CS-2001
DATA STRUCTURE

Dr. Hashim Yasin

National University of Computer and Emerging Sciences,

Faisalabad, Pakistan.

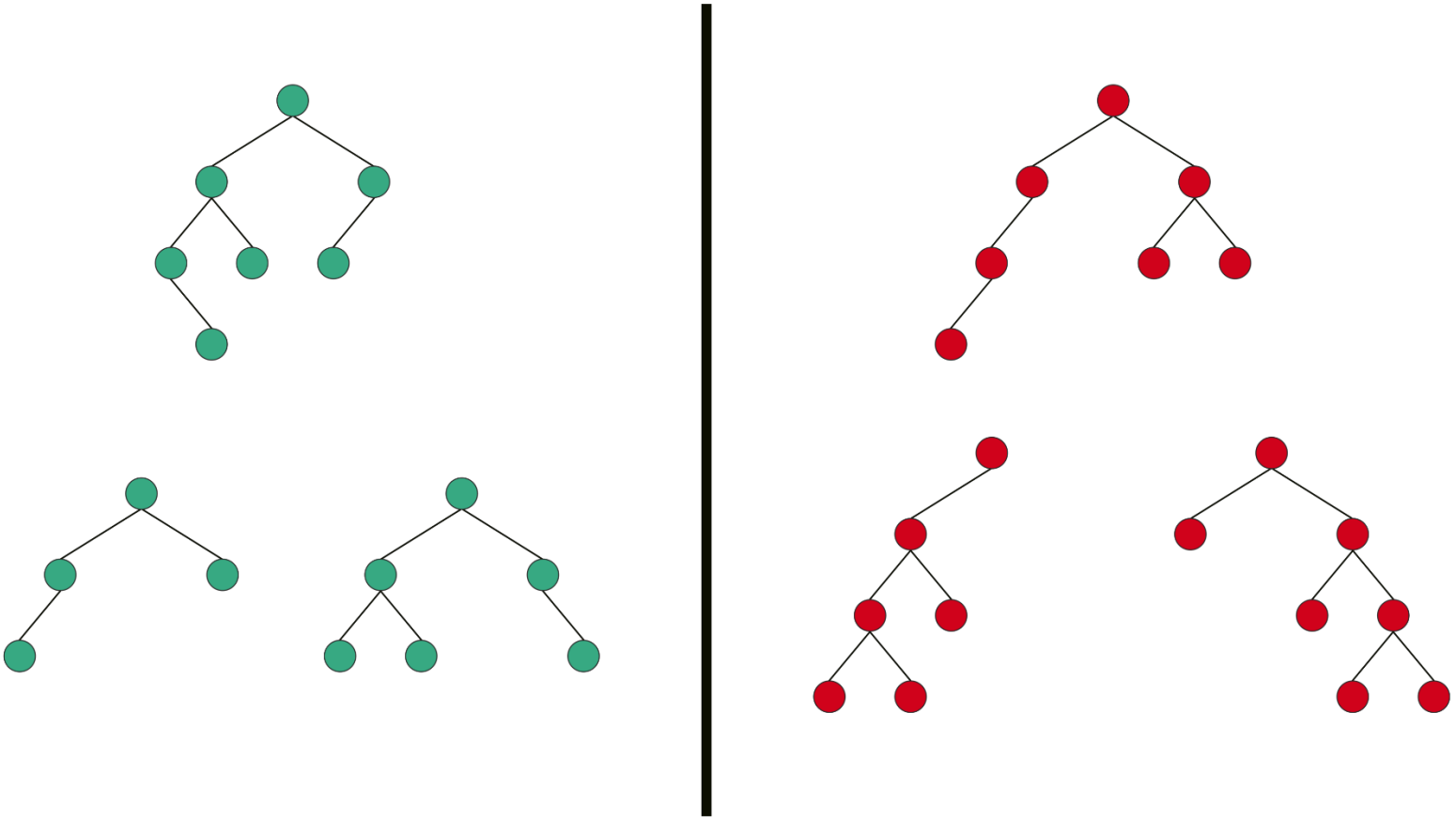
AVL TREE

Balanced Binary Trees

Balanced Binary Tree

