ECE250: Algorithms and Data Structures

AVL Trees (Part A)

Ladan Tahvildari, PEng, SMIEEE

Associate Professor

Software Technologies Applied Research (STAR) Group

Dept. of Elect. & Comp. Eng.

University of Waterloo

Materials from Weiss: Chapter 4.4.1

Binary Search Tree (BST)

- Worst case height of BST: N-1
  - Insertion, deletion can be O(N) in the worst case
- We want a tree with small height
- Height of a binary tree with N node is at least $\Theta(\log N)$
- Goal: keep the height of a binary search tree $O(\log N)$
- Balanced binary search trees

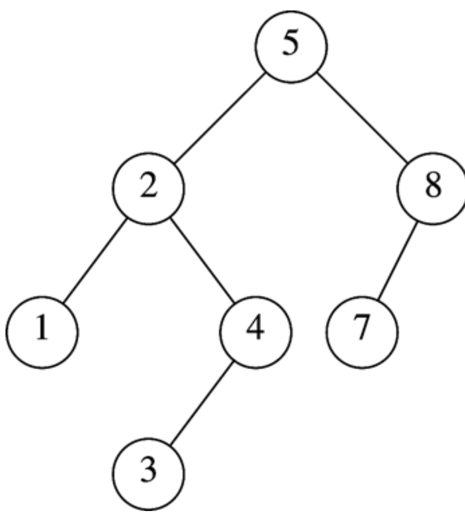
Balanced Tree?

- Suggestion 1: the left and right subtrees of root have the same height
  - But the left and right subtrees may be linear lists!
- Suggestion 2: every node must have left and right subtrees of the same height
  - Only complete binary trees satisfy
  - Too rigid to be useful
- Our choice: for each node, the height of the left and right subtrees can differ at most 1

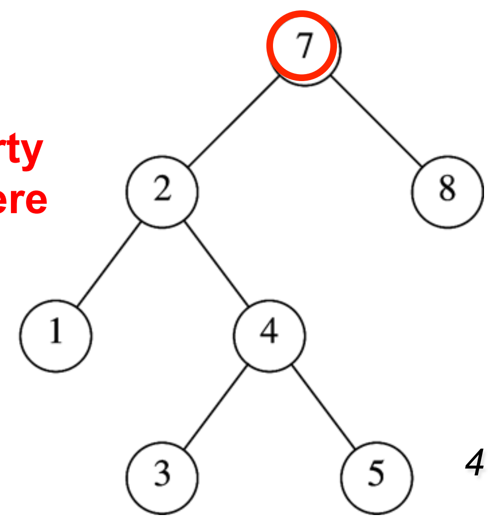
AVL Tree

- AVL tree is the first balanced binary search tree (name after its discovers, Adelson-Velskii and Landis).
- An AVL tree is a BST in which
  - for *every* node in the tree, the height of the left and right subtrees differ by at most 1.
- Height of subtree: Max # of edges to a leaf
- Height of an empty subtree: -1
  - Height of one node: 0

