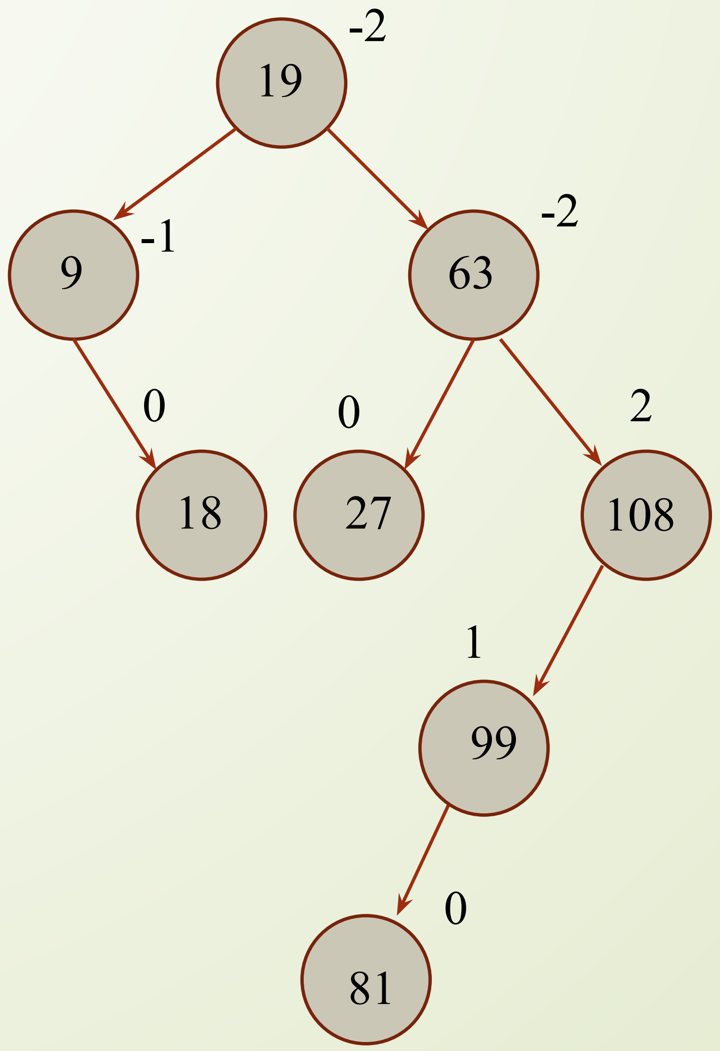
63, 9, 19, 27, 18, 108, 99, 81



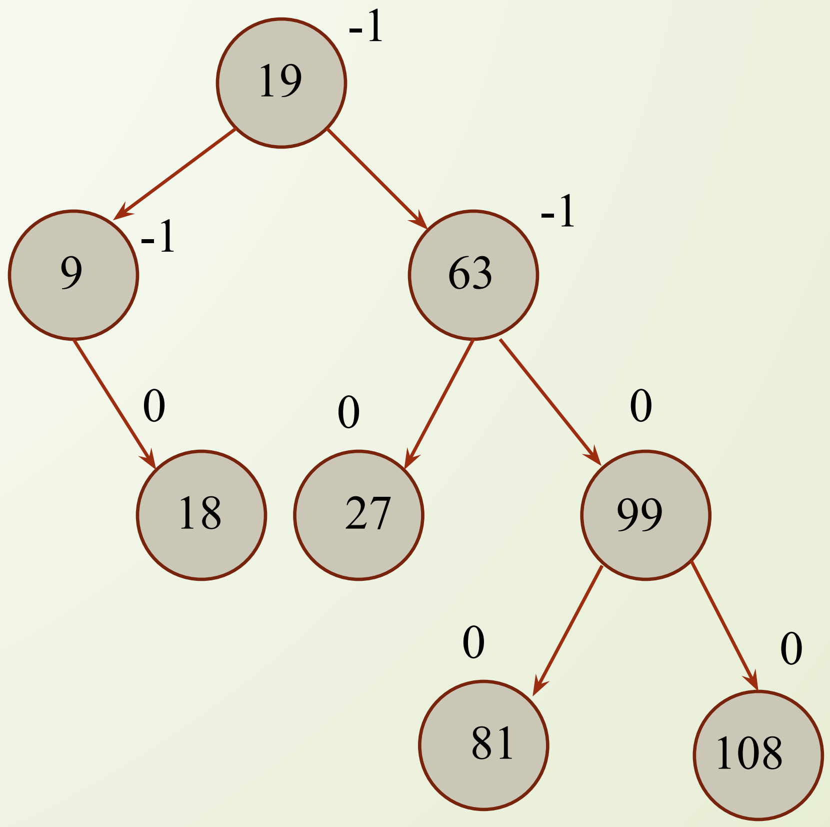
63, 9, 19, 27, 18, 108, 99, 81

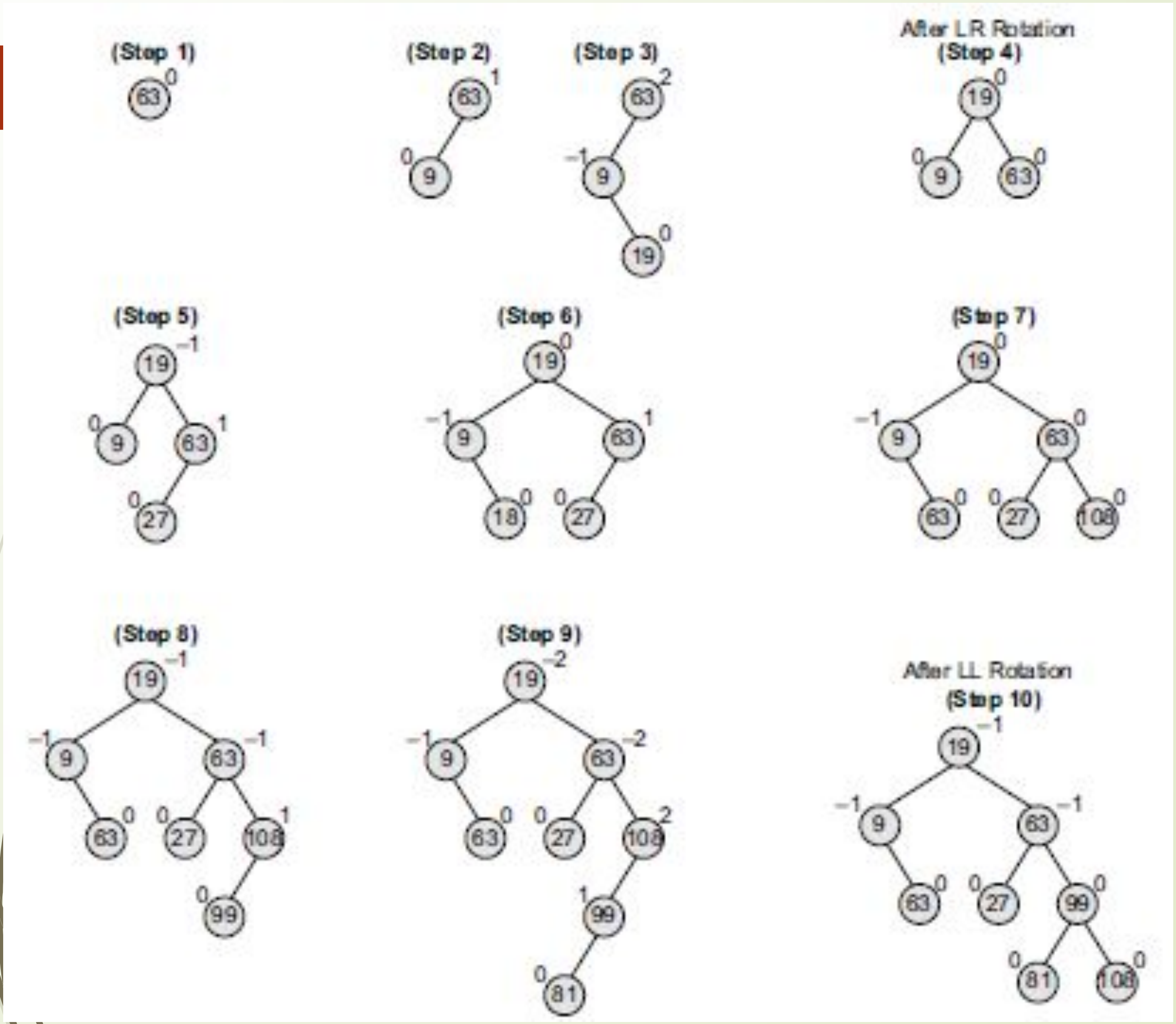


63, 9, 19, 27, 18, 108, 99, 81



63, 9, 19, 27, 18, 108, 99, 81




(Step 1)
(Step 2)
(Step 3)
After LR Rotation
(Step 4)
(Step 5)
(Step 6)
(Step 7)
(Step 8)
(Step 9)
After LL Rotation
(Step 10)