Parul® University

# UNIT 6
# AVL (Adelson-Velskii and Landis)-Tree

**Gulshan Sharma,** Professor
Computer Science & Engineering

# UNIT-6

## AVL

# Red Black Trees

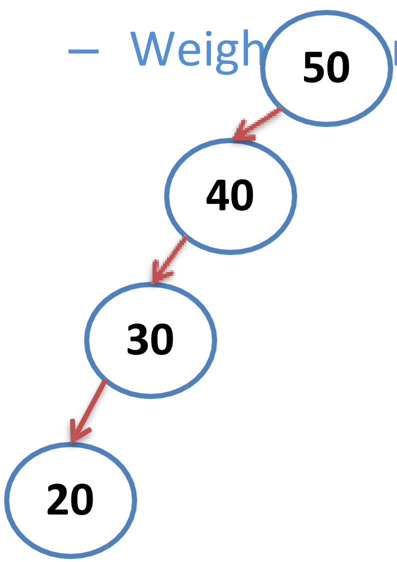
**Red-Black Trees (RBTs) are a type of self-balancing binary search tree (BST). They ensure that the tree remains approximately balanced, providing good performance for insertion, deletion, and search operations. Here's an overview of the key operations on Red-Black Trees:**

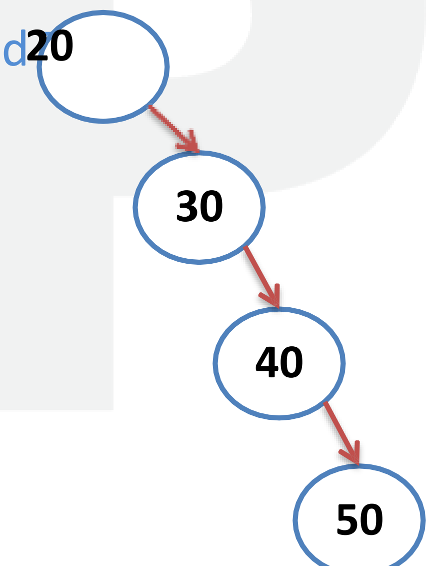
# Properties of Red-Black Trees

- **Node Color**: Each node is either red or black.
- **Root Property**: The root is always black.
- **Leaf Property**: Every leaf (NIL node) is black.
- **Red Property**: If a red node has children, then the children are always black (no two red nodes can be adjacent).
- **Depth Property**: Every path from a given node to its descendant NIL nodes has the same number of black nodes (black-height).

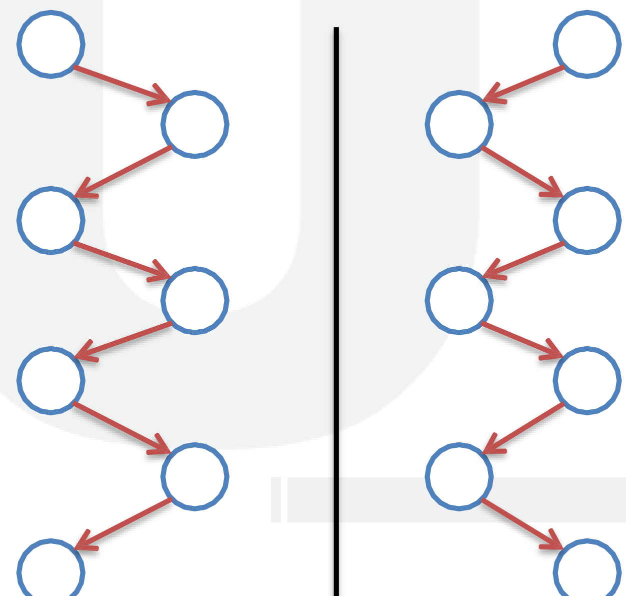
# Balanced

- Binary Search Tree gives advantage of Fast Search, but sometimes in few cases we are not able to get this advantage. E.g. look into worst case BST
- Balanced binary trees are classified into two categories
  - Height Balanced Tree (AVL Tree)
  - Weigh nced