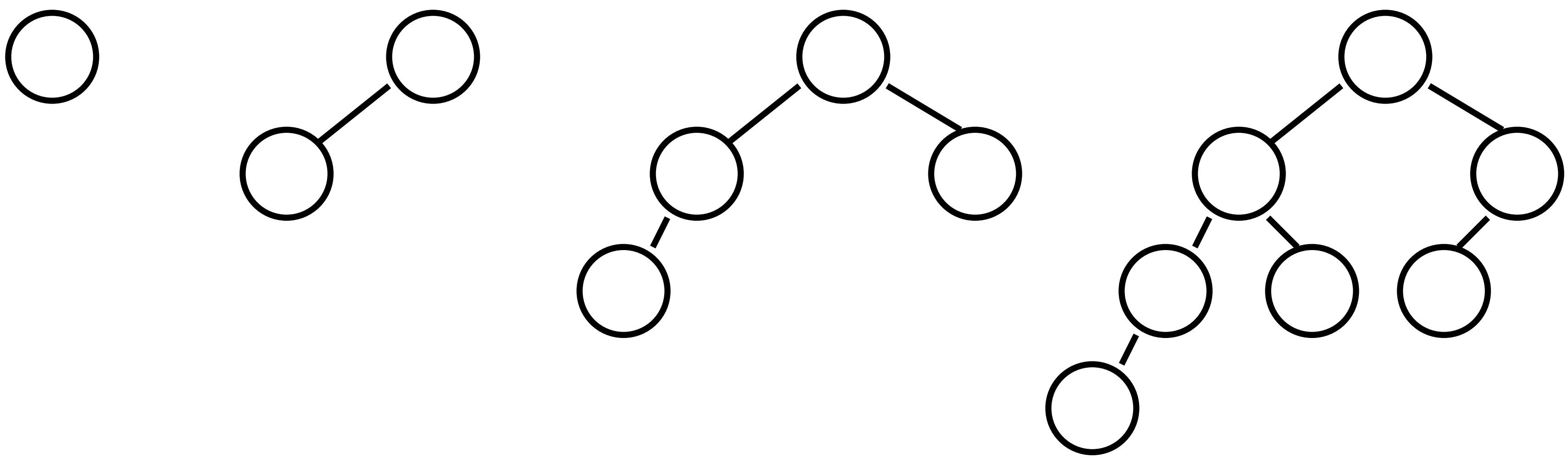
AVL tree



AVL property violated here



AVL Tree - Minimum Number of Nodes



$N_0 = 1$ $N_1 = 2$ $N_2 = 4$ $N_3 = N_1 + N_2 + 1 = 7$

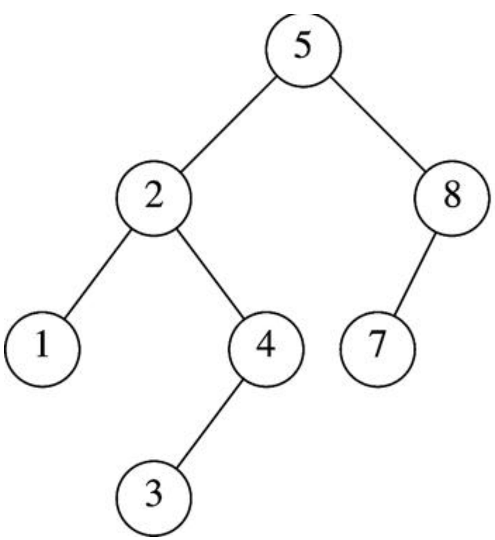
height of left=?
height right=?

Height of AVL Tree

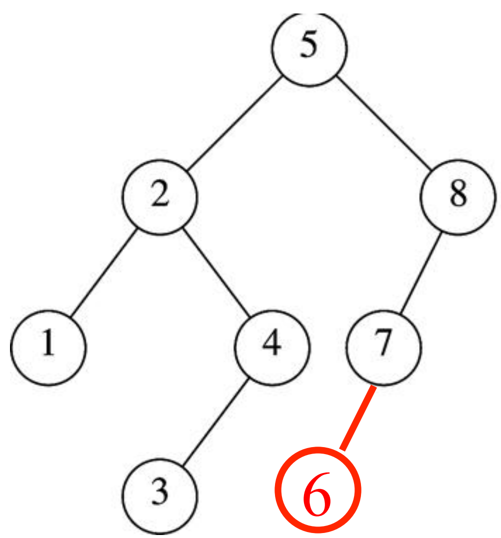
- Denote N_h the minimum number of nodes in an AVL tree of height h
- $N_0=1$, $N_1=2$ (base)
  $N_h= N_{h-1} + N_{h-2} +1$ (recursive relation)
- $N > N_h= N_{h-1} + N_{h-2} +1$
  $>2 N_{h-2} >4 N_{h-4} >\ldots>2^i N_{h-2i}$
- If h is even, let $i=h/2-1$. The equation becomes $N>2^{h/2-1}N_2$
  $\Rightarrow N>2^{h/2-1}x4 \Rightarrow h=O(\log N)$
- If h is odd, let $i=(h-1)/2$. The equation becomes $N>2^{(h-1)/2}N_1$
  $\Rightarrow N>2^{(h-1)/2}x2 \Rightarrow h=O(\log N)$
- Thus, many operations (i.e. searching) on an AVL tree will take O(log N) time

