AVL Tree

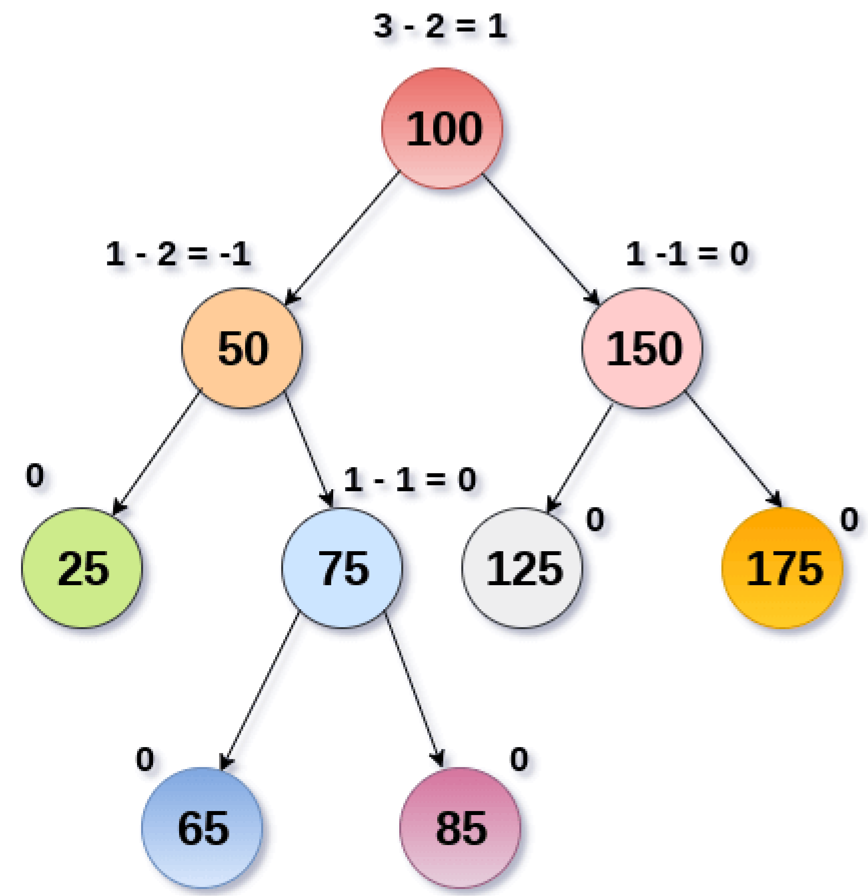
Introduction

- AVL Tree
  - Special kind of binary search tree
  - Invented by Adelson, Velsky and Landis in 1962
  - Defined as height-balanced binary search tree
    - Each node is associated with a balance factor
      - Calculated by subtracting the height of its right sub-tree from that of its left sub-tree
      - Balance Factor (k) = height(left(k)) - height(right(k))
  - Tree is said to be balanced
    - if balance factor of each node is either -1 or 0 or 1, otherwise, the tree will be unbalanced and need to be balanced

Cont…

- In an AVL Tree
  - If balance factor of any node is -1, it means that the left sub-tree is one level lower than the right sub-tree
  - If balance factor of any node is 0, it means that the left sub-tree and right sub-tree contain equal height
  - If balance factor of any node is 1, it means that the left sub-tree is one level higher than the right sub-tree