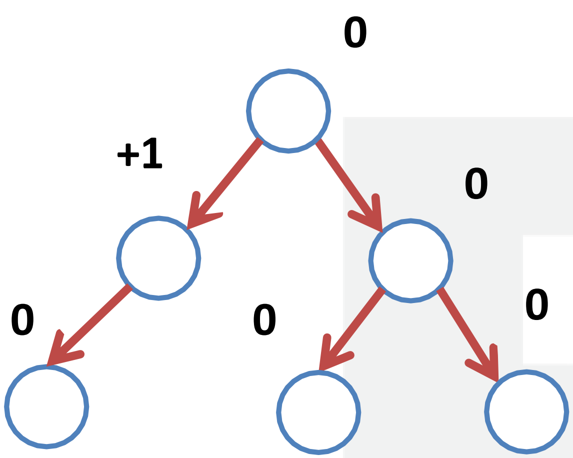


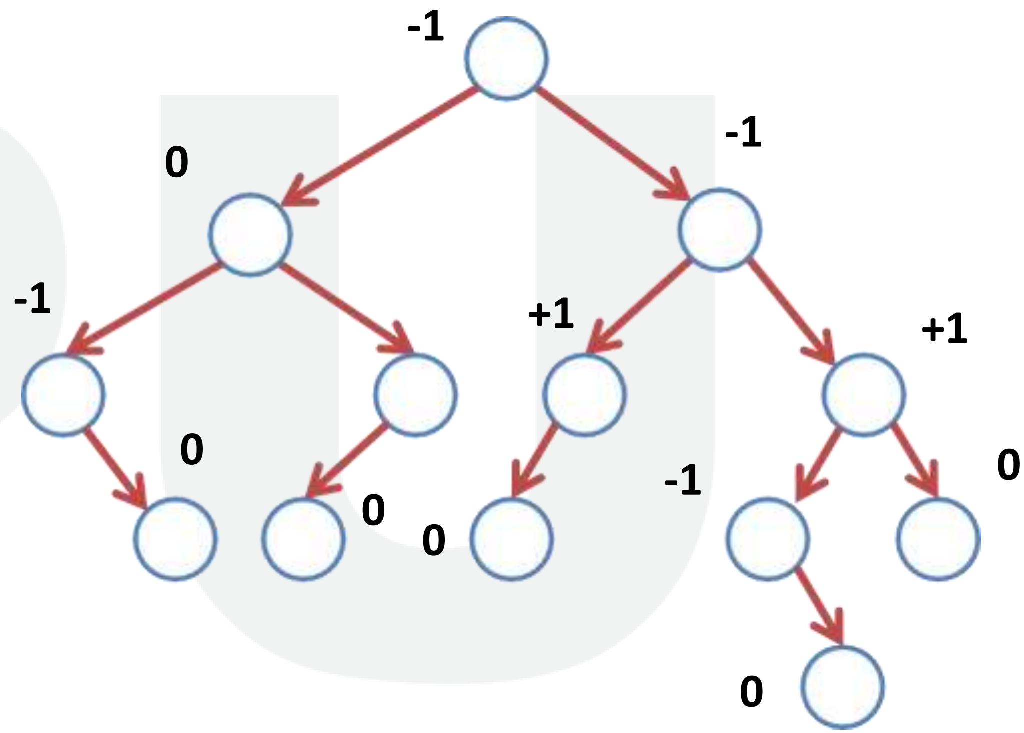




**Worst search time cases for Binary Search Tree**



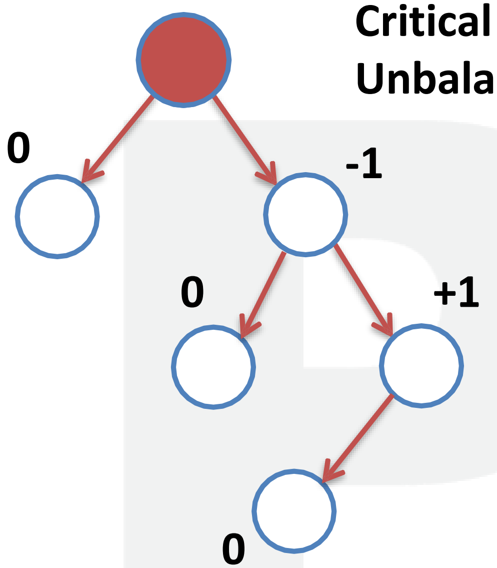
# AVL



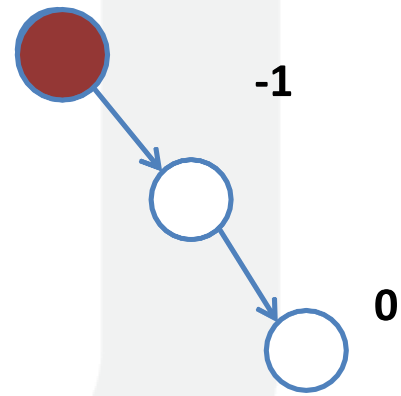



# AVL

**Critical Node**
**Unbalanced Node**




**Critical Node**
**Unbalanced Node**




- Sometimes tree becomes unbalanced by inserting or deleting any node
- Then based on position of insertion, we need to rotate the unbalanced node
- **Rotation** is the **process** to **make tree balanced**