Insertion in AVL Tree

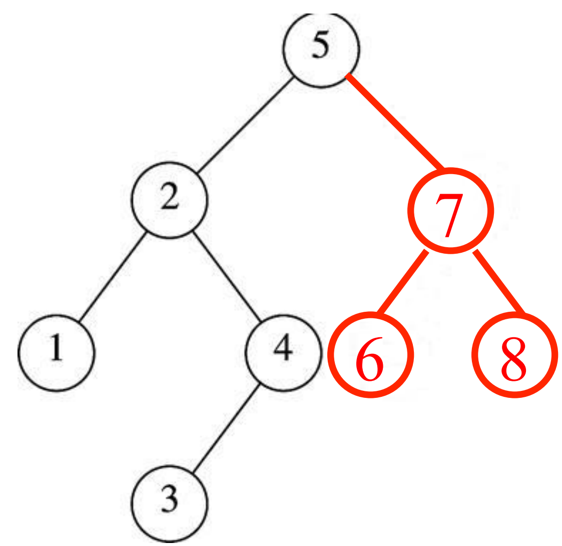
- Basically follows insertion strategy of binary search tree
  - But may cause violation of AVL tree property
- Restore the destroyed balance condition if needed



Original AVL tree



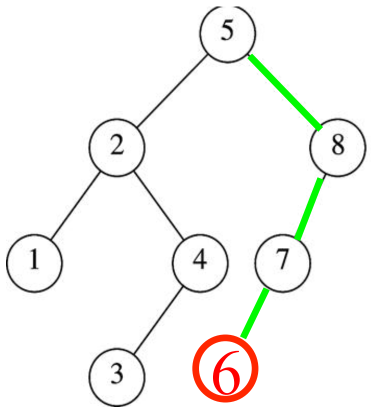
Insert 6- Property Violated



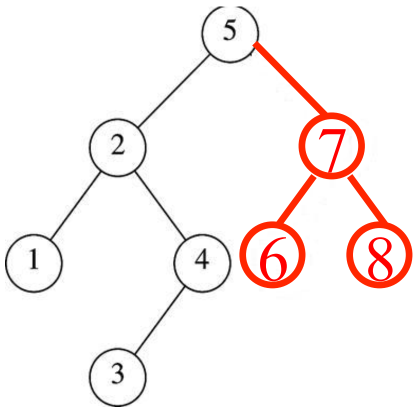
Restore AVL property

Some Observations

- After an insertion, only nodes that are on the path from the insertion point to the root might have their balance altered
  - Because only those nodes have their subtrees altered
- Rebalance the tree at the deepest such node guarantees that the entire tree satisfies the AVL property



Node 5,8,7 might have balance altered



Rebalance node 7 guarantees the whole tree be AVL

Different Cases for Rebalance

- Denote the **node** that must be rebalanced **α**
  - Case 1: an insertion into the left subtree of the left child of α
  - Case 2: an insertion into the right subtree of the left child of α
  - Case 3: an insertion into the left subtree of the right child of α
  - Case 4: an insertion into the right subtree of the right child of α