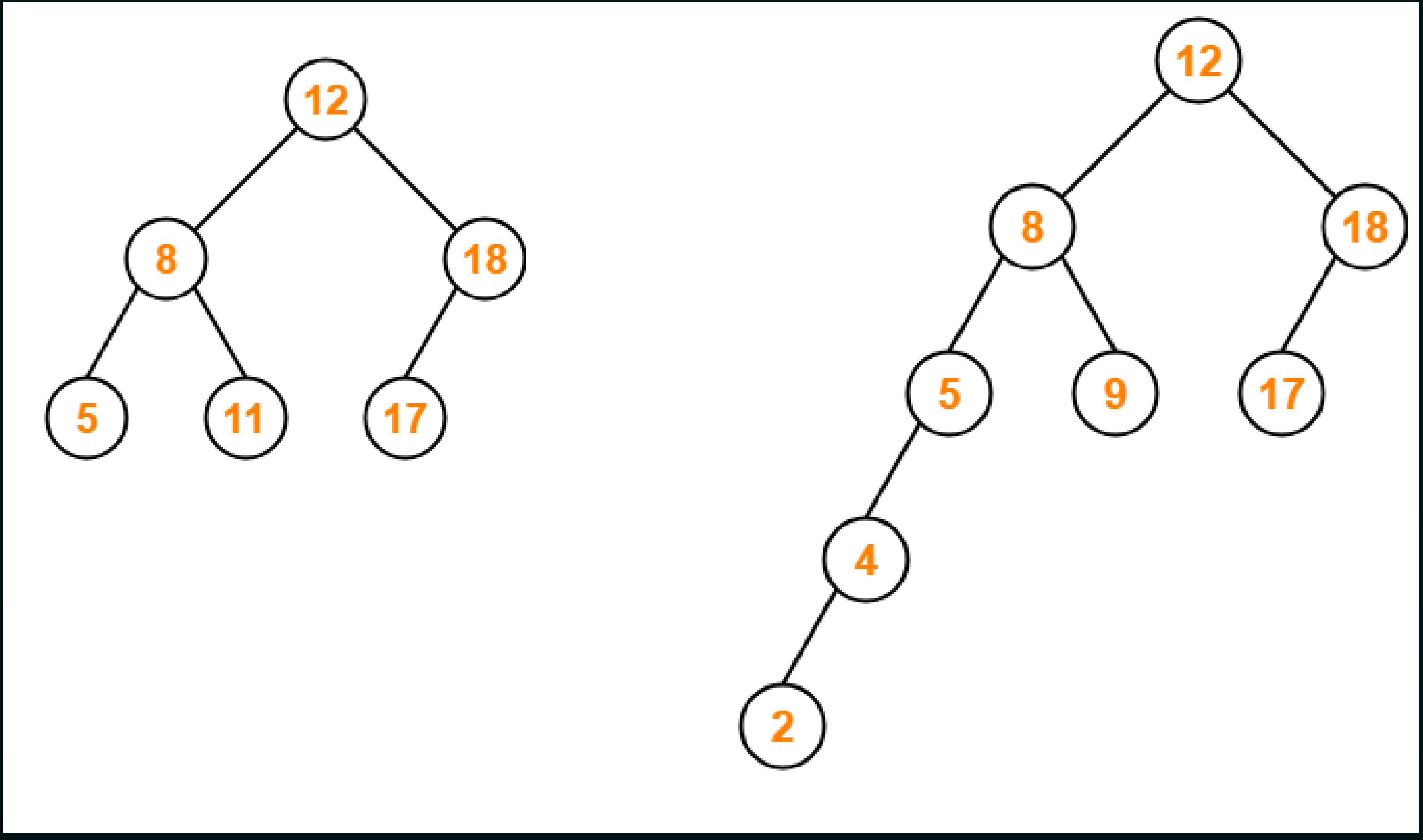
Cont…



AVL Tree

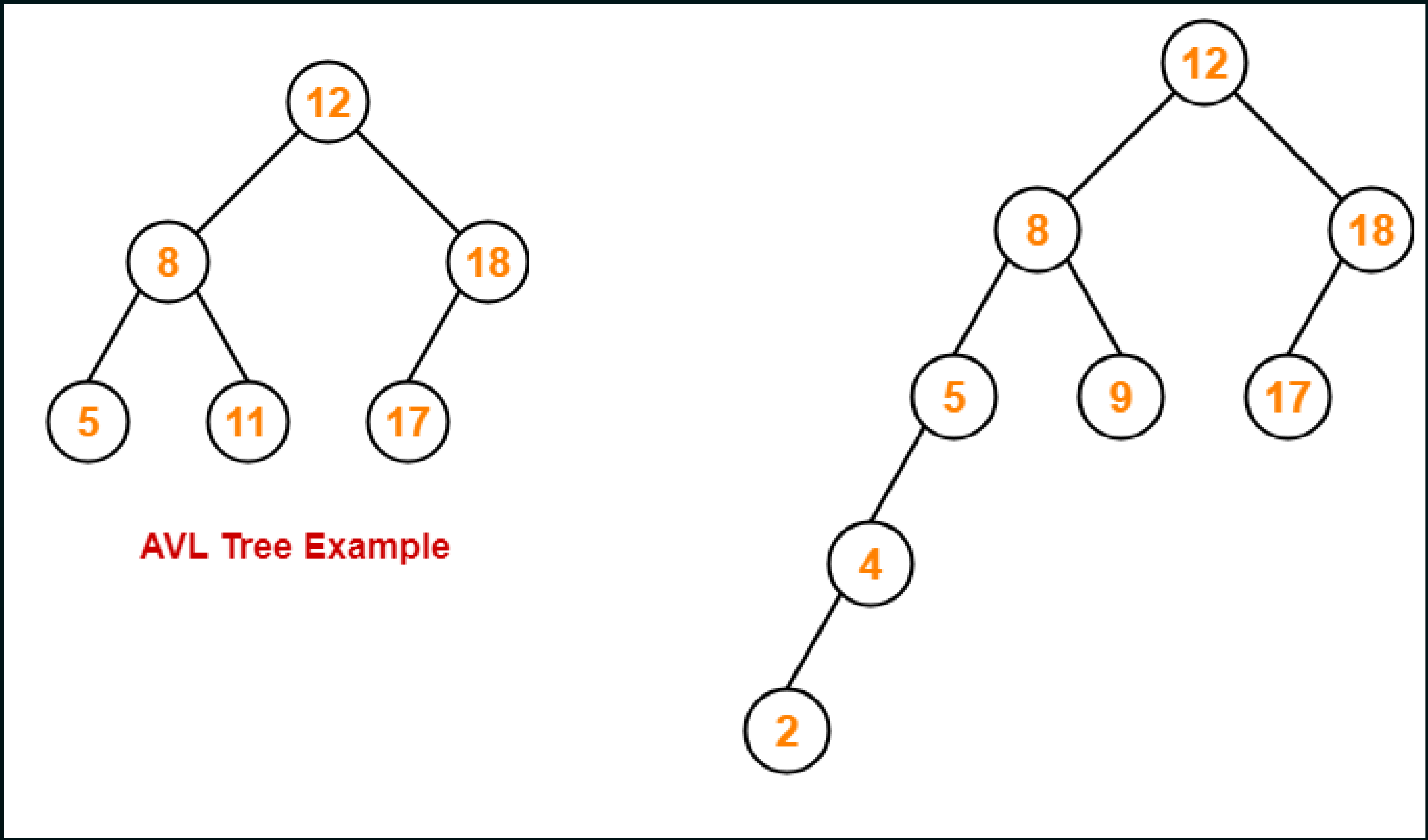
Cont…

- Example



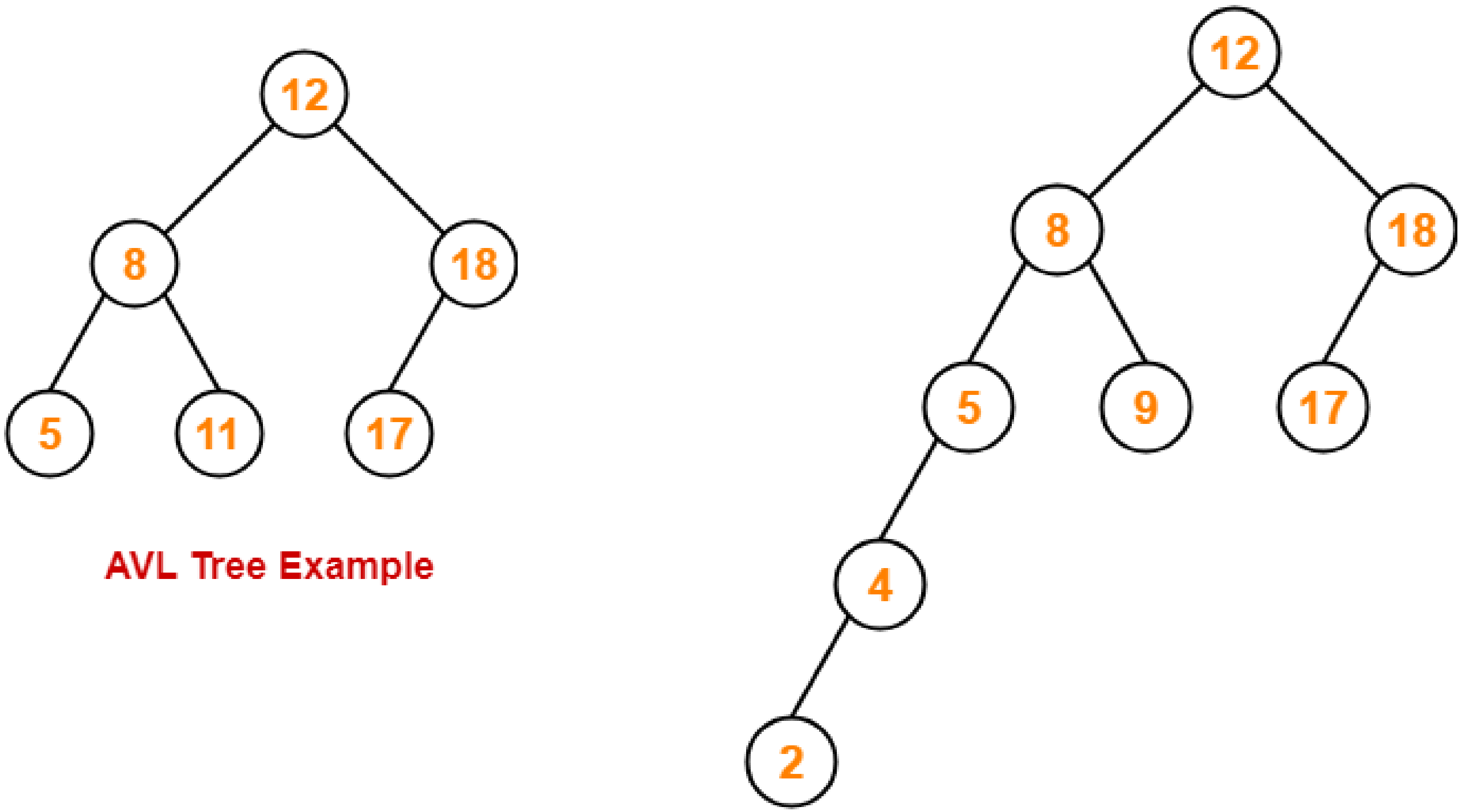
Cont…

- Example



Cont…

- Example



AVL Tree Example

Not an AVL Tree

Cont…

- Why AVL Trees?
  - AVL tree controls the height of the binary search tree by not letting it to be skewed
    - The time taken for all operations in a binary search tree of height h is O(h)
    - However, it can be extended to O(n) if the BST becomes skewed (i.e. worst case)
  - By limiting this height to log n, AVL tree imposes an upper bound on each operation to be O(log n) where n is the number of nodes

Cont…

- AVL Tree Properties
  - Property-01: Max. no. possible number of nodes in AVL tree of height H = $2^{H+1} - 1$

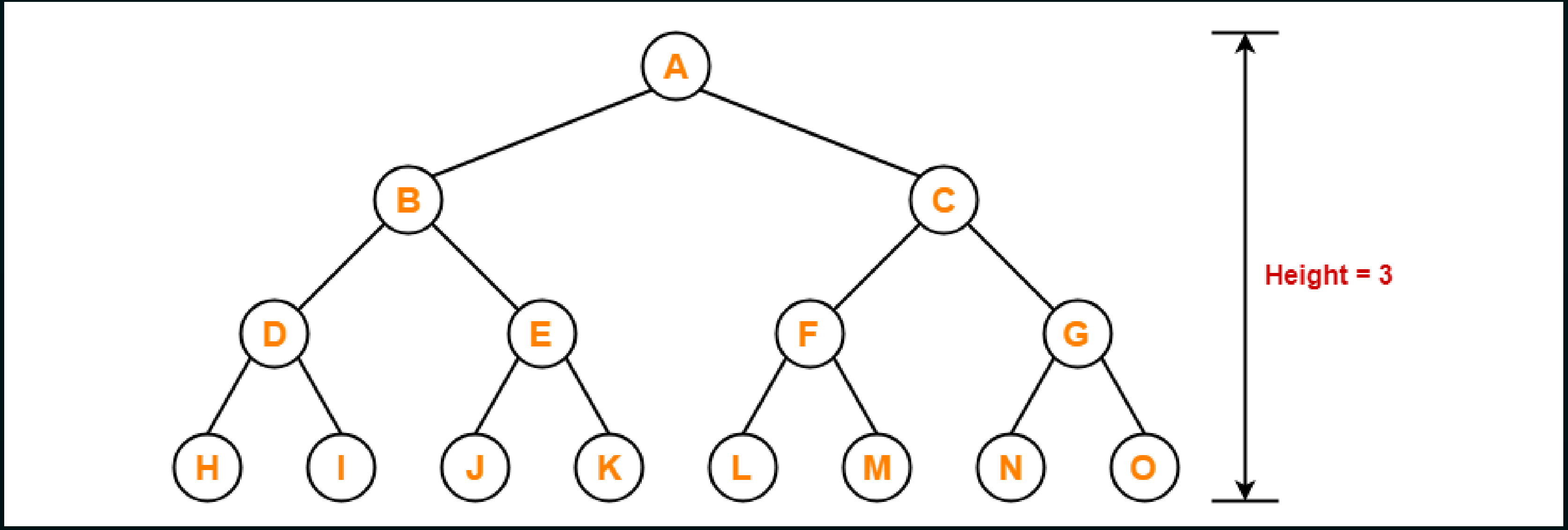
Cont…

- AVL Tree Properties
  - Property-01: Max. no. possible number of nodes in AVL tree of height H = $2^{H+1} – 1$
    - Example- Maximum possible number of nodes in AVL tree of height-3

Cont…

- AVL Tree Properties
  - Property-01: Max. no. number of nodes in AVL tree of height H = $2^{H+1} – 1$
    - Example- Max. possible number of nodes in AVL tree of height 3

      = $2^{3+1} – 1$ => 16 – 1 => 15

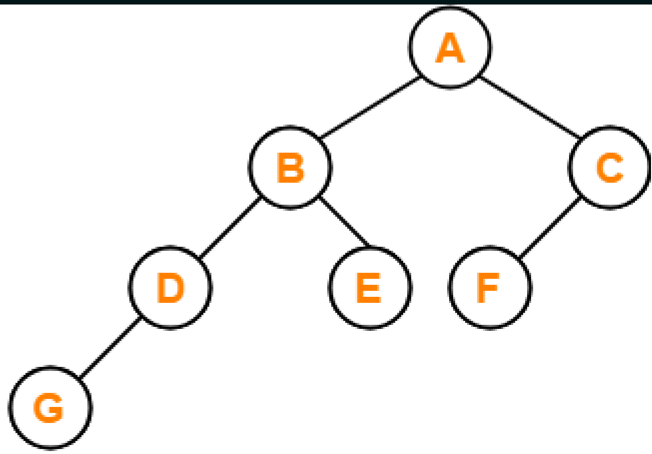


Cont…

- AVL Tree Properties
  - Property-02: Min. no. of nodes in AVL Tree of height H is given by a recursive relation

$$N(H) = N(H-1) + N(H-2) + 1$$

Given that $N(0)=1$, $N(1)=2$



AVL Tree

(Height = 3)

Cont…

- AVL Tree Properties
  - Property-03: Min possible height of AVL Tree using N nodes = $\lfloor \log_2 N \rfloor$

Cont…

- AVL Tree Properties
  - Property-03: Min possible height of AVL Tree using N nodes = $\lfloor \log_2 N \rfloor$
    - Example: Min. possible height of AVL Tree using 8 nodes

Cont…

- AVL Tree Properties
  - Property-03: Min possible height of AVL Tree using N nodes = $\lfloor \log_2 N \rfloor$
    - Example: Min. possible height of AVL Tree using 8 nodes

$$= \lfloor \log_2 8 \rfloor$$

$$= \lfloor \log_2 2^3 \rfloor$$

$$= \lfloor 3\log_2 2 \rfloor$$

$$= \lfloor 3 \rfloor$$

$$= 3$$

Cont…

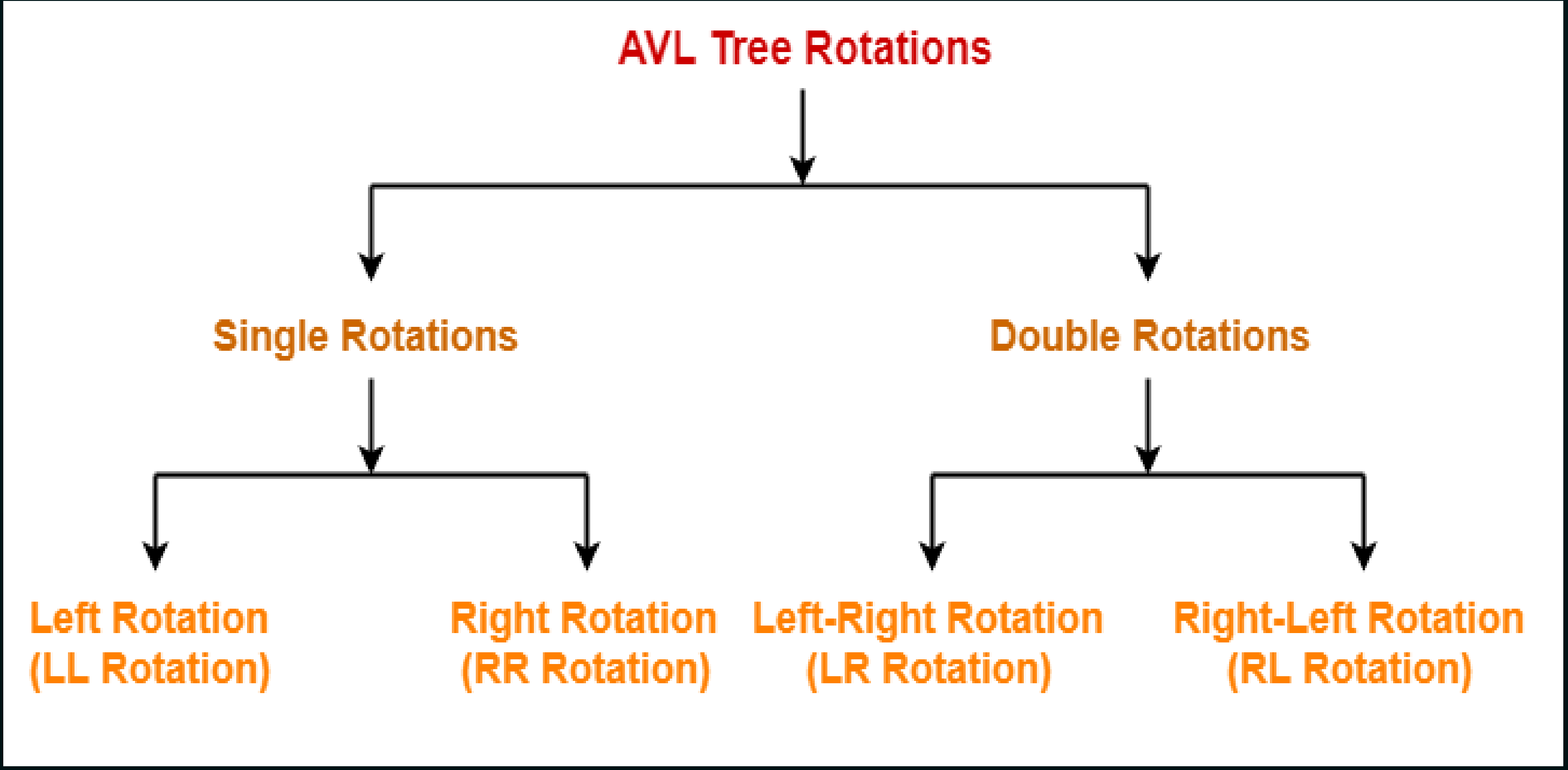
- Property-04: If there are N nodes in AVL Tree, its max. height can not exceed 1.44 $\log_2 N$
  - Worst case height of AVL Tree with N nodes = 1.44 $\log_2 N$
  - i.e. Worst case height of AVL Tree with n nodes = 1.44$\log_2 n$.

AVL Tree Operations

- Operations - Search, Insertion and Deletion
- After performing any operation on AVL tree, the balance factor of each node is checked. There might be two cases possible
  - Case-01:
    - After the operation, the balance factor of each node is either 0 or 1 or -1
    - In this case, the AVL tree is considered to be balanced
    - The operation is concluded.
  - Case-02:
    - After the operation, the balance factor of at least one node is not 0 or 1 or -1
    - In this case, the AVL tree is considered to be imbalanced.
    - Rotations are then performed to balance the tree.