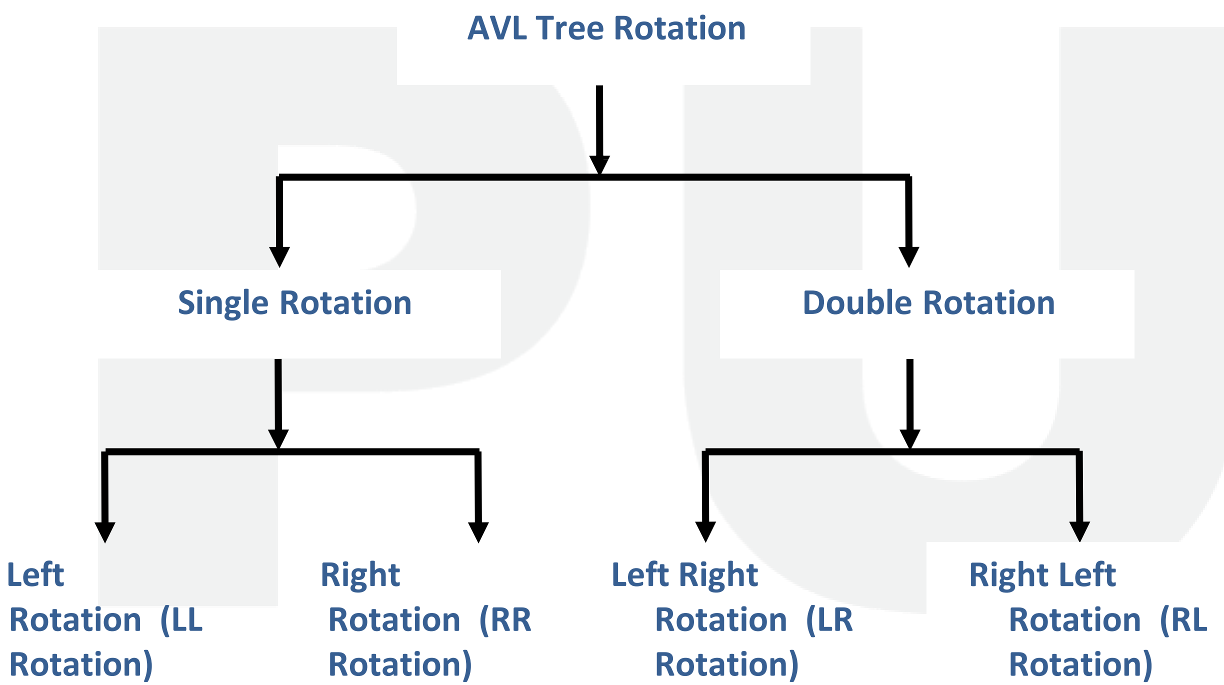
# AVL Tree

The following operations are performed on AVL tree...

- **Search**
- **Insertion**
- **Deletion**

# AVL Tree

There are **four** rotations and they are classified into **two** types.



# Operation on an

- In AVL tree, after performing operations like insertion and deletion we need to check the **balance factor** of every node in the tree. Permissible balance factors:
- If every node satisfies the balance factor condition then we conclude the operation otherwise we must make it balanced.
- If there exists a node in a tree where this is not true, then such a tree is called **Unbalanced**

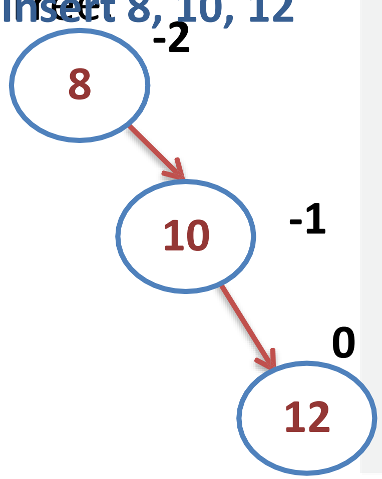
***Rotation operations are used to make the tree balanced.***

**Rotation is the process of moving nodes either to left or to right to make the tree balanced.**

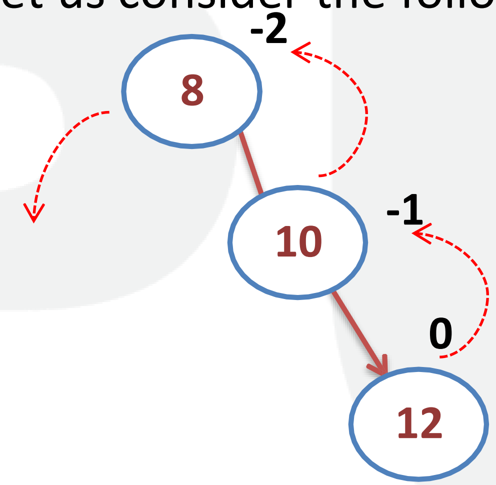
# Single Left Rotation (LL Rotation )

- In LL Rotation, every node moves one position to left from the current position.
- To understand LL Rotation, let us consider the following insertion operation in AVL Tree