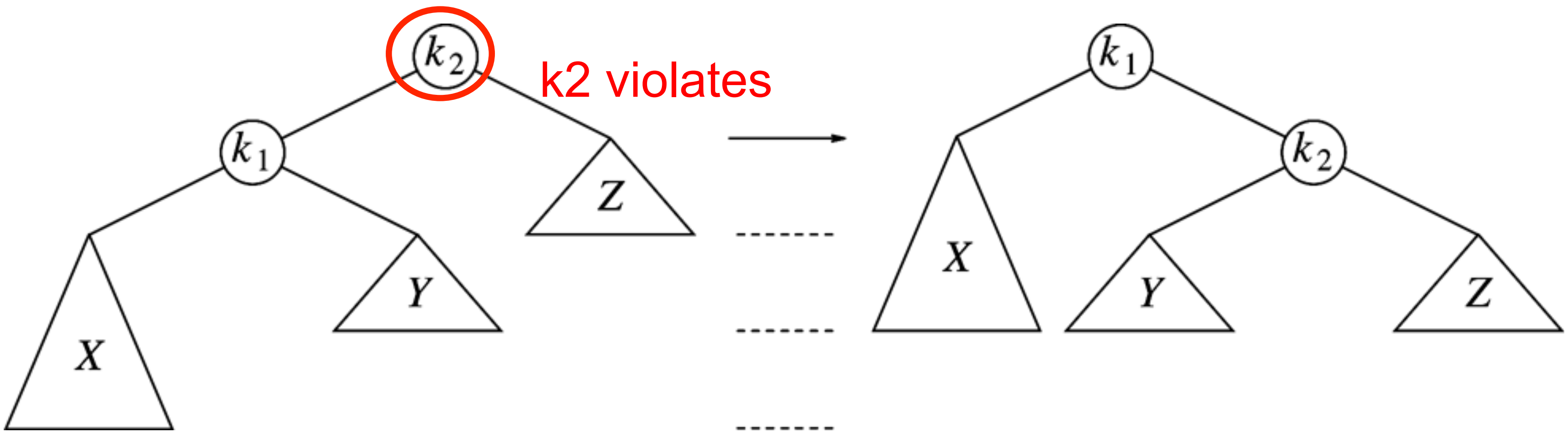
Rotations

- Rebalance of AVL tree are done with simple modification to tree, known as rotation
- Insertion occurs on the “outside” (i.e., left-left or right-right) is fixed by single rotation of the tree
- Insertion occurs on the “inside” (i.e., left-right or right-left) is fixed by double rotation of the tree

Insertion Algorithm

- First, insert the new key as a new leaf just as in ordinary binary search tree
- Then, trace the path from the new leaf towards the root. For each node x encountered, check if heights of left(x) and right(x) differ by at most 1
  - If yes, proceed to parent(x)
  - If not, restructure by doing either a single rotation or a double rotation
- **Note:** once we perform a rotation at a node x, we won't need to perform any rotation at any ancestor of x.

Single Rotation to Fix Case 1(left-left)