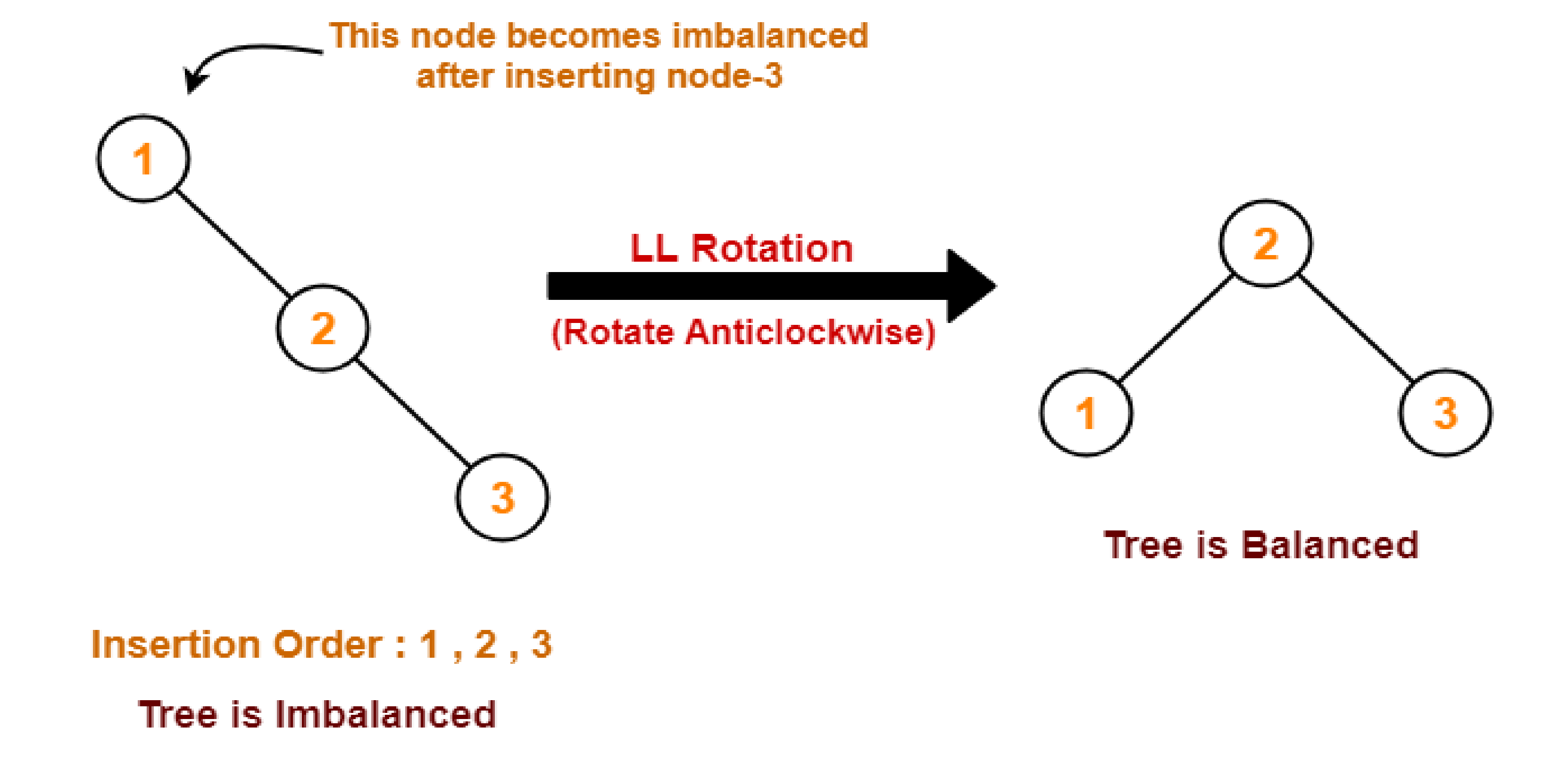
Cont…

- AVL Tree Rotations
  - 4 kinds of rotations possible in AVL Trees



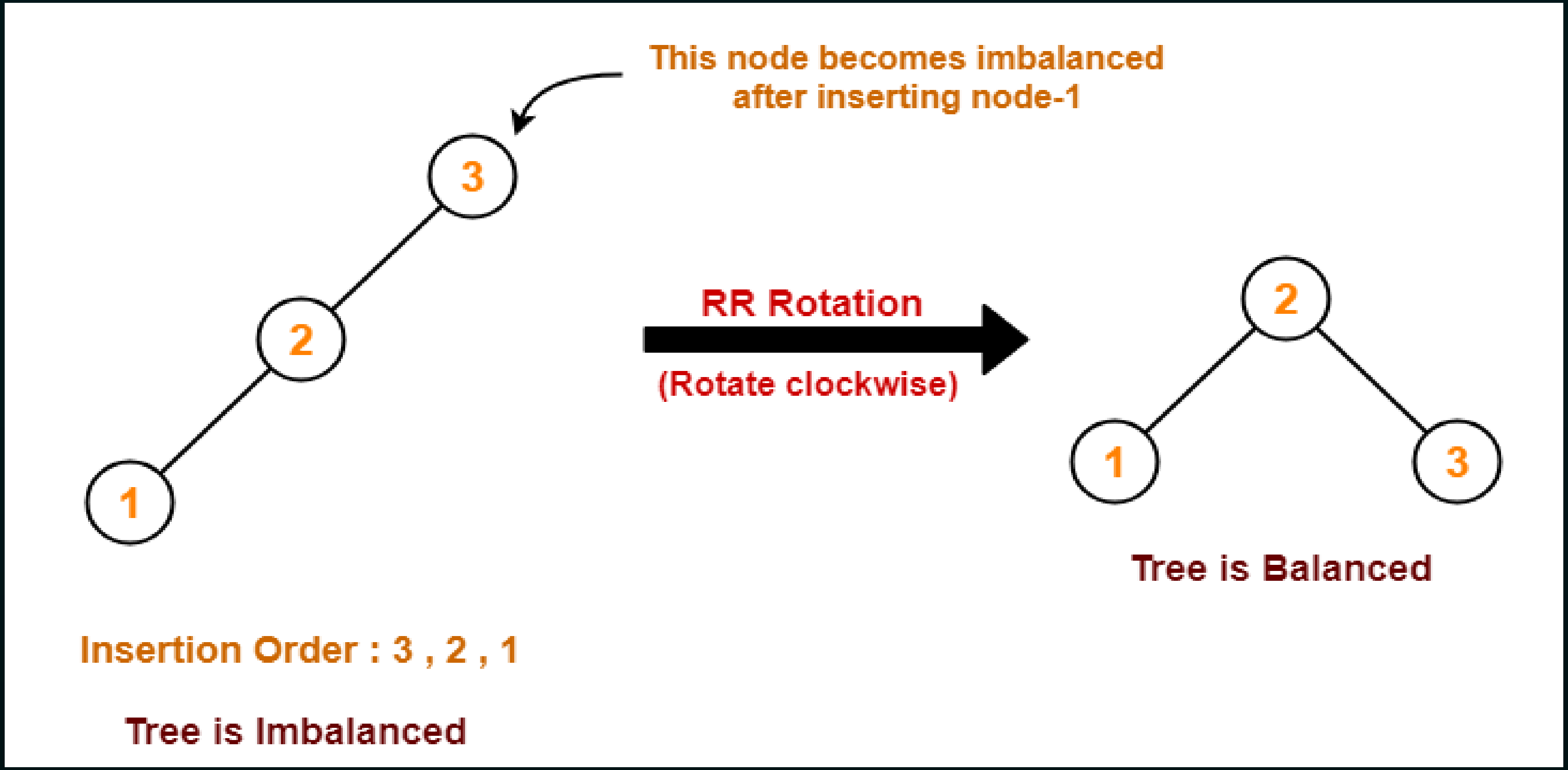
Cont…

- Cases Of Imbalance
  - Case-01:



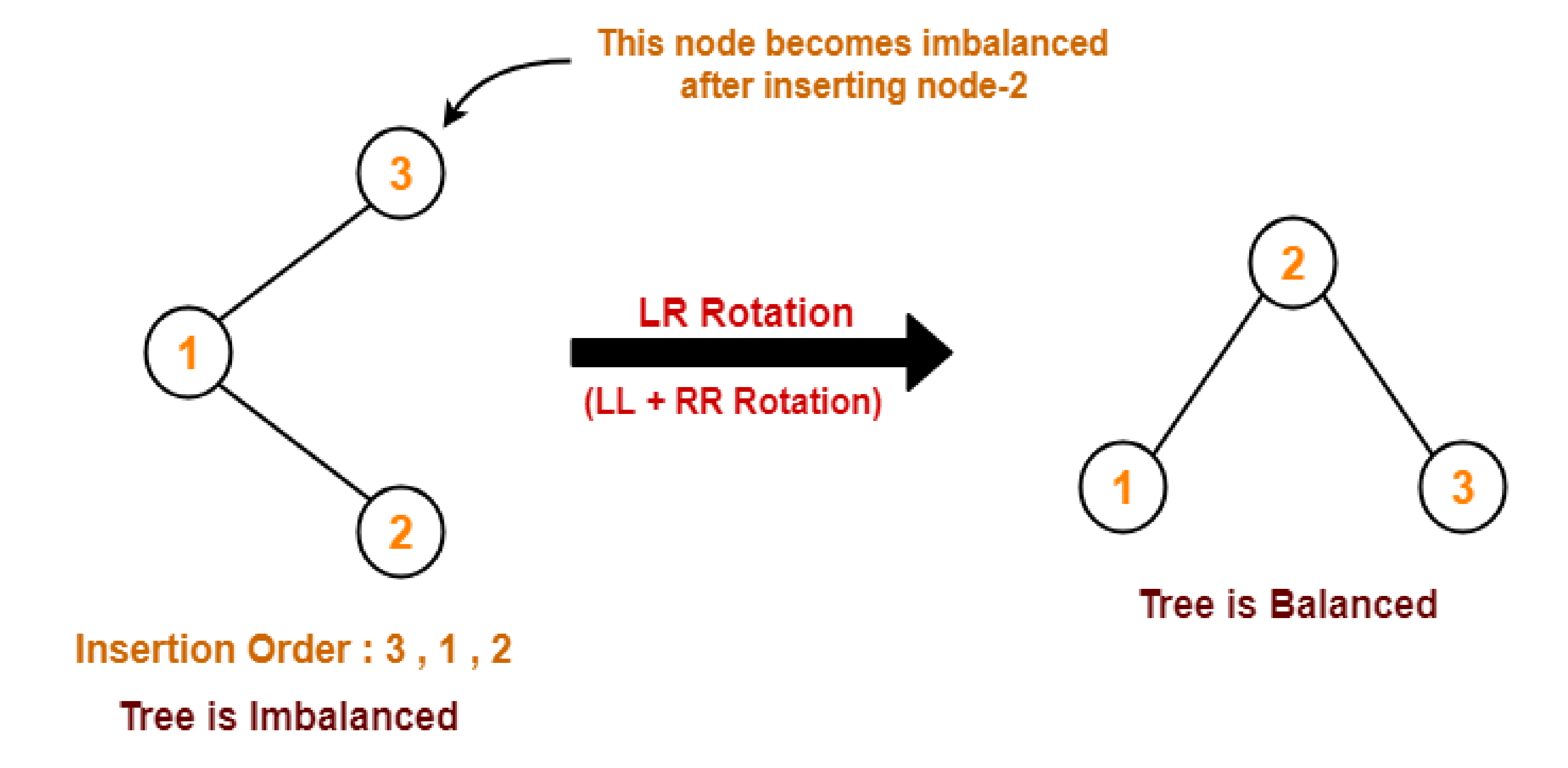
Cont…

- Cases Of Imbalance
  - Case-02:



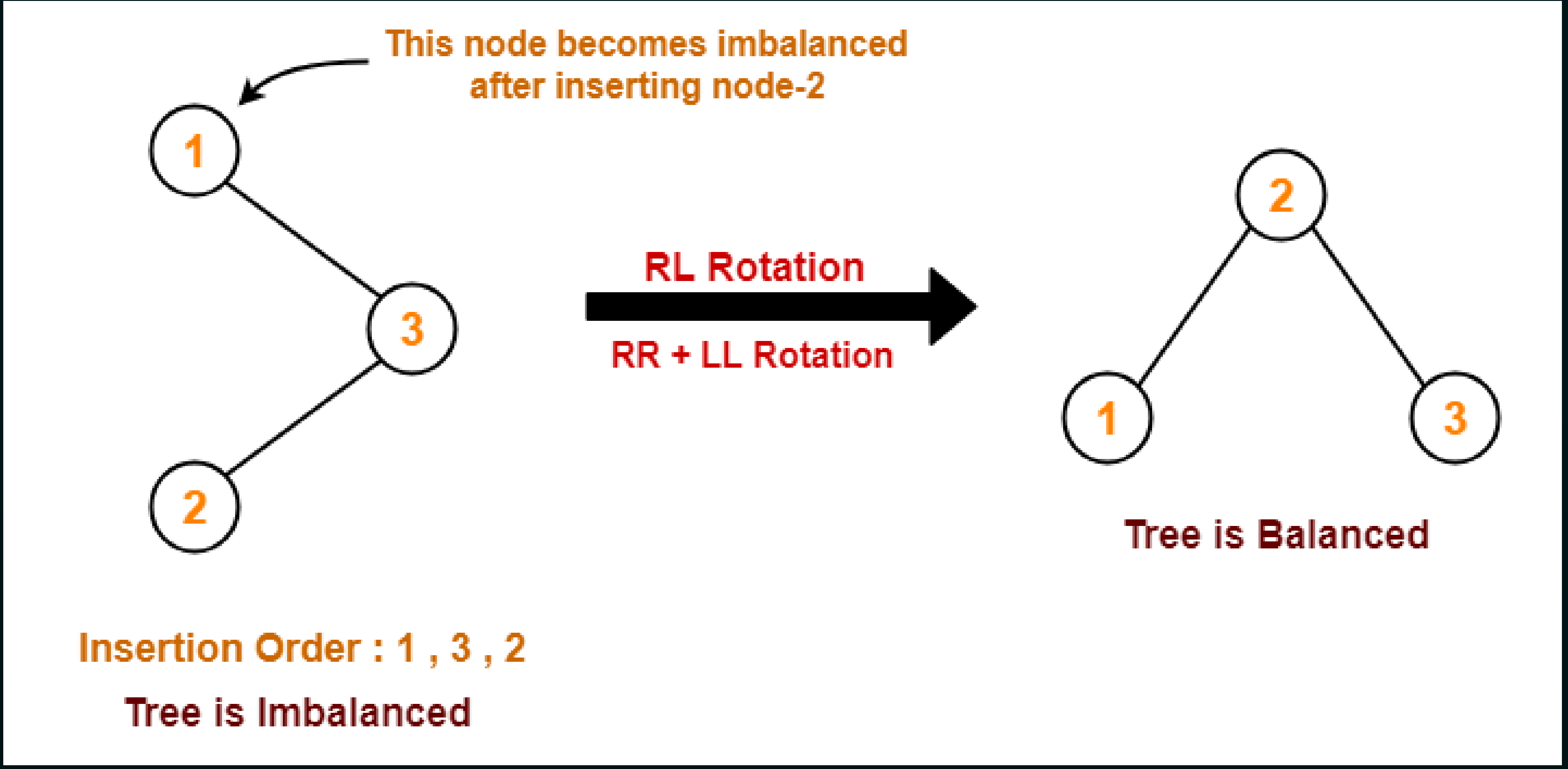
Cont…

- Cases Of Imbalance
  - Case-03:



Cont…

- Cases Of Imbalance
  - Case-04:



Insertion in AVL Tree

- To insert an element in the AVL tree, follow the following steps
  - Step 1: Insert the element in the AVL tree in the same way the insertion is performed in BST
  - Step 2: After insertion, check the balance factor of each node of the resulting tree
- Note-01:
  - Balance factor of only those nodes will be affected that lies on the path from the newly inserted node to the root node
- Note-02:
  - There is no need to check the balance factor of every node
  - Check the balance factor of only those nodes that lies on the path from the newly inserted node to the root node

Insertion in AVL Tree

- Note-03:
  - After inserting an element in the AVL tree,
    - If tree becomes imbalanced, then there exists one particular node in the tree by balancing which the entire tree becomes balanced automatically.
    - To re-balance the tree, balance that particular node
  - To find that particular node,
    - Check the balance factor of each node that is encountered while traversing the path from newly inserted node to the root node
    - The first encountered imbalanced node will be the node that needs to be balanced
  - To balance that node,
    - Count three nodes in the direction of leaf node and use the concept of AVL tree rotations to re balance the tree

Cont…

- Construct AVL Tree for the following sequence of numbers-

  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

Cont…

- Construct AVL Tree for the following sequence of numbers-

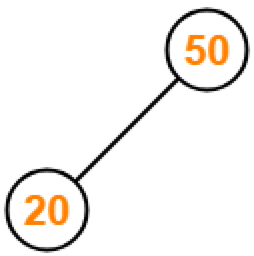
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

Insert 50

Cont…

- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

Insert 20

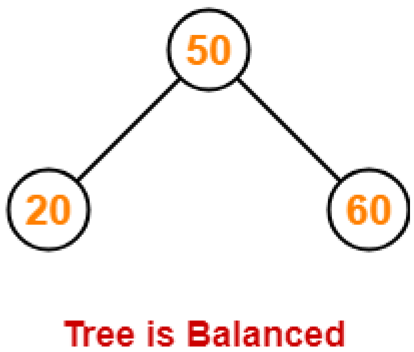


Tree is Balanced

Cont…

- Construct AVL Tree for the following sequence of numbers-
50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