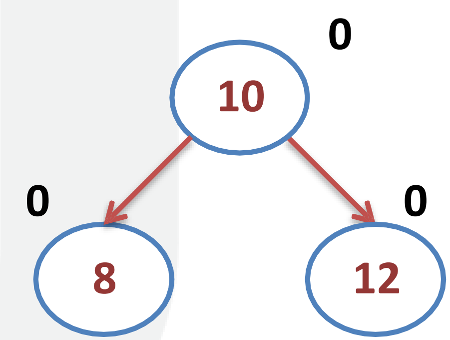
Insert 8, 10, 12




Tree is imbalanced




To make balanced we use LL Rotation which moves nodes one position to left

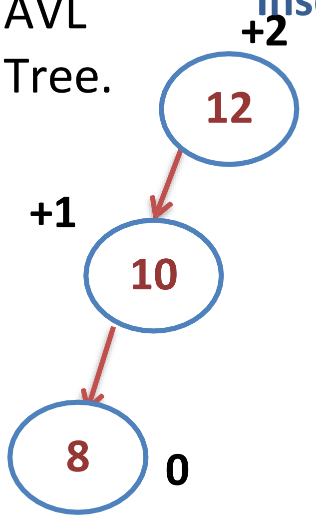



After LL Rotation Tree is balanced

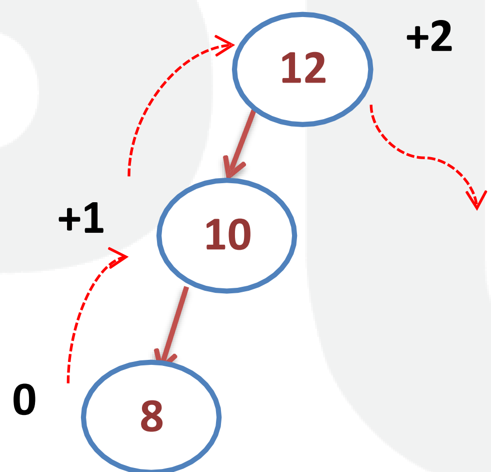
# Single Right Rotation (RR Rotation )

- In RR Rotation, every node moves one position to right from the current position.
- To understand RR Rotation, let us consider the following insertion operation in AVL Tree.