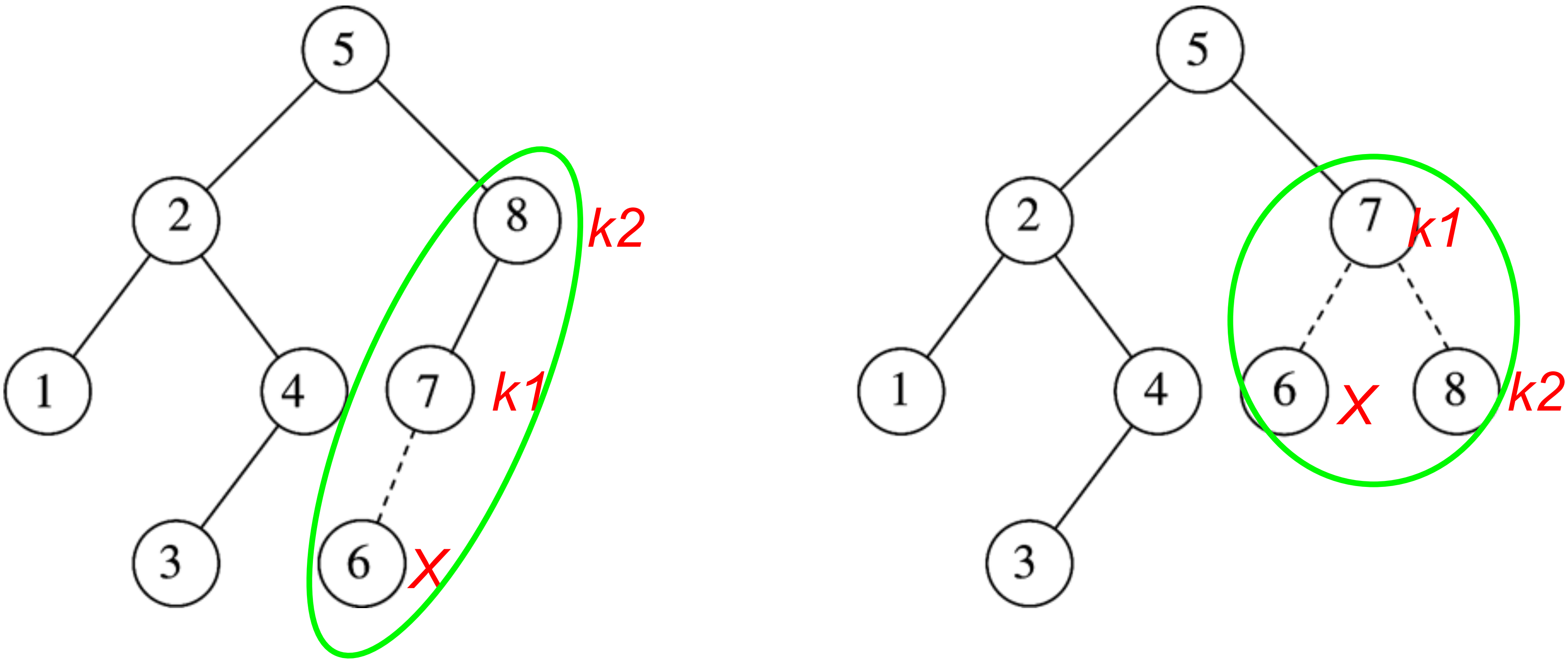
An insertion in subtree X,
AVL property violated at node k2

Solution: single rotation

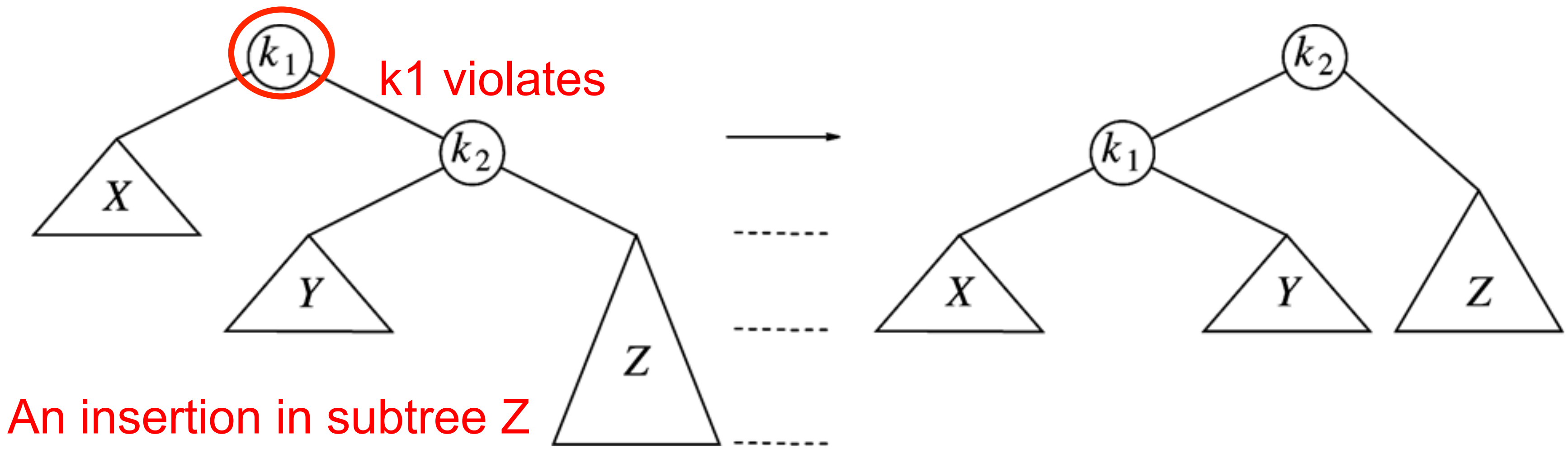
AVL-property quiz:

1. Can Y have the same height as the new X?
2. Can Y have the same height as Z?

Single Rotation Case 1 - Example



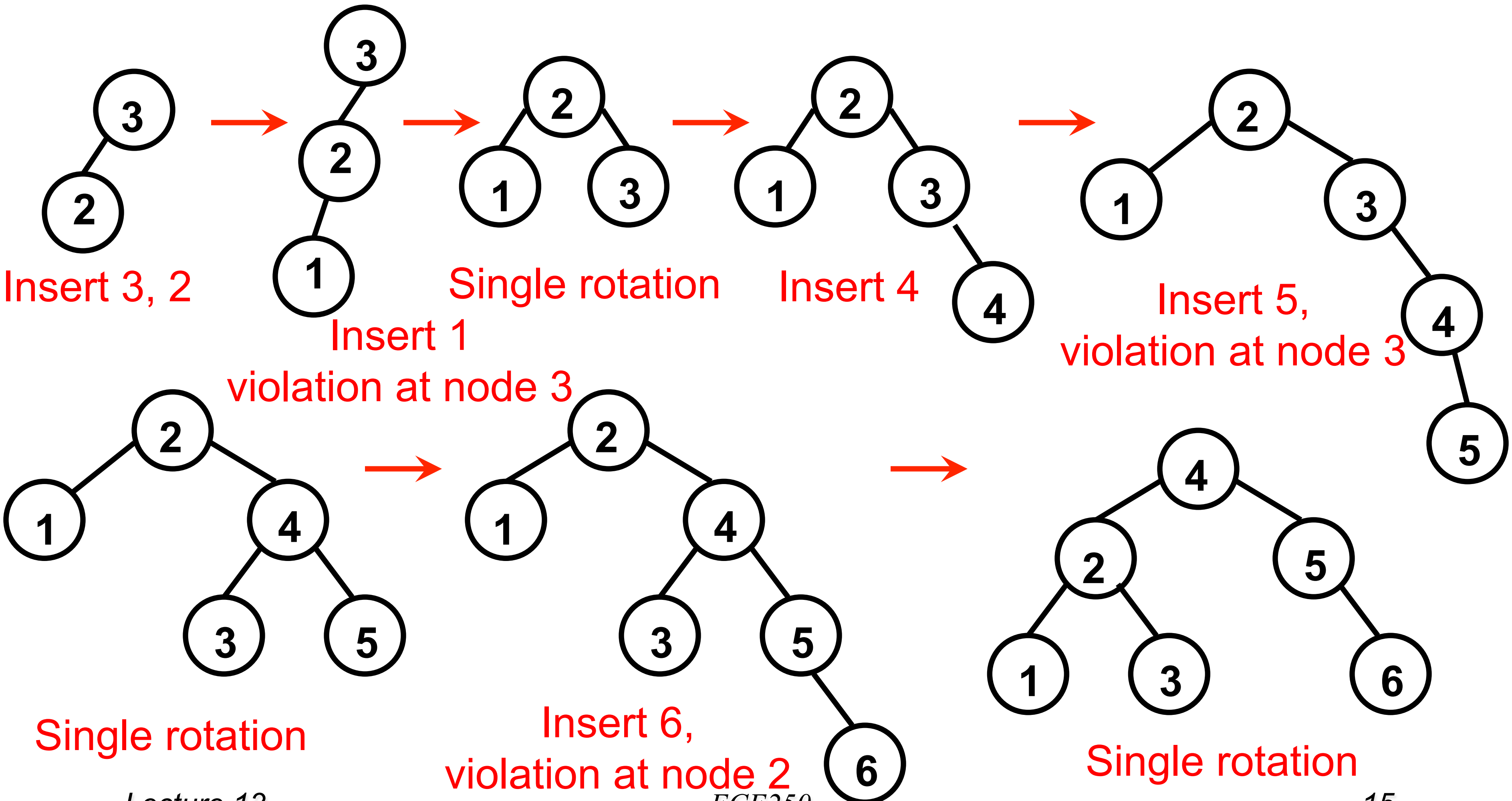
Single Rotation to Fix Case 4 (right-right)