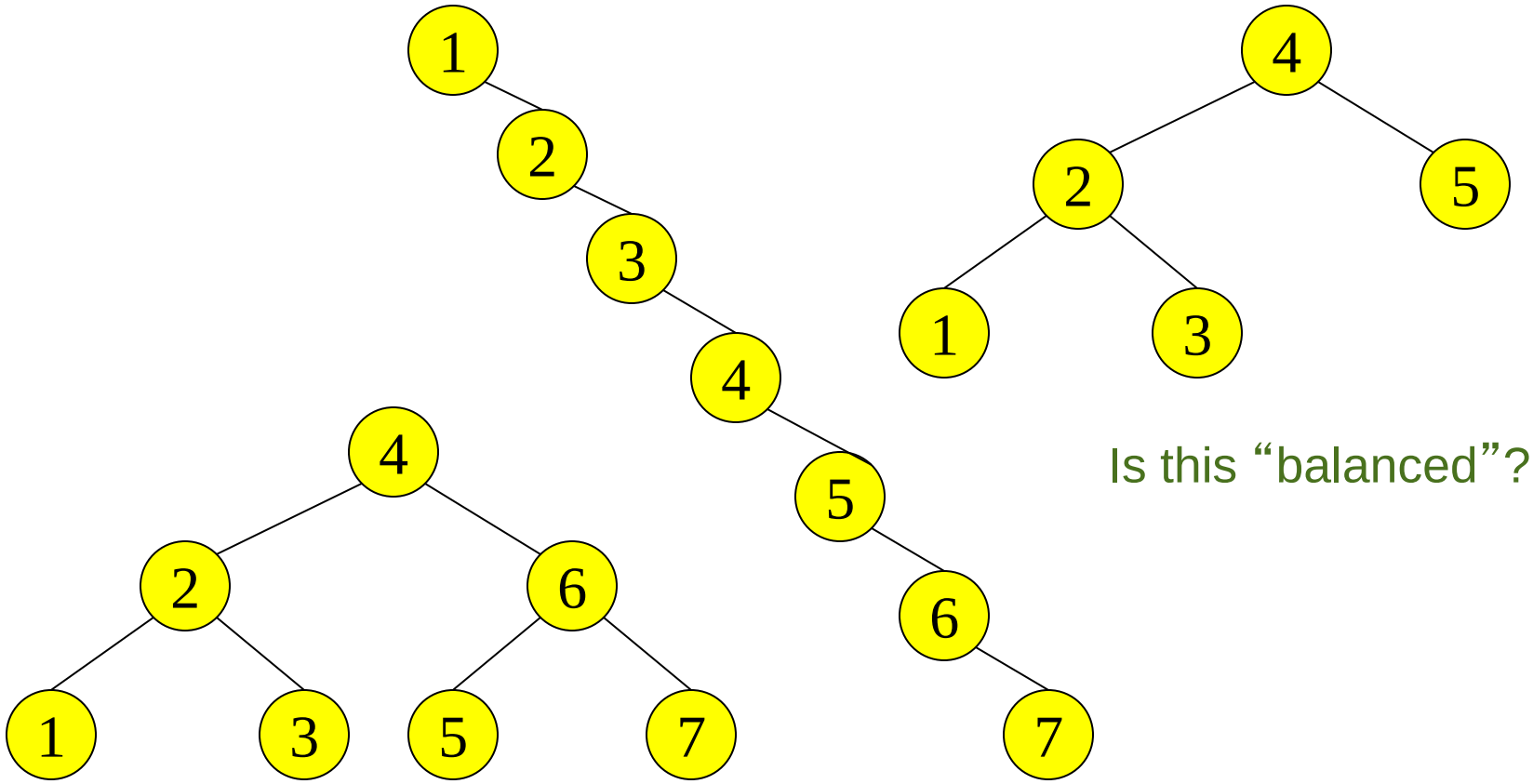
- is a Binary tree in which height of the left and the right sub-trees of every node may differ by at most 1.
  - For every node, heights of left and right subtree can differ by no more than 1