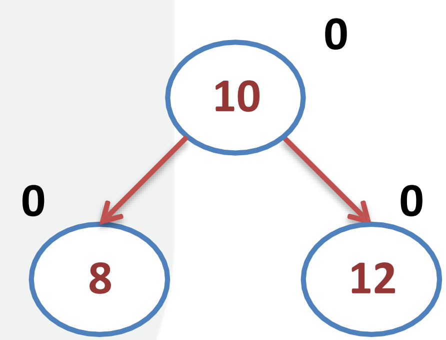
Insert 12, 10, 8




Tree is imbalanced



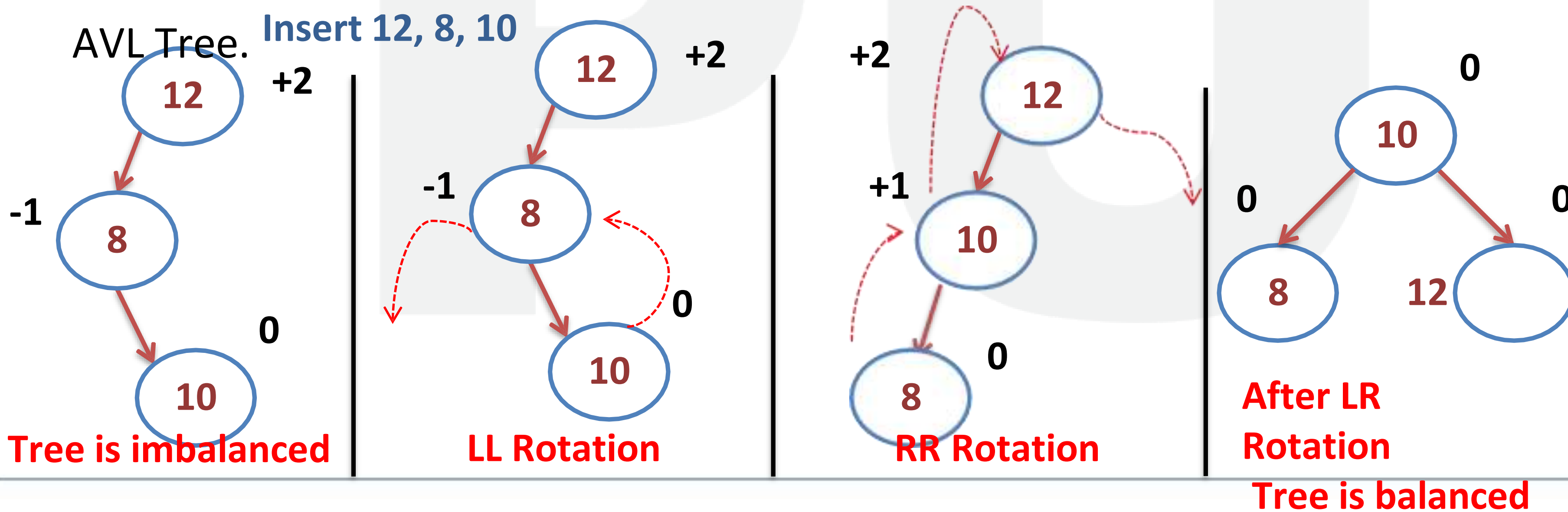

To make balanced we use RR Rotation which moves nodes one position to Right




After RR Rotation Tree is balanced

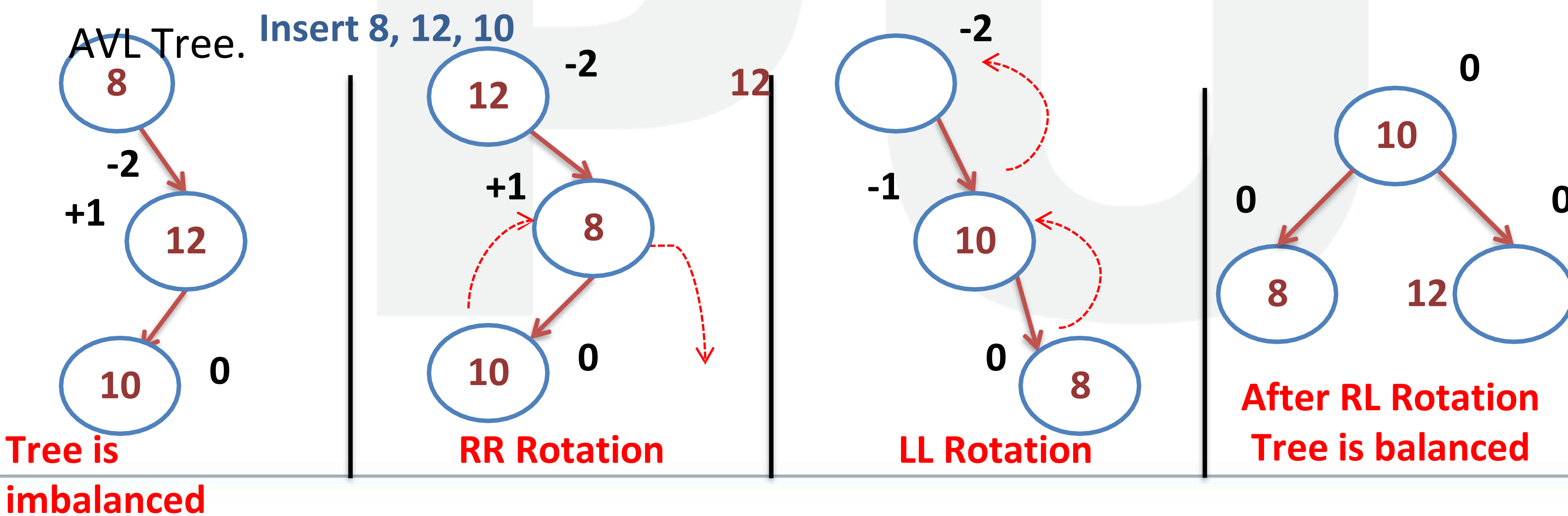
# Left Right Rotation (LR Rotation )

- The LR Rotation is a sequence of single left rotation followed by a single right rotation
- In LR Rotation, at first, every node moves one position to the left and one position to right from the current position.
- To understand LR Rotation, let us consider the following insertion operation in AVL Tree.



# Right Left Rotation (LR Rotation )

- The RL Rotation is a sequence of single right rotation followed by a single left rotation
- In RL Rotation, at first, every node moves one position to the right and one position to left from the current position.
- To understand RL Rotation, let us consider the following insertion operation in AVL Tree.



# AVL Tree

The following operations are performed on AVL tree...

- **Search**
- **Insertion**
- **Deletion**

# Search Operation in AVL

- In an AVL tree, the search operation is performed with **O(log n)** time complexity.
- The search operation in the AVL tree is similar to the search operation in a Binary search tree.
- We use the following steps to search an element in AVL tree.

**Step 1 -** Read the search element from the user.

**Step 2 -** Compare the search element with the value of root node in the tree.

**Step 3 -** If both are matched, then display "Given node is found!!!" and terminate the function

**Step 4 -** If both are not matched, then check whether search element is smaller or larger than that node value.

# Search Operation in AVL

**Step 5 -** If search element is smaller, then continue the search process in left subtree.

**Step 6 -** If search element is larger, then continue the search process in right subtree.

**Step 7 -** Repeat the same until we find the exact element or until the search element is compared with the leaf node.

**Step 8 -** If we reach to the node having the value equal to the search value, then display "Element is found" and terminate the function.

**Step 9 -** If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.

# Insertion Operation in AVL

- In an AVL tree, the insertion operation is performed with **O(log n)** time complexity.
- In AVL Tree, a new node is always inserted as a leaf node.
- The insertion operation is performed as follows.

**Step 1 -** Insert the new element into the tree using Binary Search Tree insertion logic.

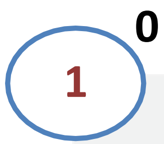
**Step 2 -** After insertion, check the **Balance Factor** of every node.

**Step 3 -** If the **Balance Factor** of every node is **0 or 1 or -1** then go for next operation.

**Step 4 -** If the **Balance Factor** of any node is other than **0 or 1 or -1** then that tree is said to be imbalanced. In this case, perform suitable **Rotation** to make it balanced and go for next operation.