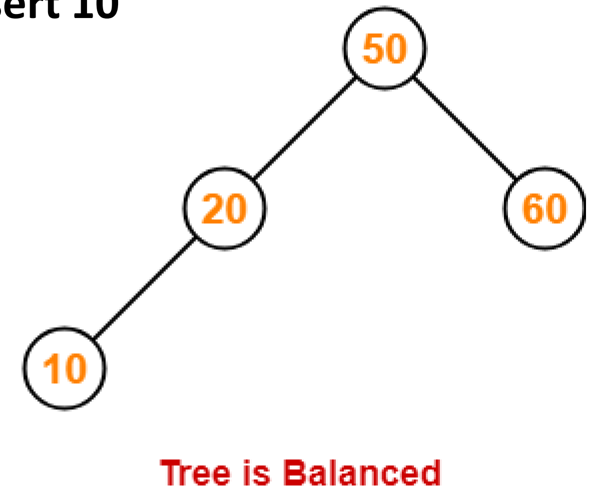
Insert 60



Cont…

- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

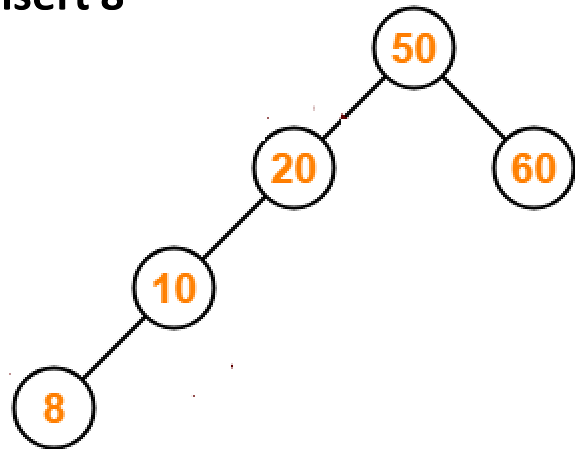
Insert 10



Cont…

- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

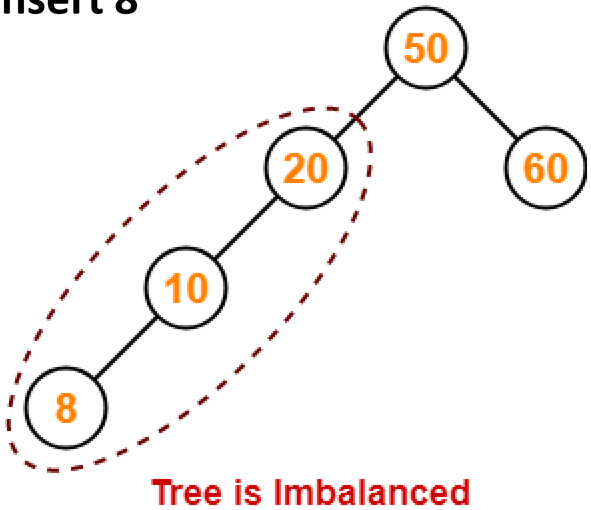
Insert 8



Cont…

- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

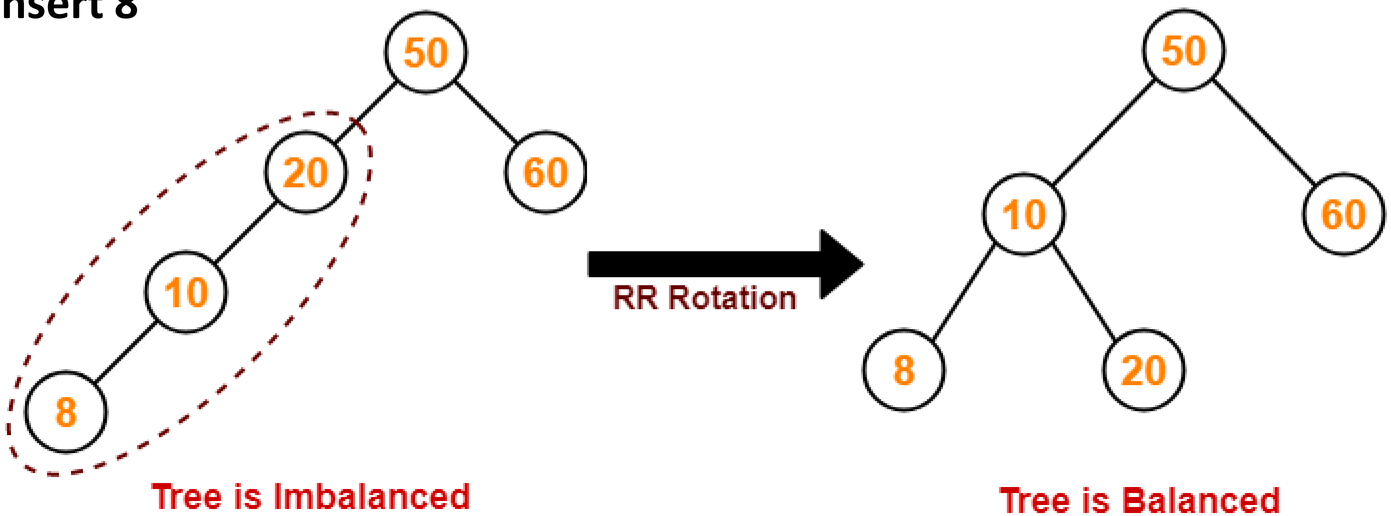
Insert 8



Cont…

- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

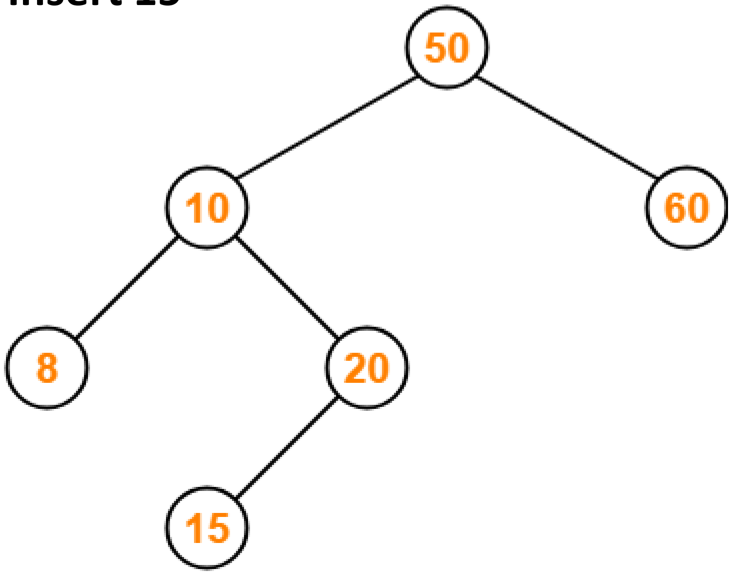
Insert 8



Cont…

- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

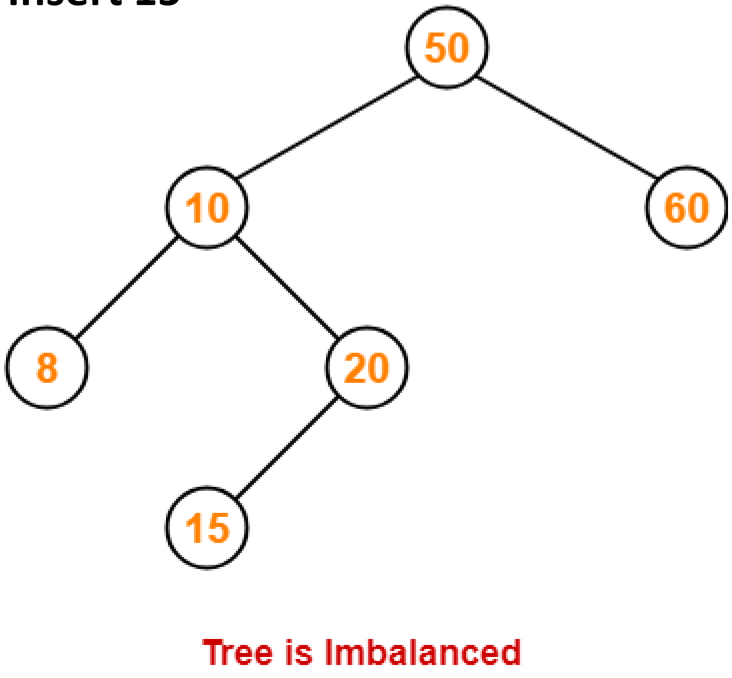
Insert 15



Cont…

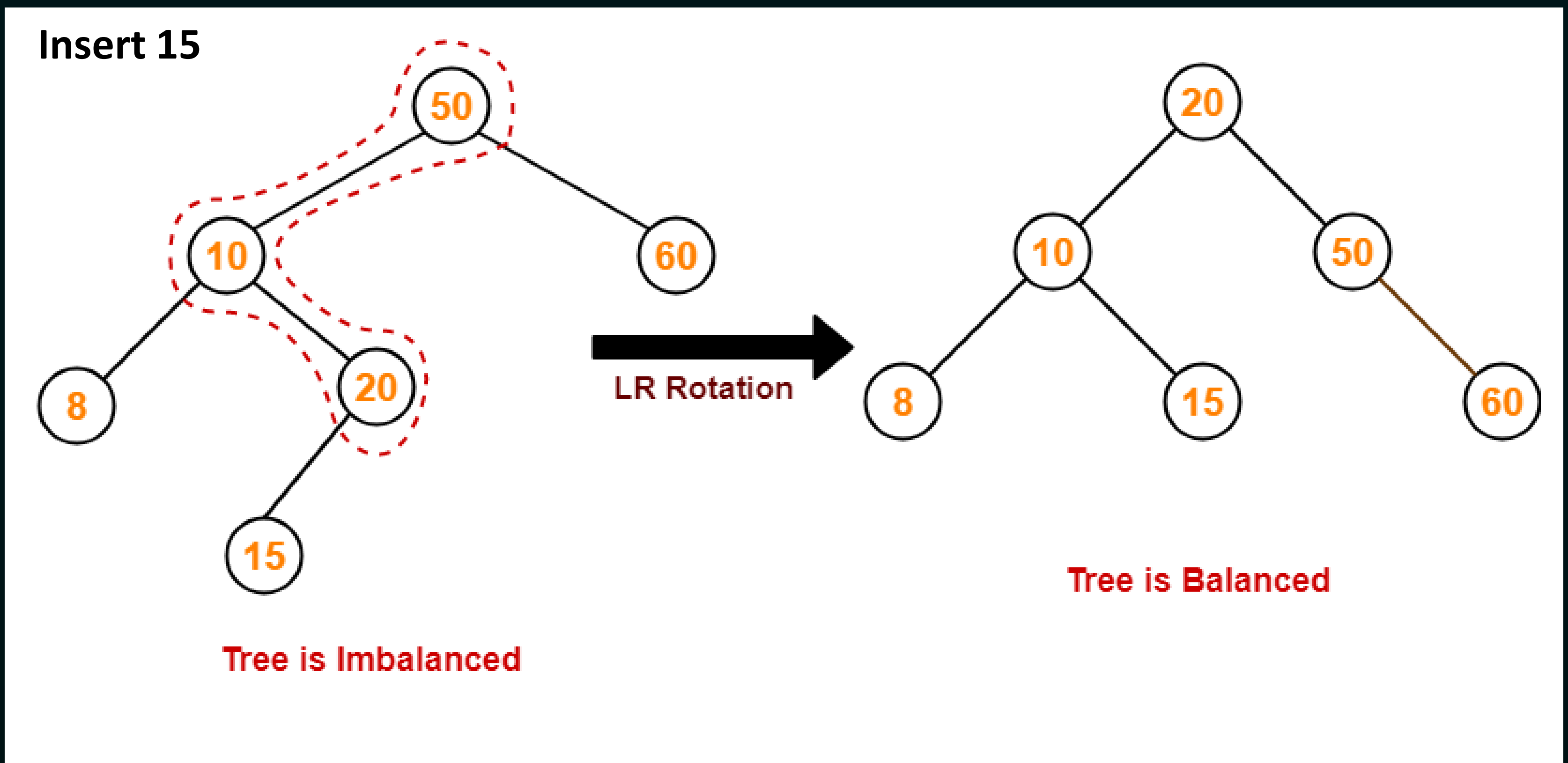
- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

Insert 15



Cont…

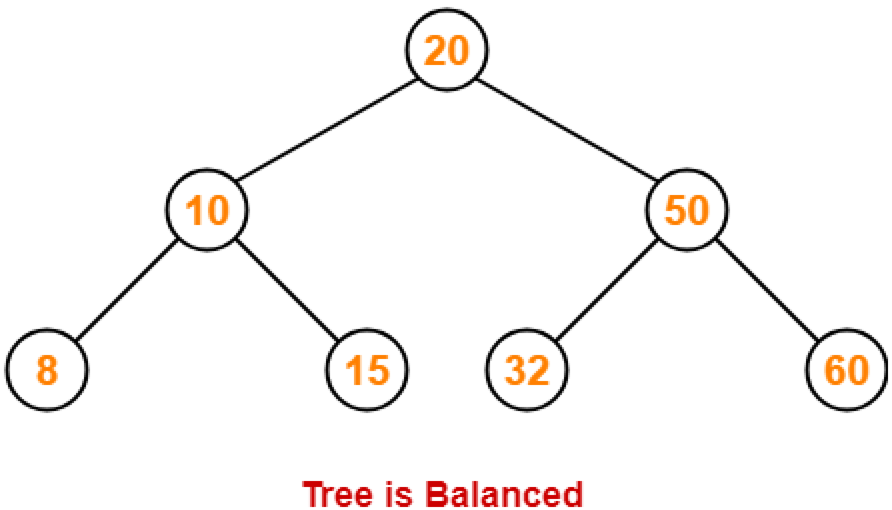
- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48



Cont…

- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

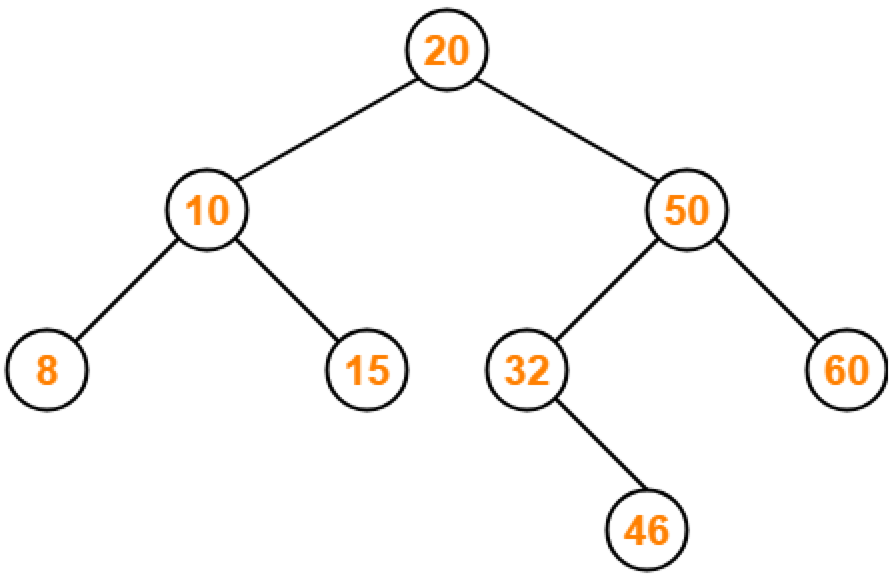
Insert 32



Cont…

- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

Insert 46

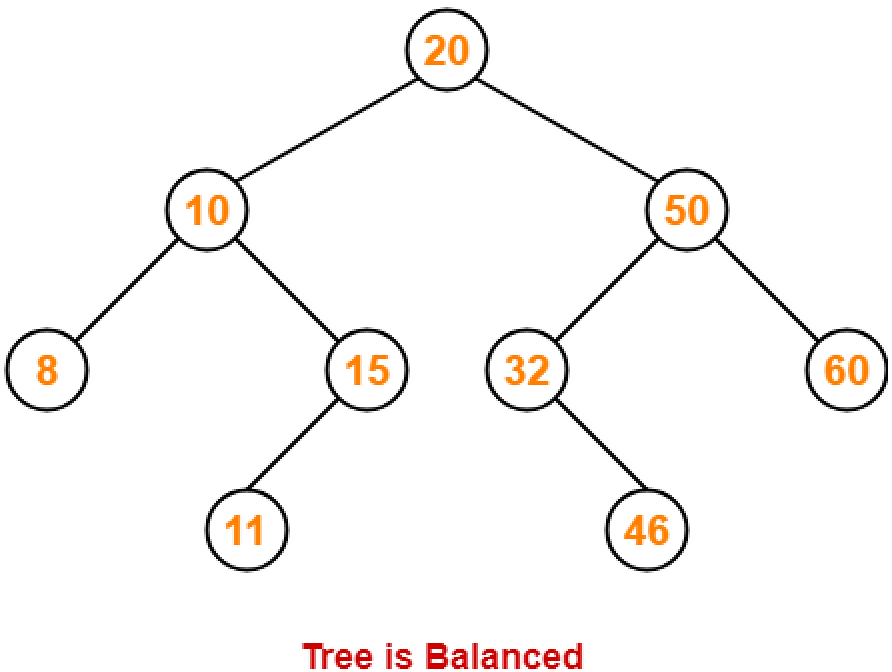


Tree is Balanced

Cont…

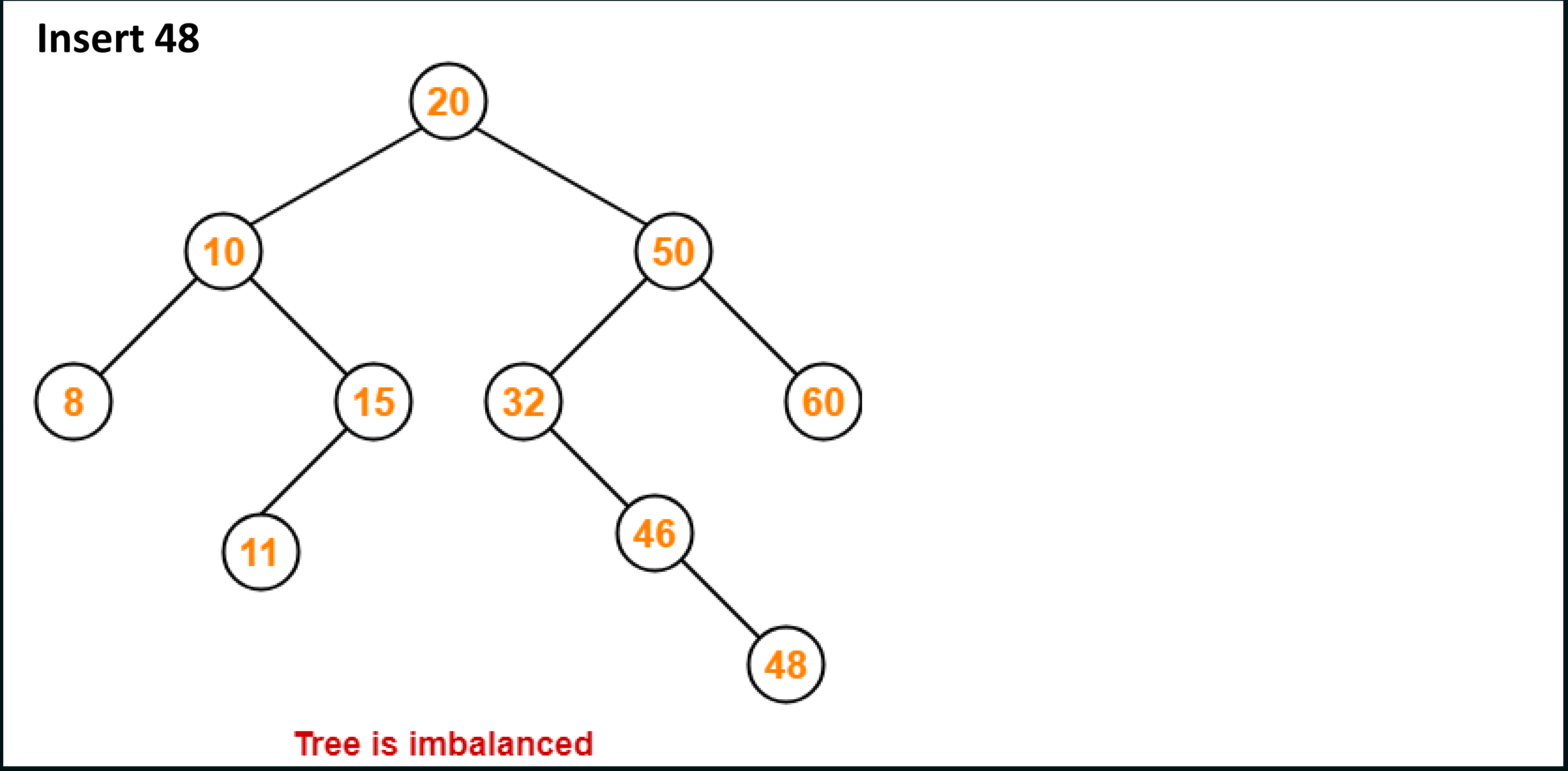
- Construct AVL Tree for the following sequence of numbers-
  50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48