Balanced Binary Trees



Valid and Invalid Structure of Balanced Binary Tree

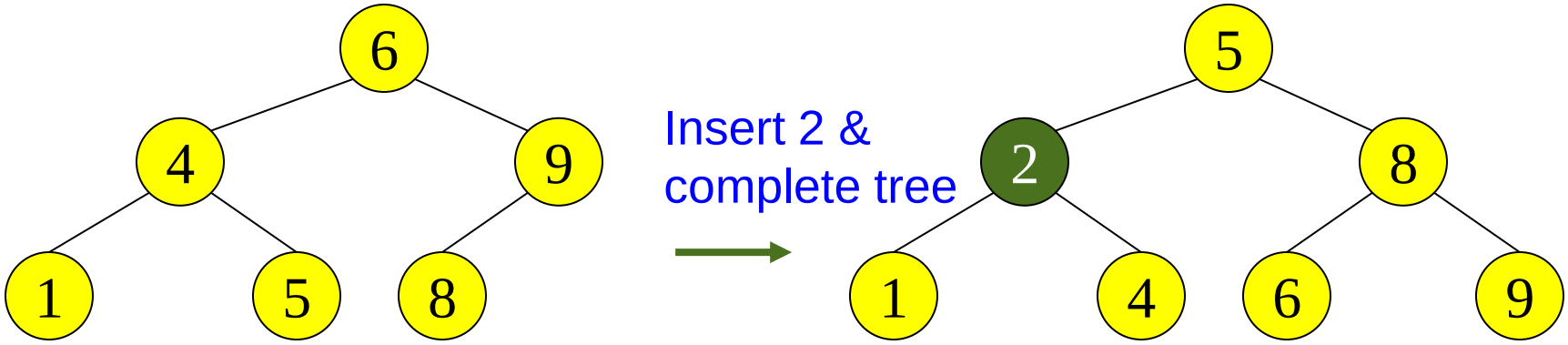
Balanced and Unbalanced BST



Is this “balanced”?

Perfect Balance

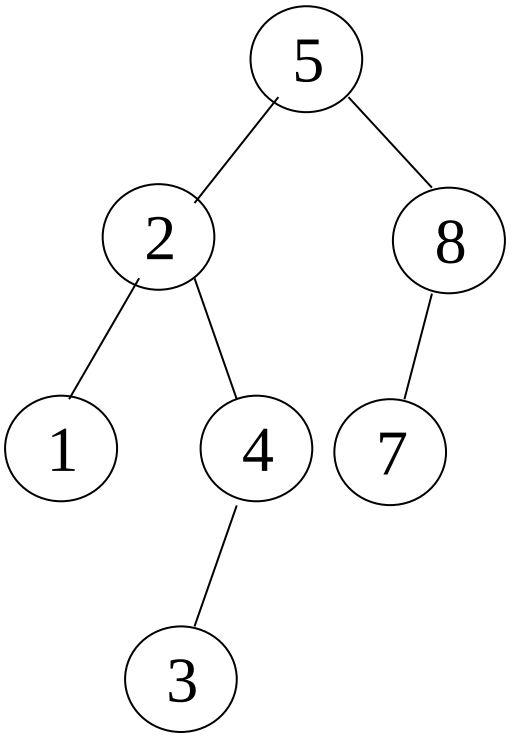
- We want a complete tree after every operation
  - tree is full except possibly in the lower right
- This is expensive
  - For example, insert 2 in the tree on the left and then rebuild as a complete tree



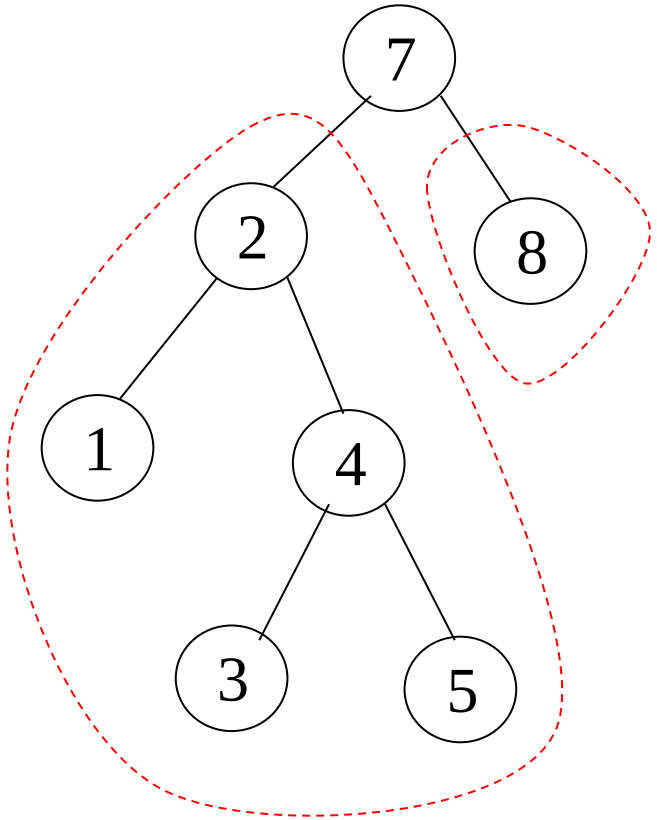
AVL - Good but not Perfect Balance

- AVL trees are height-balanced binary search trees.
- An AVL tree has **balance factor** calculated at every node.
  - For every node, heights of left and right subtree can differ by no more than 1