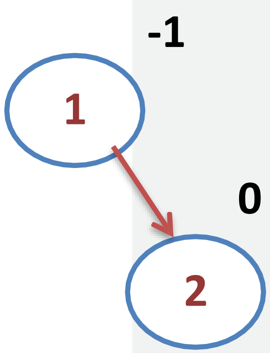
# Construct an AVL tree by inserting numbers from 1 to

## Insert 1




**Tree is balanced**

## Insert 2



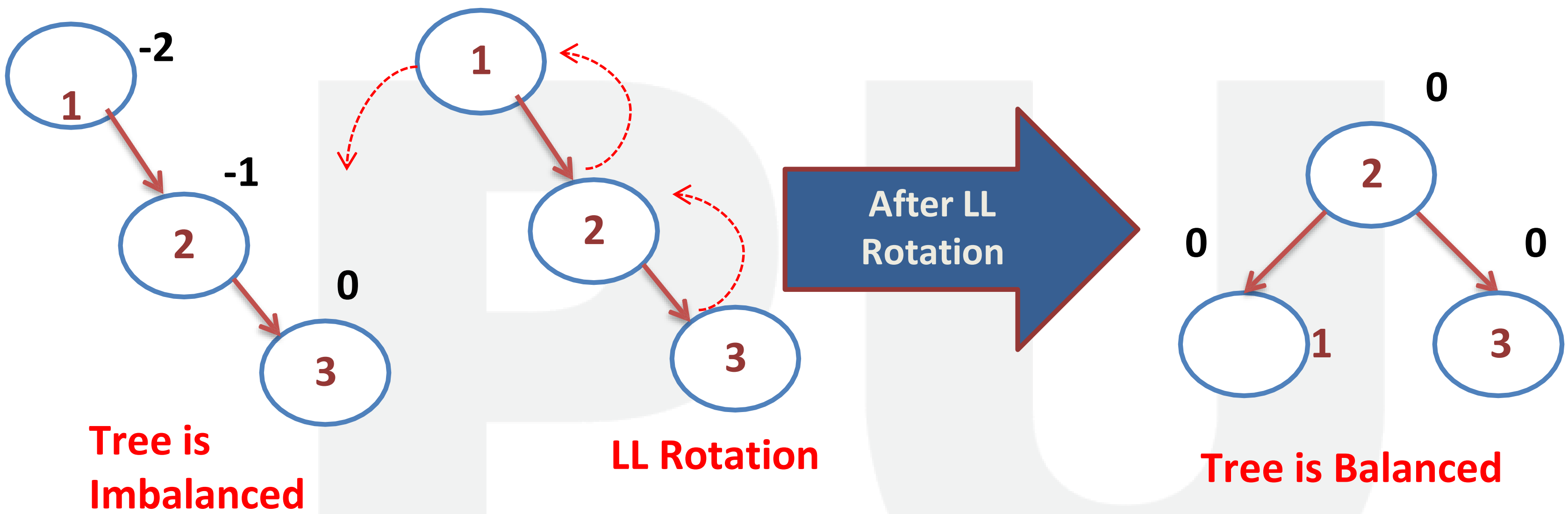

**Tree is balanced**

Parul® University

DIGITAL LEARNING CONTENT

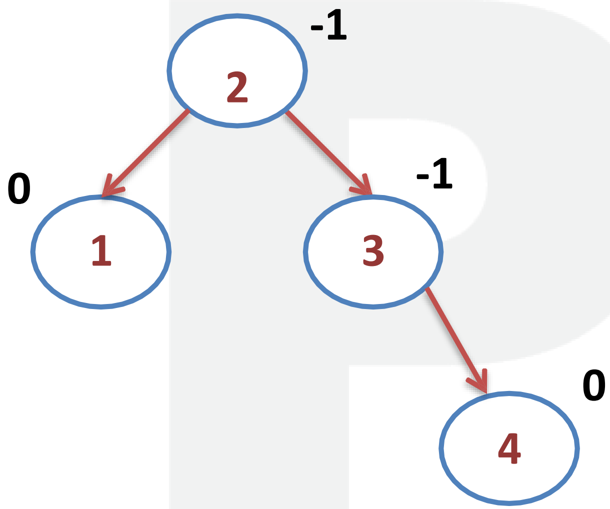
# Construct an AVL tree by inserting numbers from 1 to