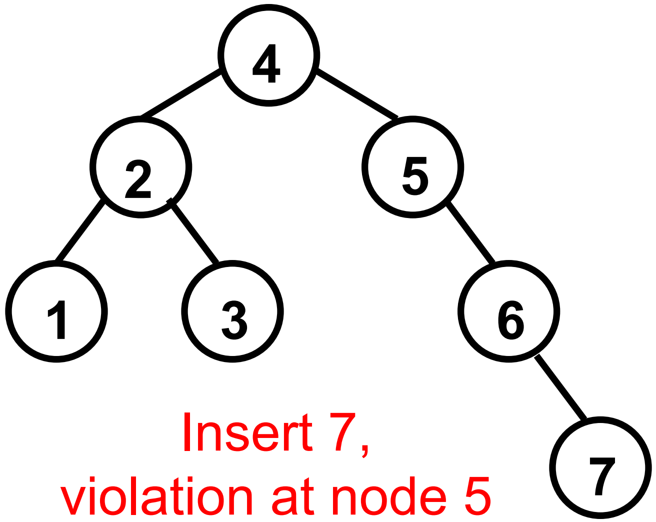
- Case 4 is a symmetric case to case 1
- Insertion takes O(Height of AVL Tree) time, Single rotation takes O(1) time

Example - Single Rotation

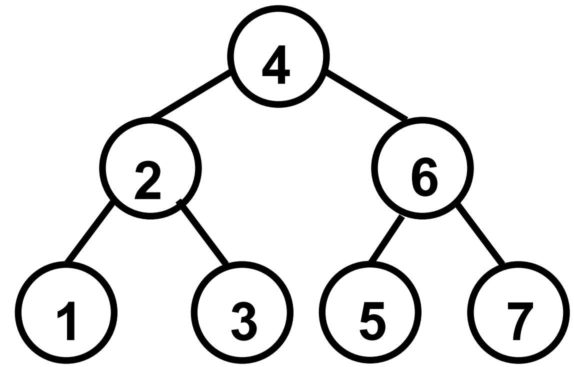
Sequentially insert 3, 2, 1, 4, 5, 6 to an AVL Tree



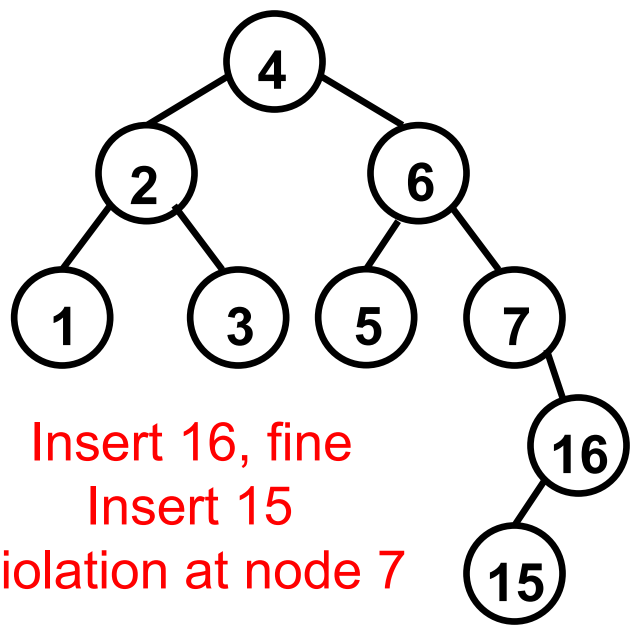
Continue to insert 7, 16, 15, 14, 13, 12, 11, 10, 8, 9