AVL Trees



An AVL Tree



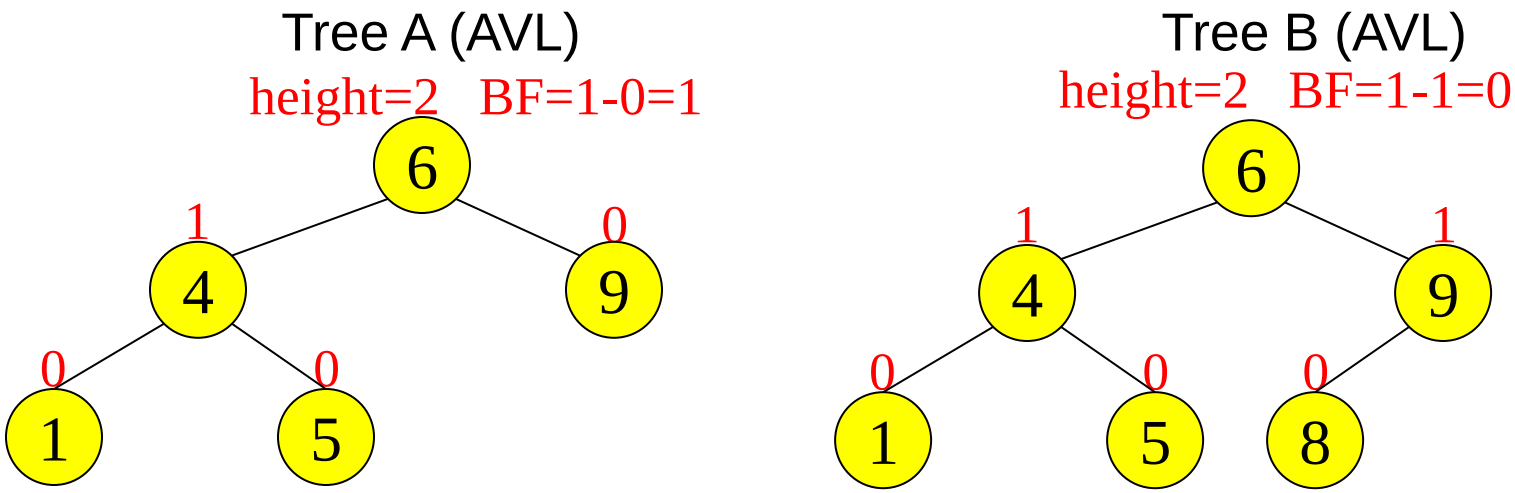
Not an AVL Tree

AVL Trees

Balancing Factor:

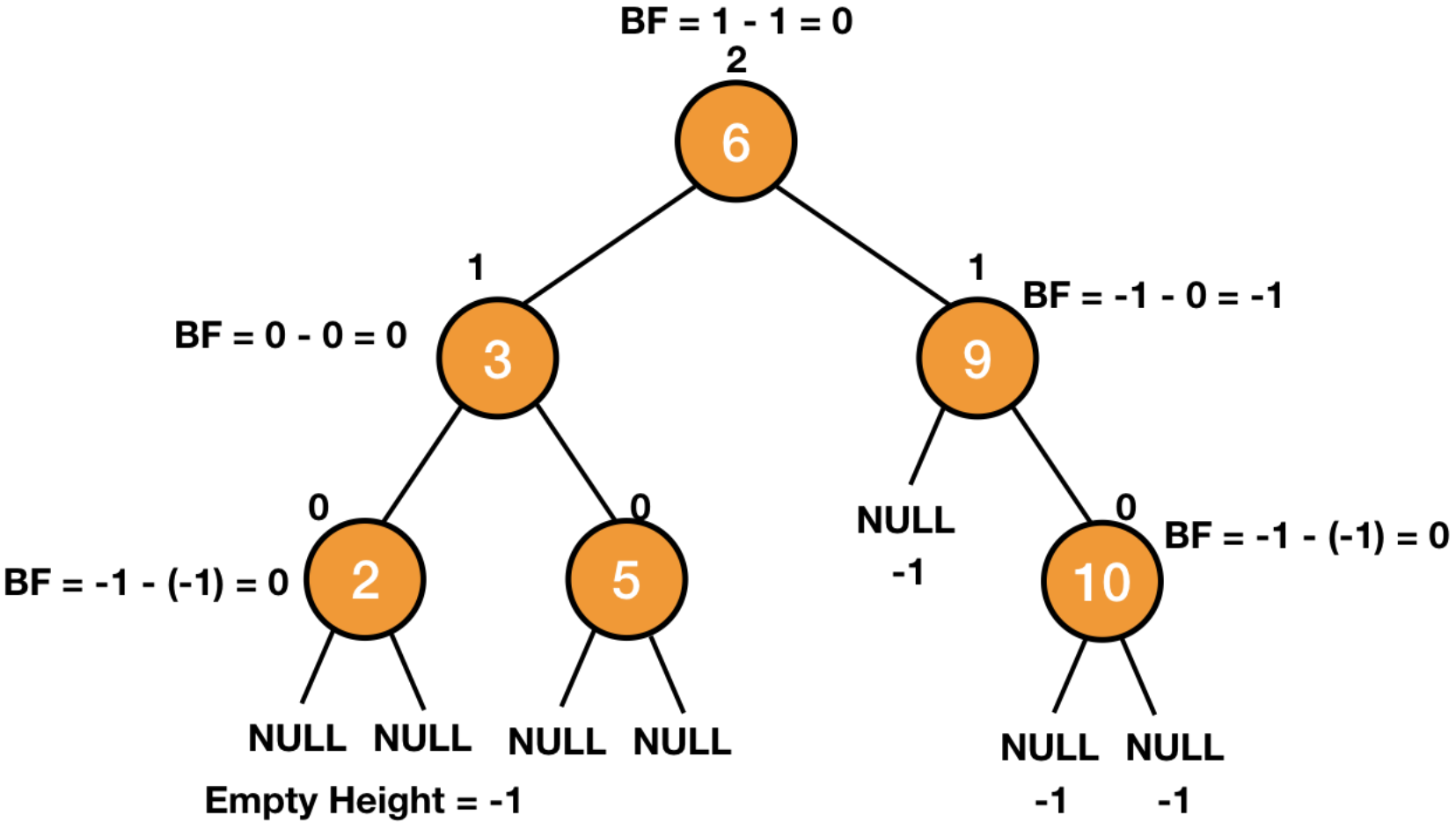
- The height of the left subtree minus the height of the right subtree of a node is called the *balance of the node(Balancing Factor)*.
  - For an AVL tree, the Balance Factors (BF) of the nodes are always -1, 0 or 1.
  - BF= height(left sub-tree) – height(right sub-tree)
- The height of an empty tree is defined to be 0.

Node Heights



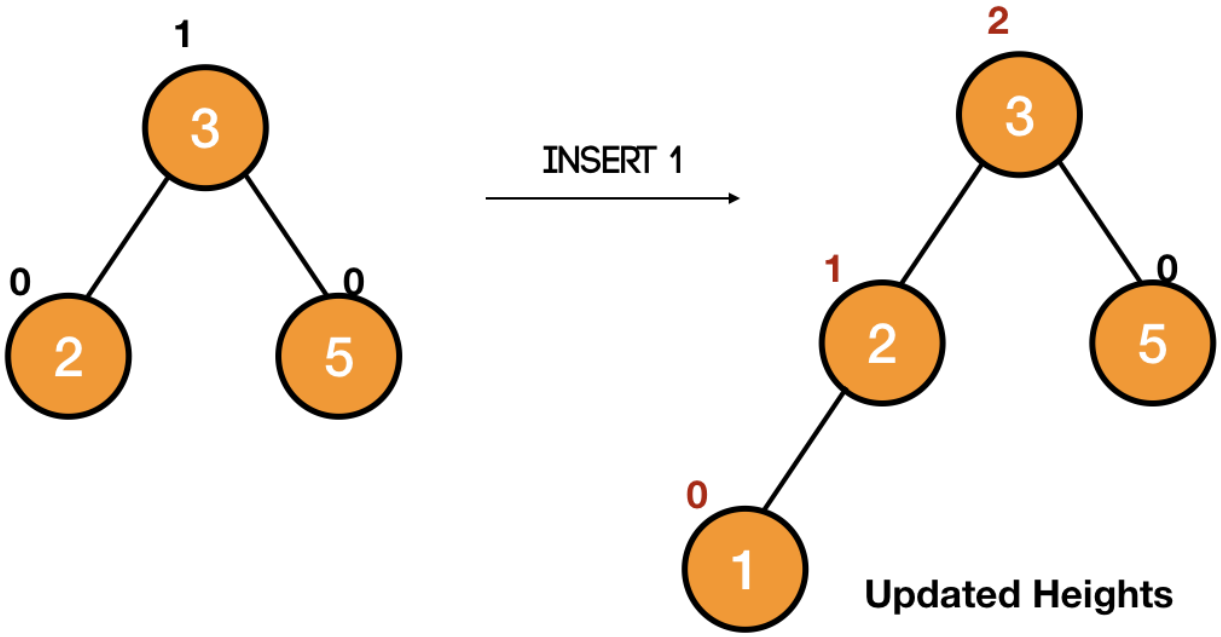
height of node = h
balance factor = h_{left}-h_{right}
empty height = 0