Insert 11



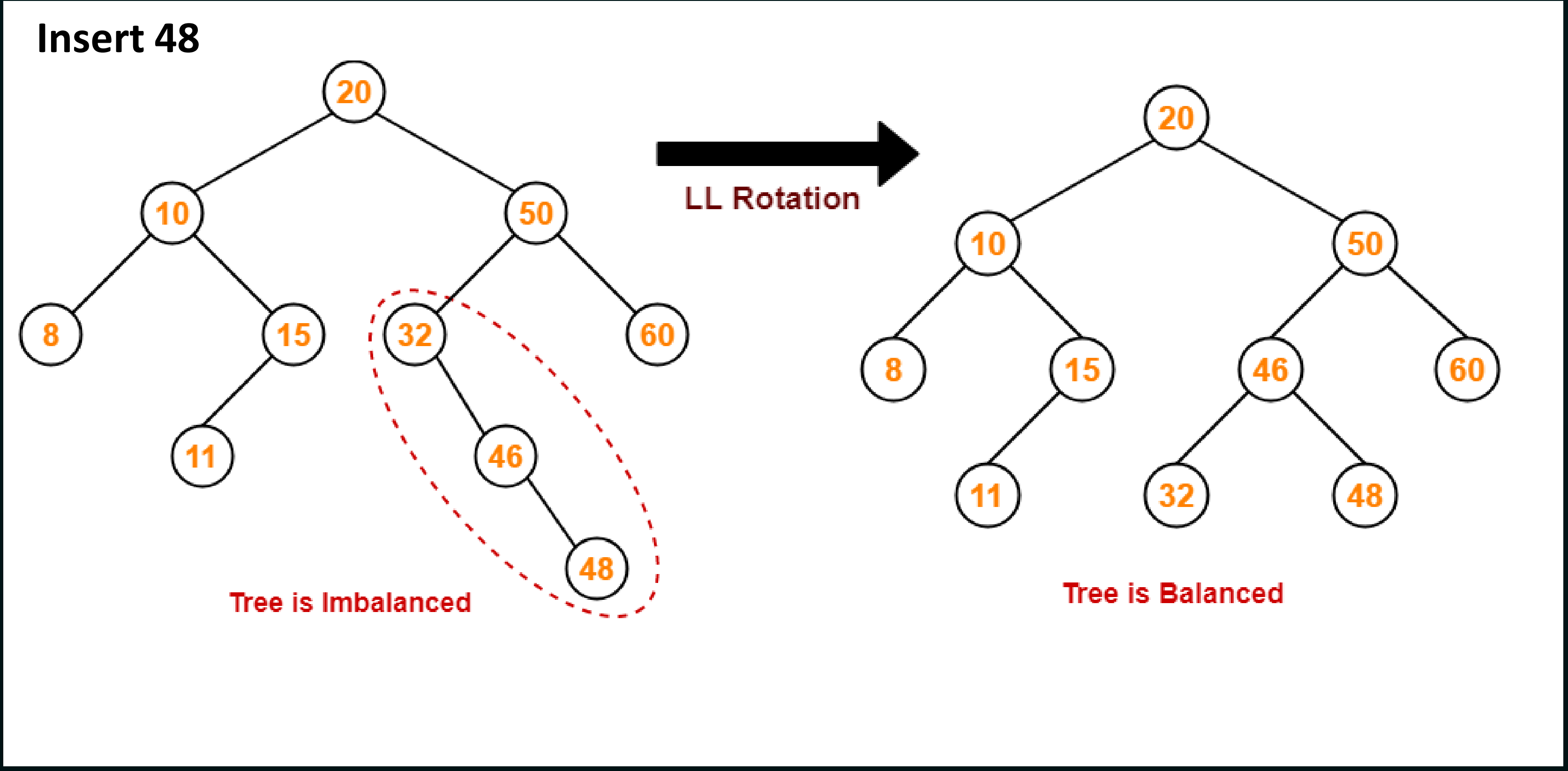
Cont…

- Construct AVL Tree for the following sequence of numbers-
50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48



Cont…

- Construct AVL Tree for the following sequence of numbers- 50 , 20 , 60 , 10 , 8 , 15 , 32 , 46 , 11 , 48



Cont…

- Construct an AVL tree having the following elements

  H, I, J, B, A, E, C, F, D, G, K, L

Cont…

- Construct an AVL tree having the following elements

  H, I, J, B, A, E, C, F, D, G, K, L

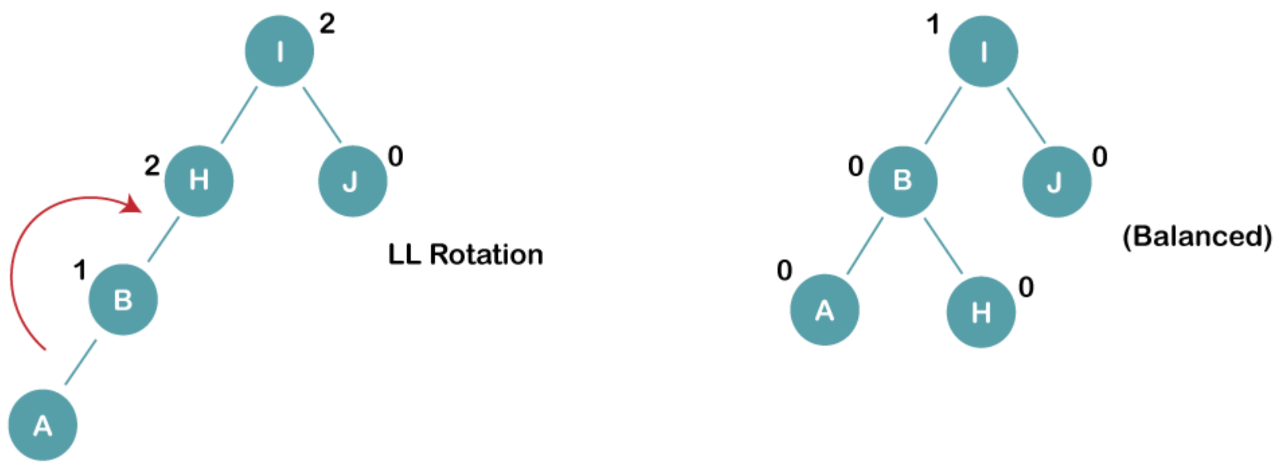
Insert H, I, J

Cont…

- Construct an AVL tree having the following elements

  H, I, J, B, A, E, C, F, D, G, K, L

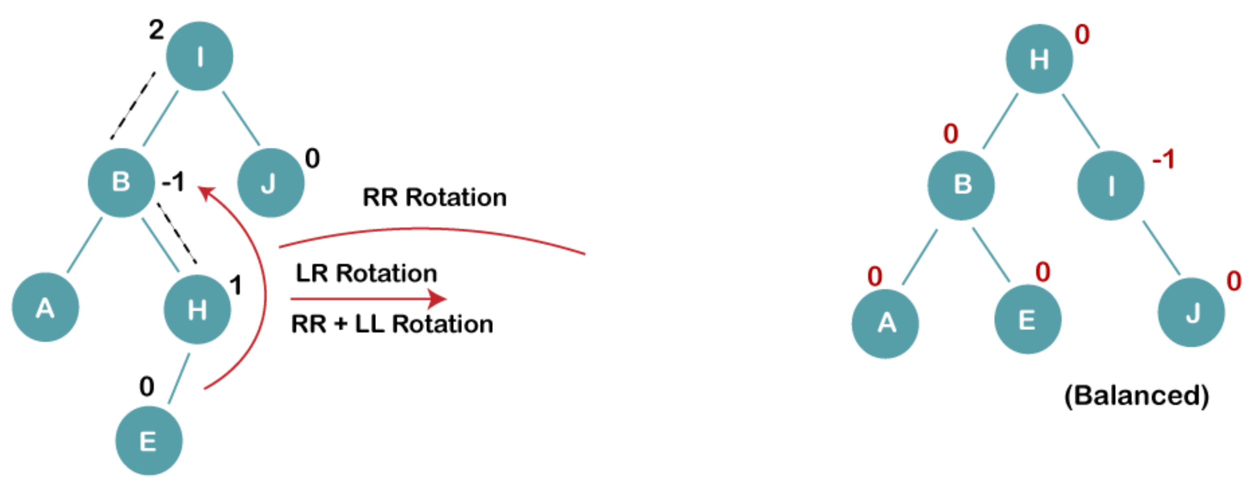
Insert B, A



Cont…

- Construct an AVL tree having the following elements
  H, I, J, B, A, E, C, F, D, G, K, L

Insert E

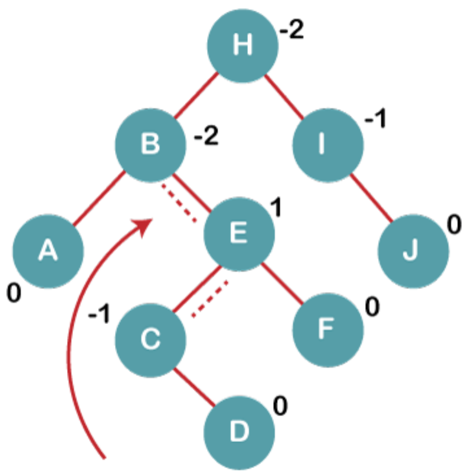


Cont…

- Construct an AVL tree having the following elements

  H, I, J, B, A, E, C, F, D, G, K, L

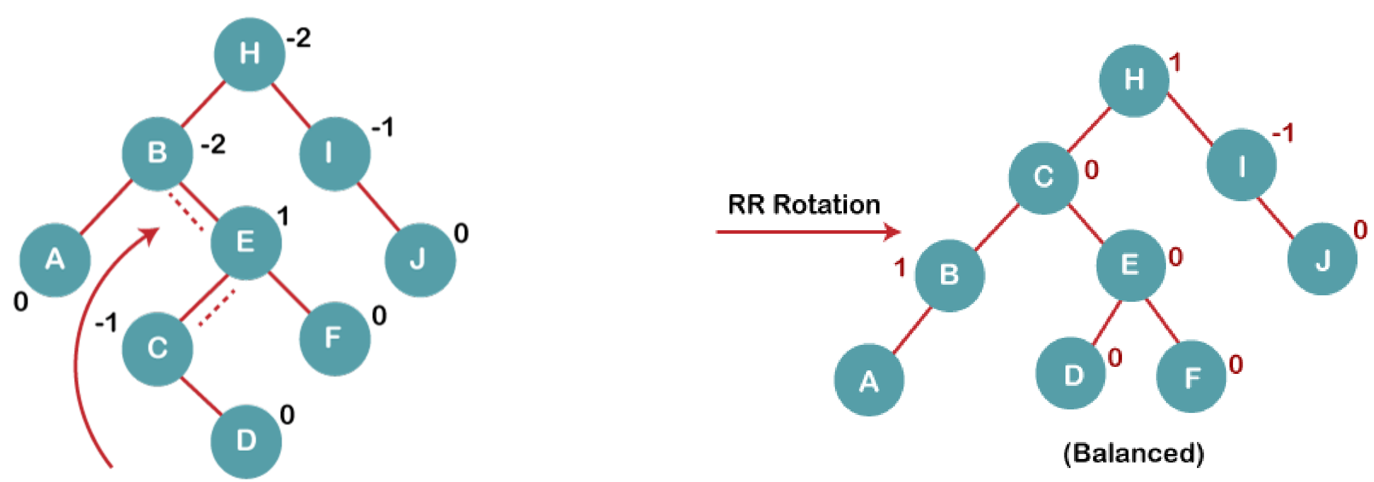
Insert C, F, D



Cont…

- Construct an AVL tree having the following elements
  H, I, J, B, A, E, C, F, D, G, K, L