## Insert 3



# Construct an AVL tree by inserting numbers from 1 to

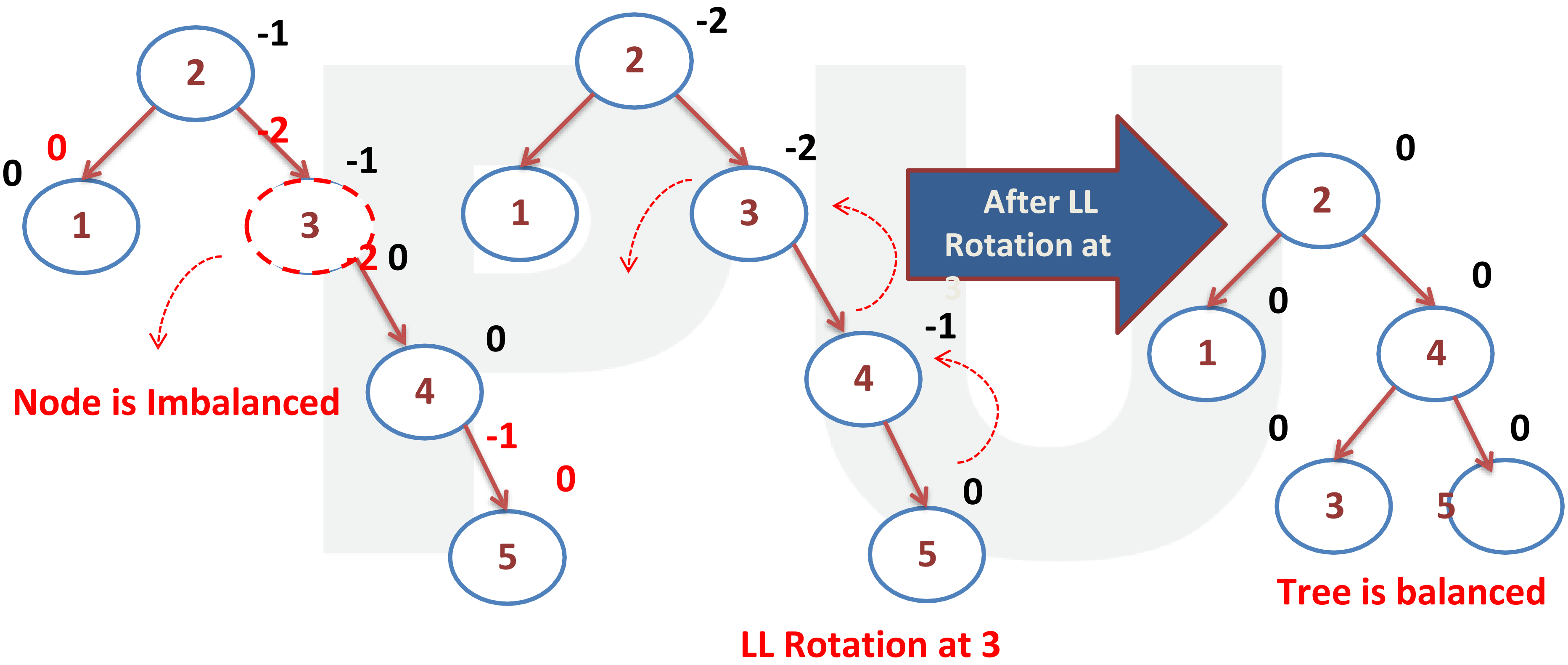
## Insert 4




Tree is balanced

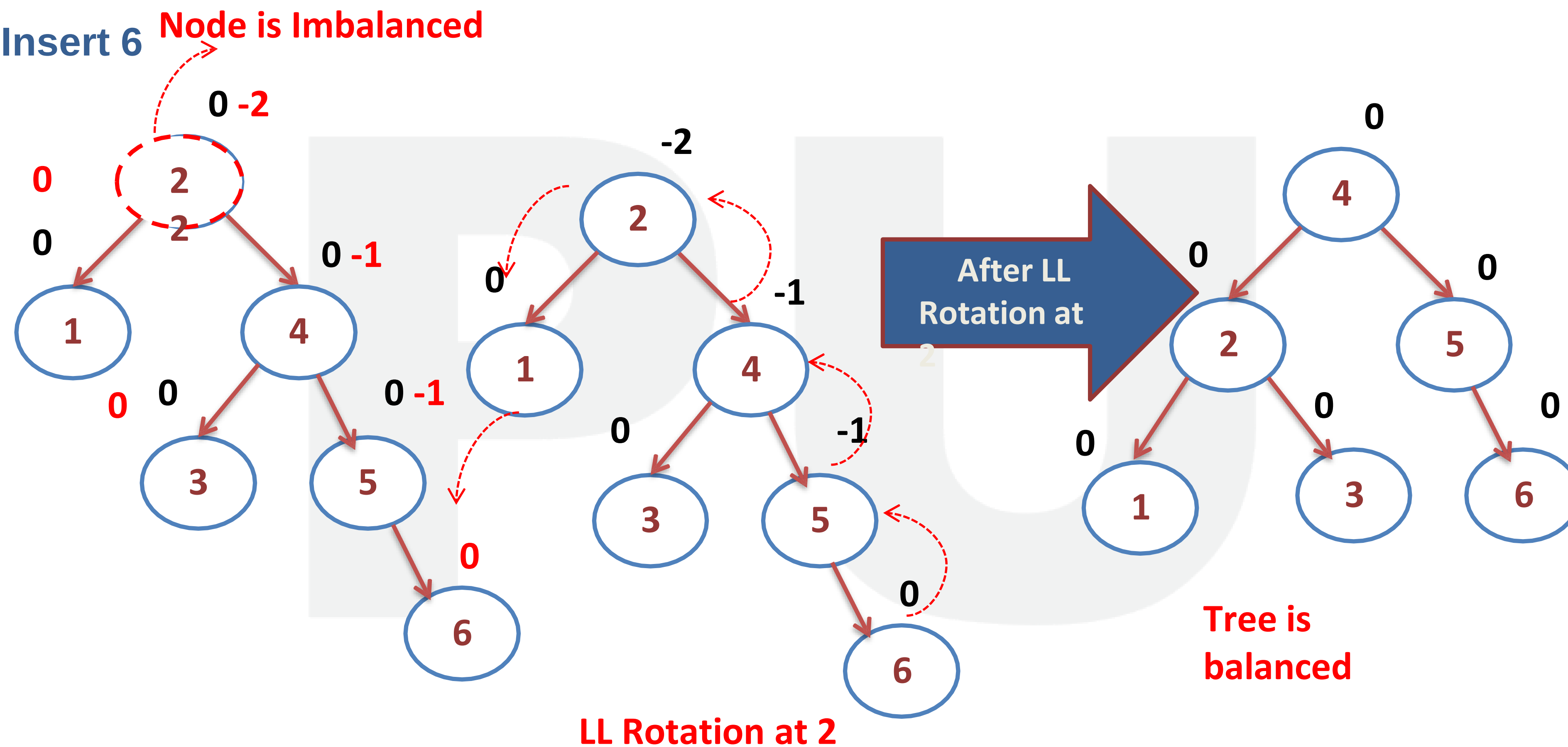
# Construct an AVL tree by inserting numbers from 1 to