AVL Trees


BF = 1 - 1 = 0
2
6
1
1
BF = 0 - 0 = 0
3
9
BF = -1 - 0 = -1
0
0
NULL
-1
0
BF = -1 - (-1) = 0
BF = -1 - (-1) = 0
2
5
10
NULL NULL NULL NULL
NULL NULL
-1 -1
Empty Height = -1

AVL Tress

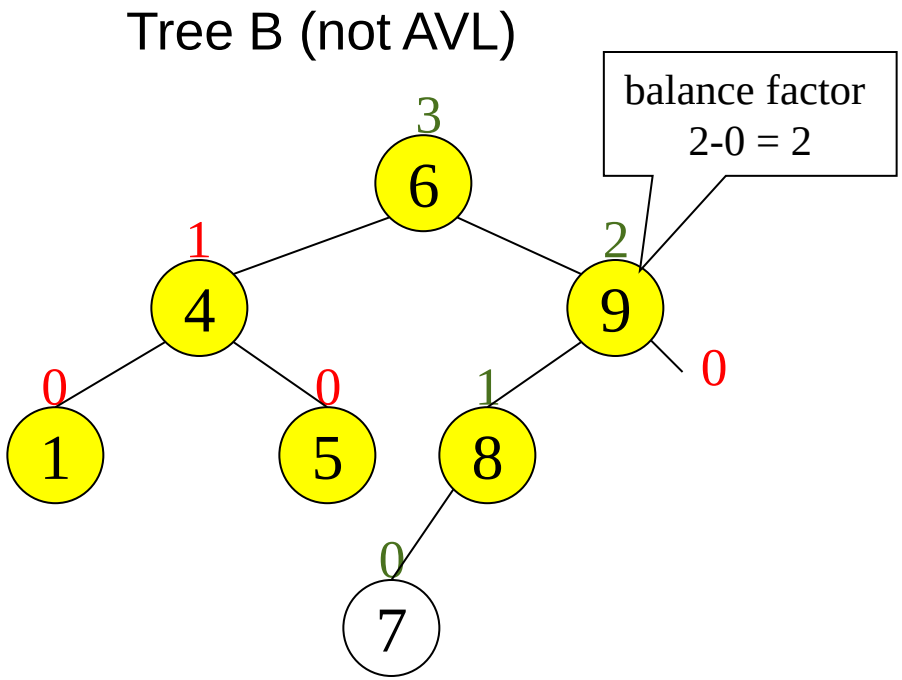
- Given an AVL tree, if insertions or deletions are performed, the AVL tree *may not* remain height balanced.



AVL Tress

- Given an AVL tree, if insertions or deletions are performed, the AVL tree *may not* remain height balanced.

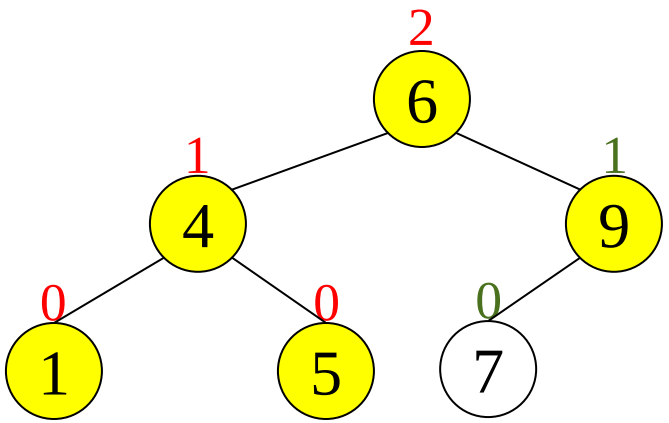
For Example: After Insertion 7, *the AVL tree becomes height unbalanced.*