Insert C, F, D

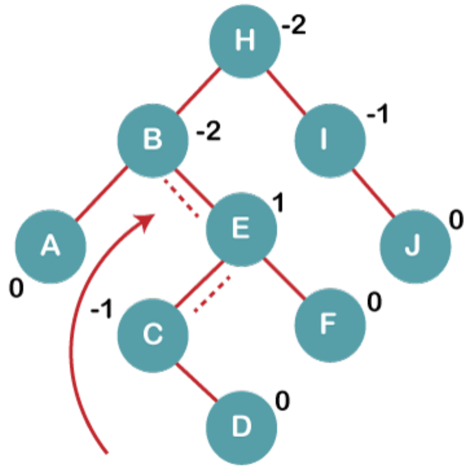


Cont…

- Construct an AVL tree having the following elements

  H, I, J, B, A, E, C, F, D, G, K, L

Insert C, F, D

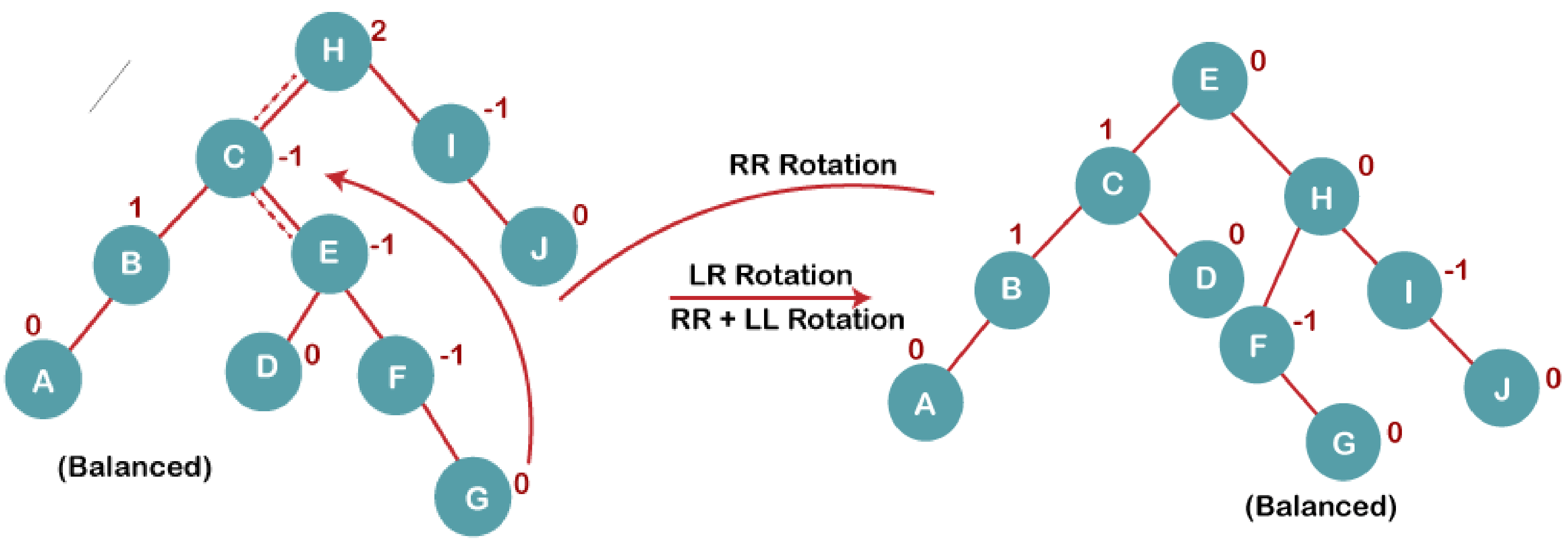


Cont…

- Construct an AVL tree having the following elements

  H, I, J, B, A, E, C, F, D, G, K, L

Insert G

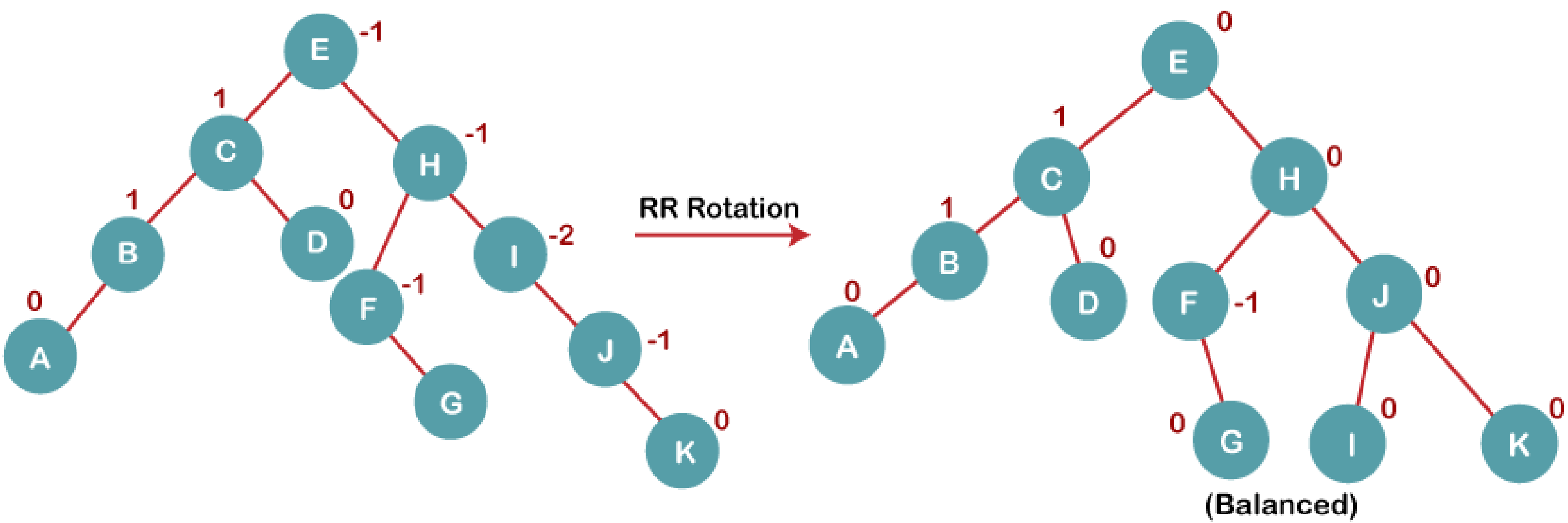


Cont…

- Construct an AVL tree having the following elements

  H, I, J, B, A, E, C, F, D, G, K, L

Insert K

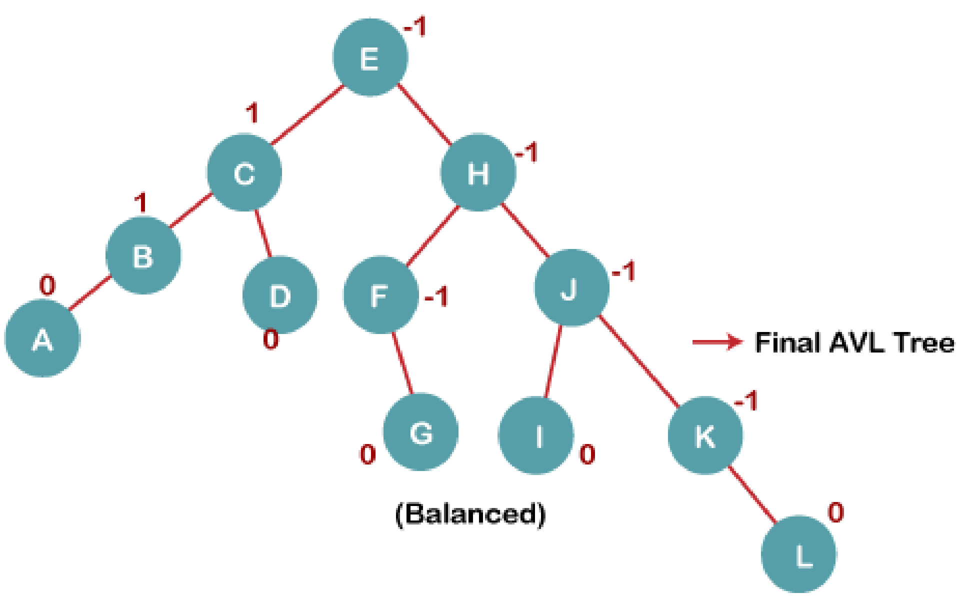


Cont…

- Construct an AVL tree having the following elements

  H, I, J, B, A, E, C, F, D, G, K, L

Insert L

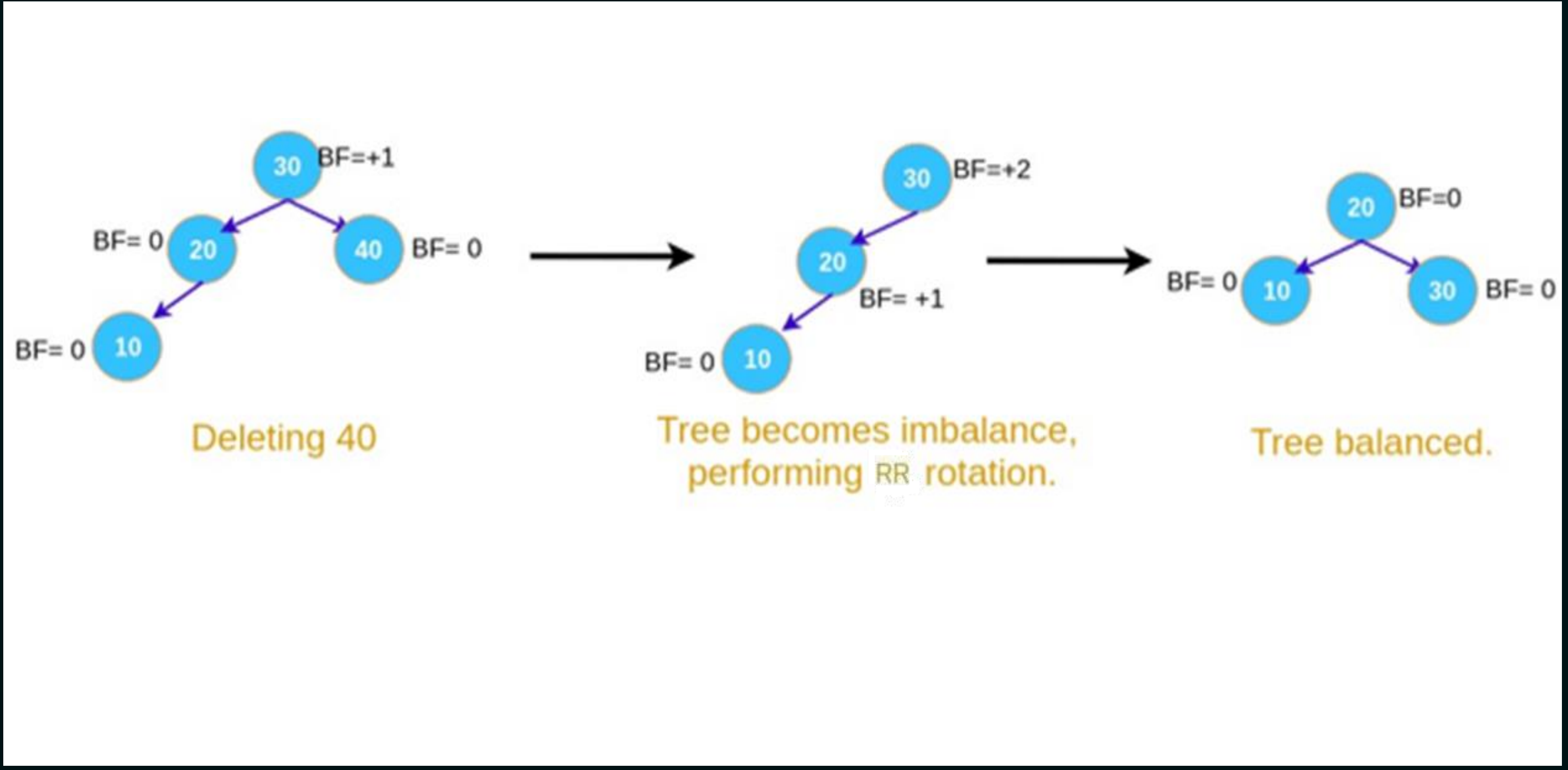


Deletion in AVL Tree

- Deletion in AVL Trees
  - Done using the same logic as in binary search tree
  - However, after deletion, restructure the tree, if needed, to maintain its balanced height
- Steps of Deletion
  - Step 1: Find the element in the tree.
  - Step 2: Delete the node, as per the BST Deletion.
  - Step 3: Two cases are possible:-
    - Case 1: Deleting from the right subtree.
    - Case 2: Deleting from left subtree.

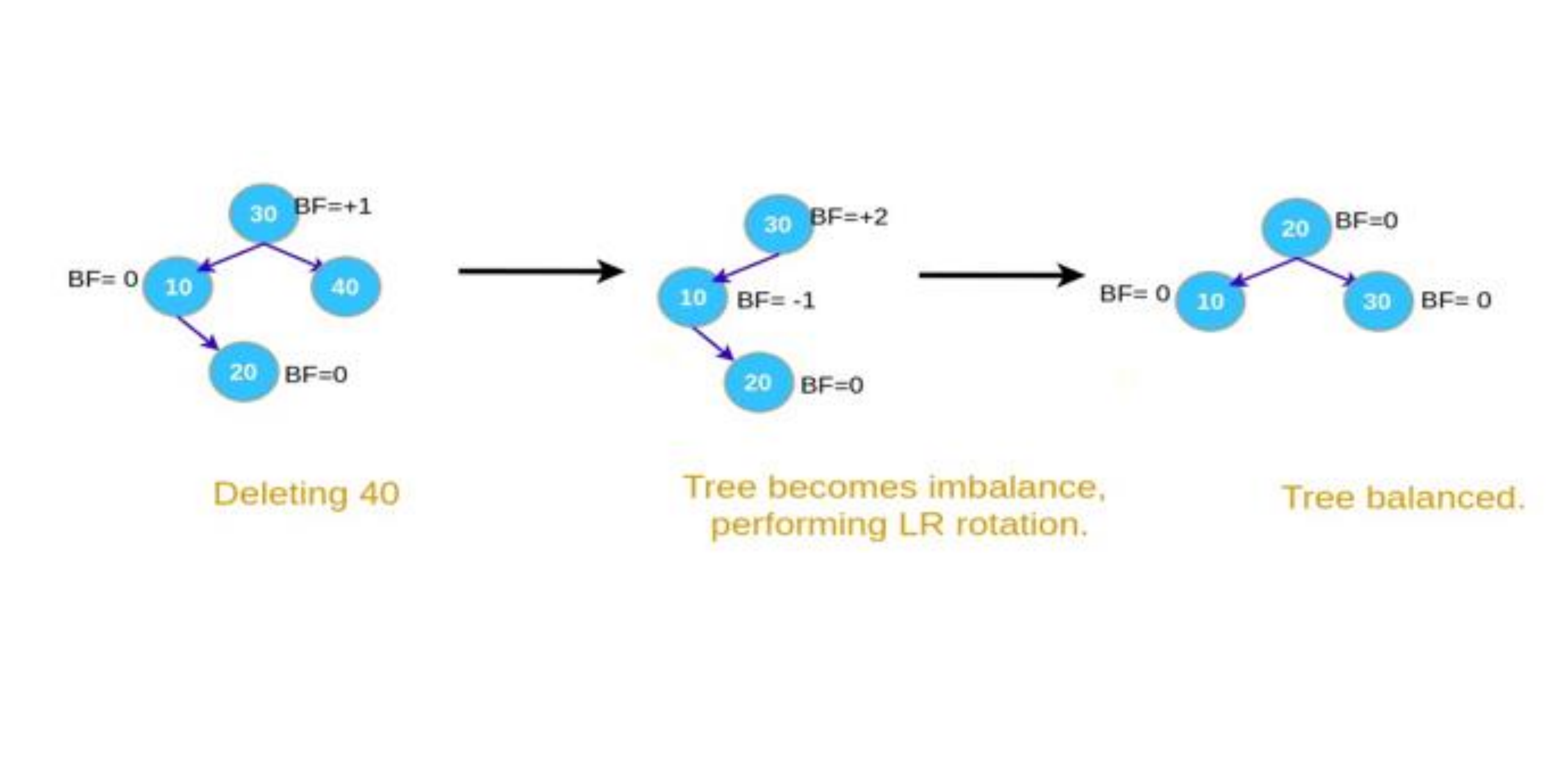
Cont…

- Case 1: Deleting from the right subtree
  - 1A. If BF(node)=+2 and BF(node->left-child) = +1, perform RR rotation