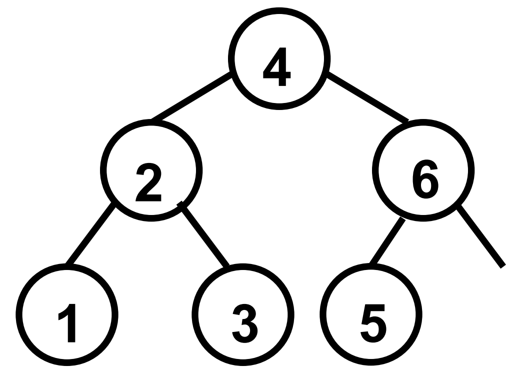
Insert 7,
violation at node 5



Single rotation



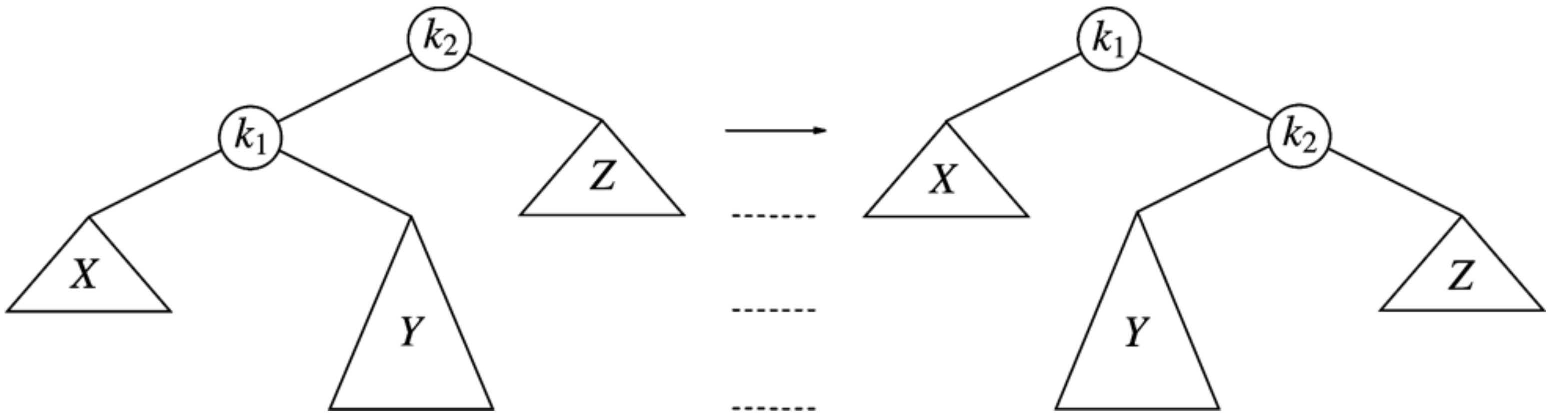
Insert 16, fine
Insert 15
violation at node 7



Single rotation
But….
Violation remains

What is the result?

Single Rotation Fails to fix Case 2&3



Case 2: violation in k2 because of insertion in subtree Y

Single rotation result

- **Single rotation fails to fix case 2&3**
- Take case 2 as an example (case 3 is a symmetry to it)
  - The problem is subtree Y is too deep
  - Single rotation doesn't make it any less deep

Single Rotation Fails

- What shall we do?
- We need to rotate twice
  - Double Rotation