## Insert 5



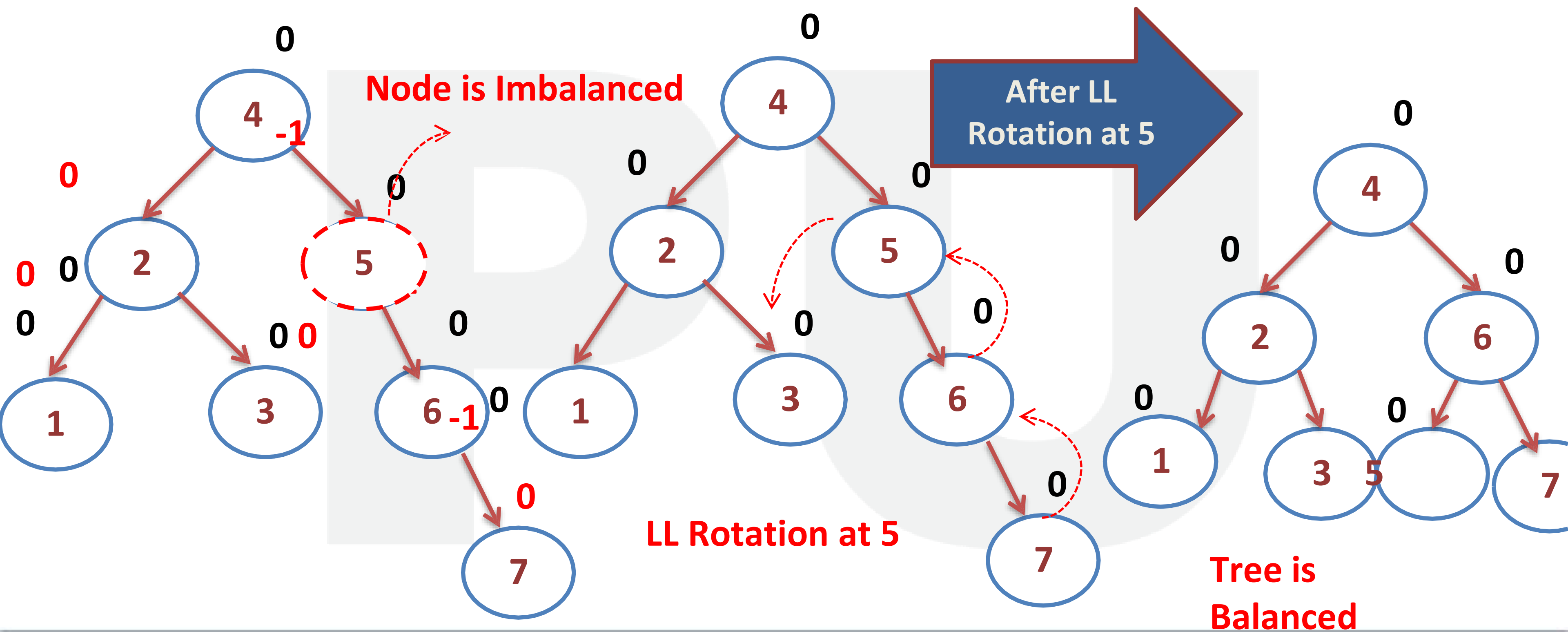
# Construct an AVL tree by inserting numbers from 1 to

Insert 6



# Construct an AVL tree by inserting numbers from 1

## Insert 7



# Deletion Operation in AVL

- The deletion operation in AVL Tree is similar to deletion operation in BST.
- But after every deletion operation, we need to check with the Balance Factor condition.
- If the tree is balanced after deletion go for next operation otherwise perform suitable rotation to make the tree Balanced.

# DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.i