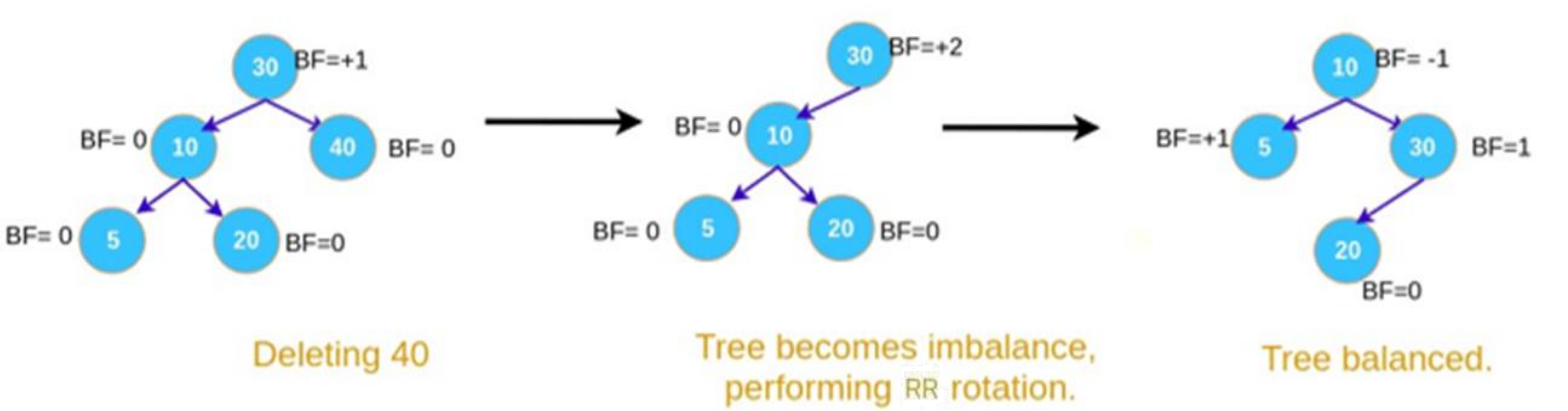
Cont…

- Case 1: Deleting from the right subtree
  - 1B. If BF(node)=+2 and BF(node->left-child)=-1, perform LR rotation.



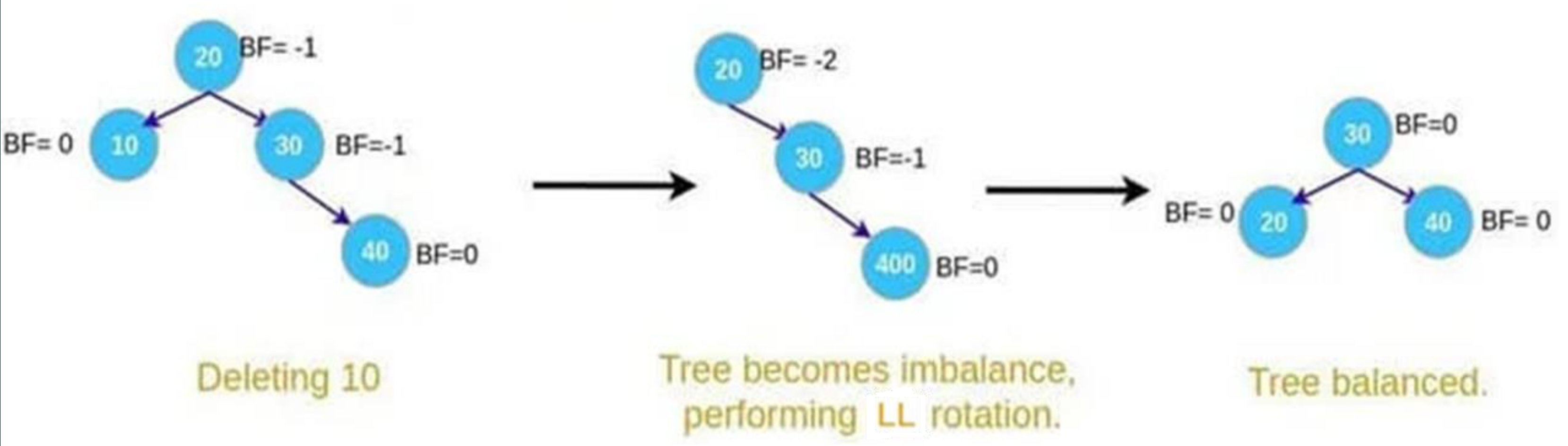
Cont…

- Case 1: Deleting from the right subtree
  - 1C. If BF(node)=+2 and BF(node->left-child)= 0, perform RR rotation.



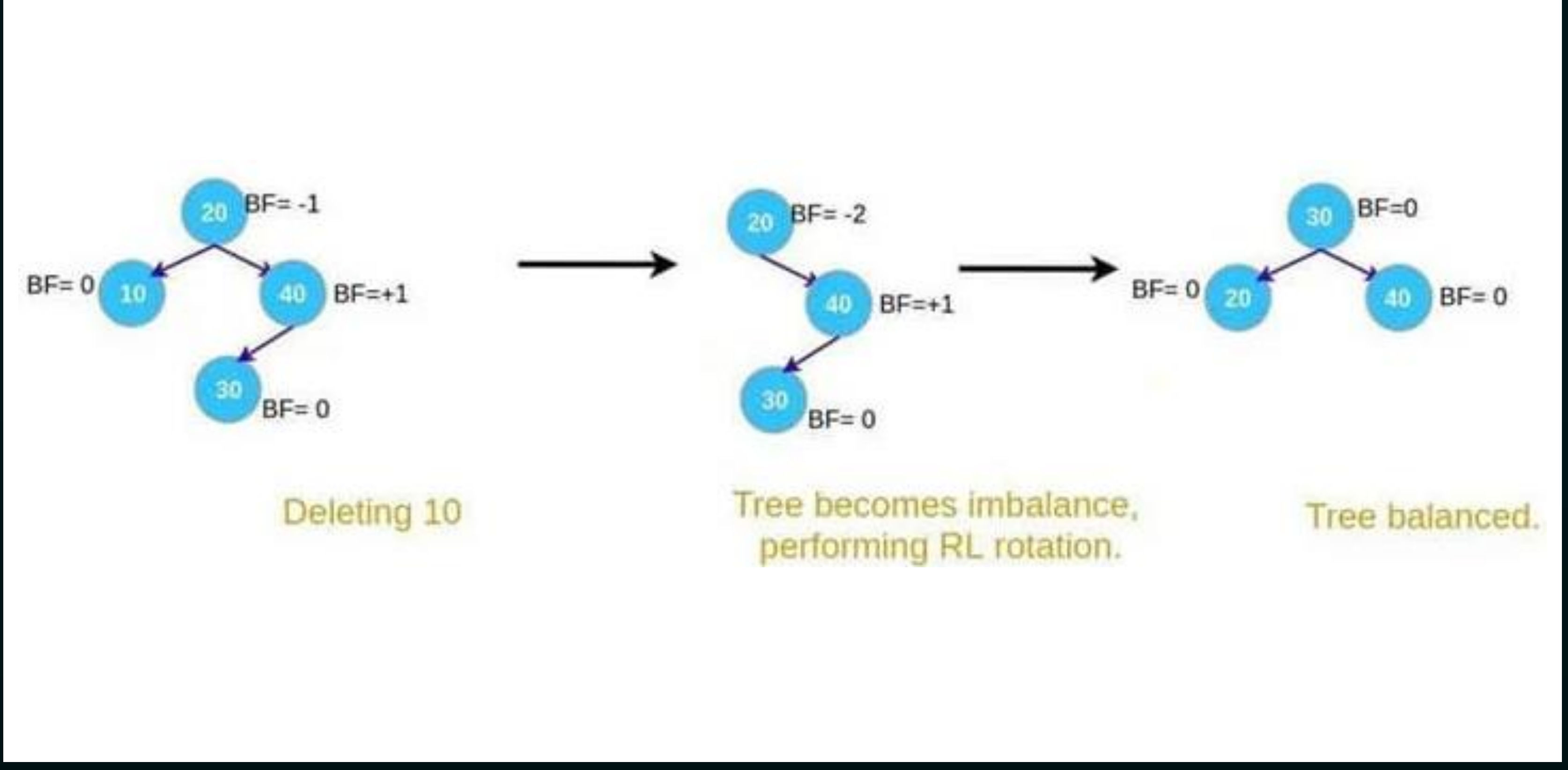
Cont…

- Case 2: Deleting from the left subtree
  - 2A. If BF(node)=-2 and BF(node->right-child) =-1, perform LL rotation.



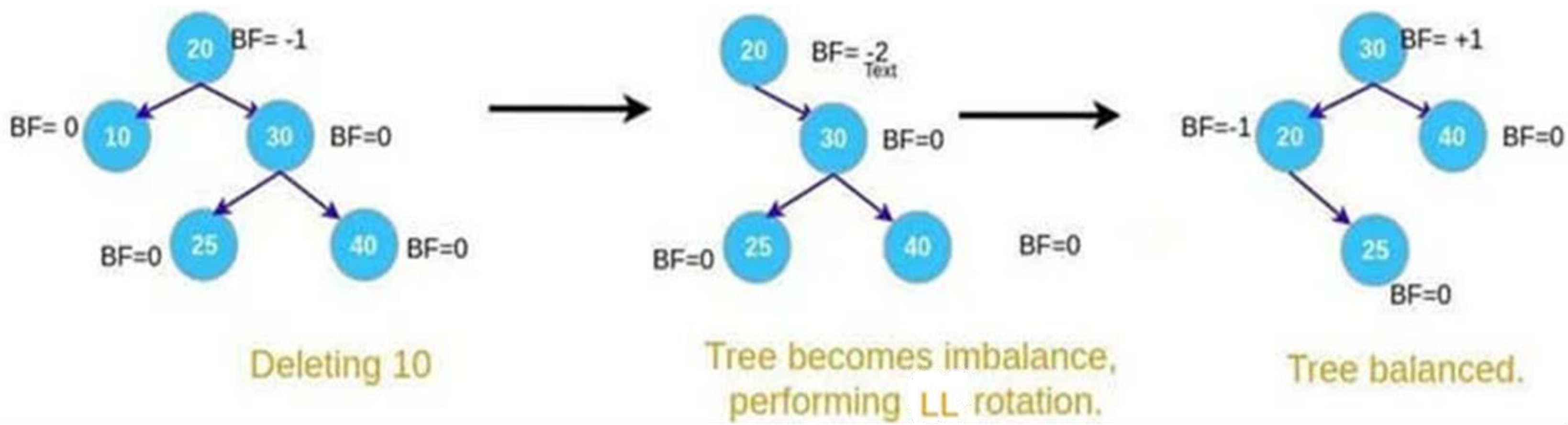
Cont…

- Case 2: Deleting from the left subtree
  - 2B. If BF(node)=-2 and BF(node->right-child)=+1, perform RL rotation.



Cont…

- Case 2: Deleting from the left subtree
  - 2C. If BF(node)=-2 and BF(node->right-child)=0, perform LL rotation.



Cont…

- Advantages of AVL Trees
  - The height of the AVL tree is always balanced. The height never grows beyond log N, where N is the total number of nodes in the tree.
  - It gives better search time complexity when compared to simple Binary Search trees.
  - AVL trees have self-balancing capabilities.

Cont…

- Summary
  - AVL trees are self-balancing binary search trees.
  - Balance factor is the fundamental attribute of AVL trees
    - The balance factor of a node is defined as the difference between the height of the left and right subtree of that node.
    - The valid values of the balance factor are -1, 0, and +1.
  - The insert and delete operation require rotations to be performed after violating the balance factor.
  - The time complexity of insert, delete, and search operation is O(log N).
  - AVL trees follow all properties of Binary Search Trees.
    - The left subtree has nodes that are lesser than the root node. The right subtree has nodes that are always greater than the root node.
  - AVL trees are used where search operation is more frequent compared to insert and delete operations.