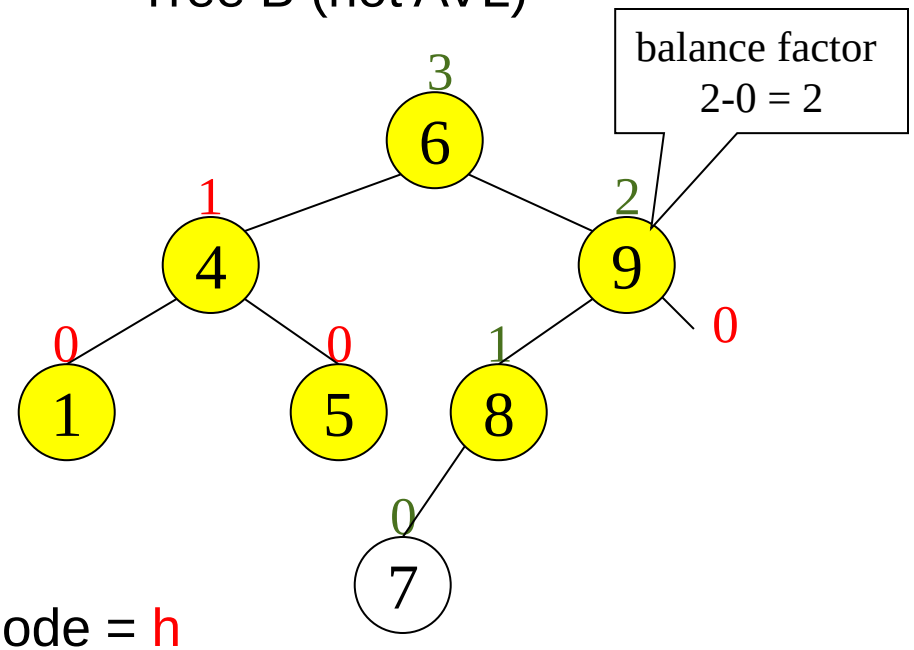
Node Heights

Node Heights after Insert 7:

Tree A (AVL)

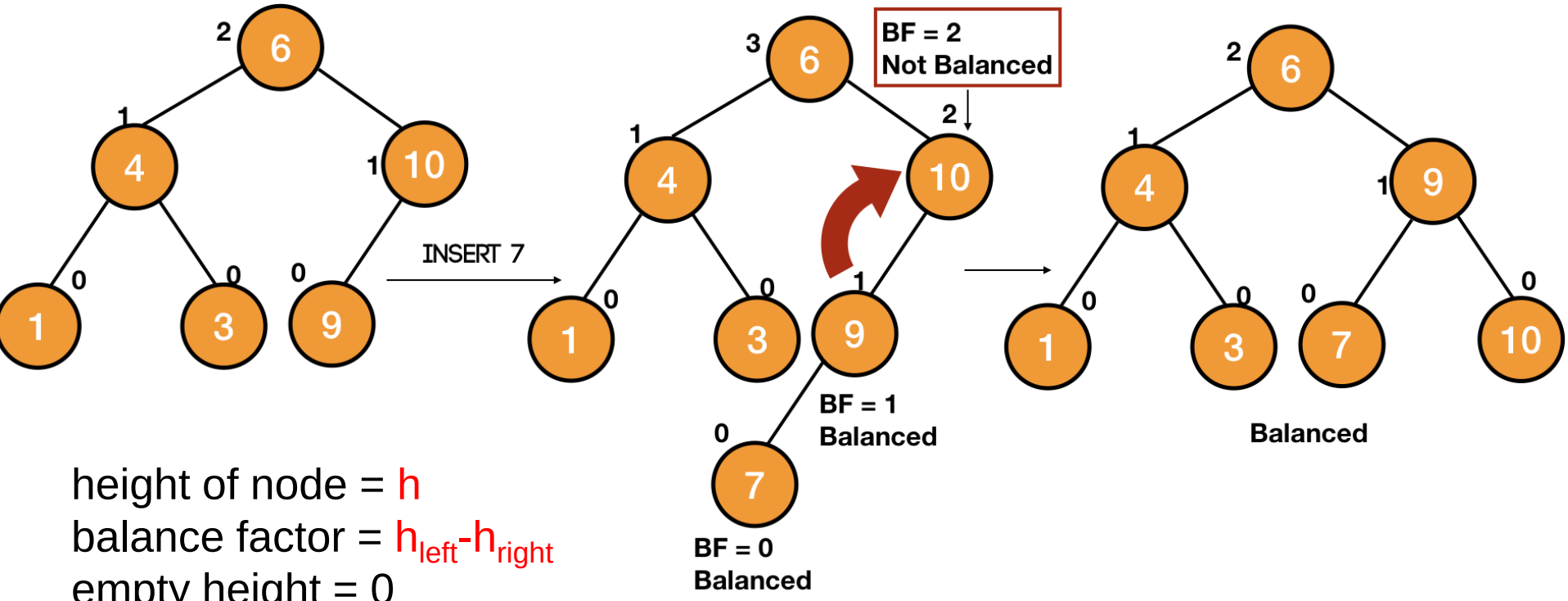


Tree B (not AVL)



height of node = h
balance factor = h_{left}-h_{right}
empty height = 0

Node Heights



height of node = h
balance factor = h_{left}-h_{right}
empty height = 0

AVL Trees

To maintain the height balanced property of the AVL tree after insertion or deletion, it is necessary to perform a *transformation* on the tree so that,

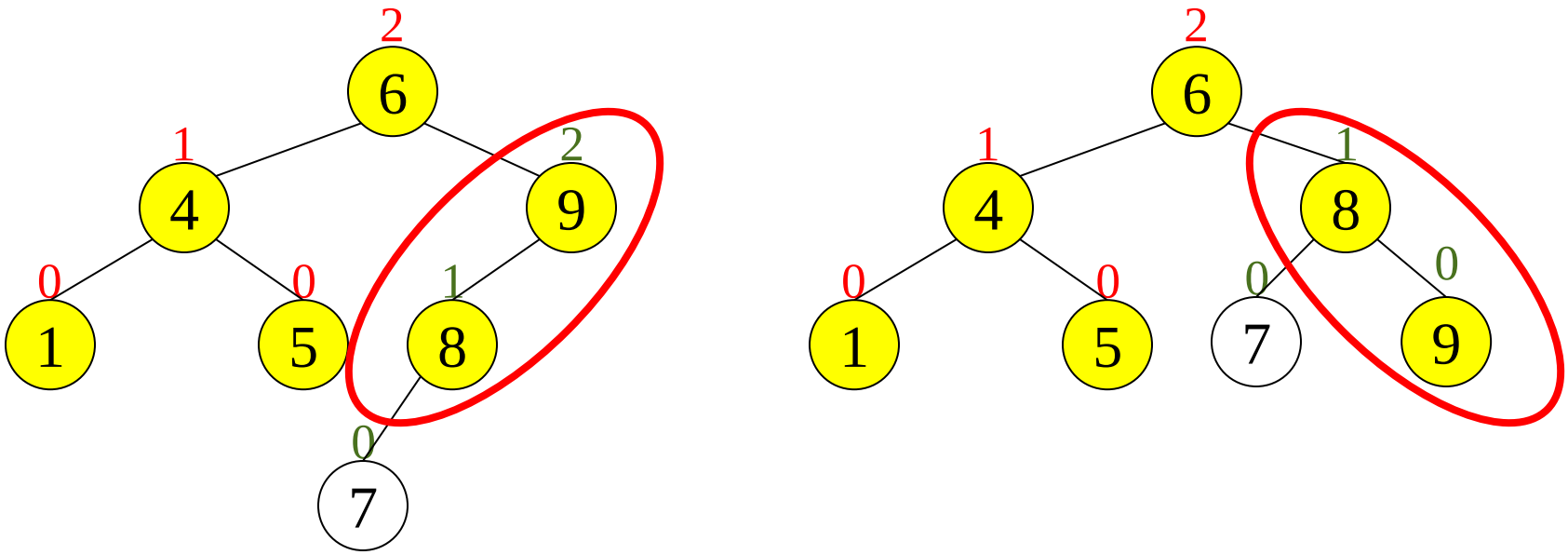
(1) the *in-order traversal of the transformed tree is the same as for the original tree* (i.e., the new tree remains a binary search tree).

(2) the tree after transformation is height-balanced.

Insertion in AVL Trees

- Insert operation may cause balance factor to become 2 or –2 for some node
  - only nodes on the path from insertion point to root node have possibly changed in height
  - Follow the path up to the root, find the first node (i.e., deepest) whose new balance violates the AVL condition. Call this node *a*
  - If a new balance factor (the difference h_{left}-h_{right}) is 2 or –2, adjust tree by *rotation* around the node

Insertion in AVL Trees



Reading Materials

- Schaum's Outlines: Chapter # 7
- D. S. Malik: Chapter # 11
- Nell Dale: Chapter # 8
- Allen Weiss: Chapter # 4
- Tenebaum: Chapter # 5