Data Structures and Algorithms

Lecture 10: AVL Trees

Department of Computer Science & Technology
United International College

Problem with BST

- The cost for all the operations are bounded by the tree height
- Worst case height of a binary search tree is n-1

All the operations are O(n)

- Goal: Keep the height of a binary search tree O(log(n))
- Solution: (somehow) Balanced binary search trees

Balanced Trees

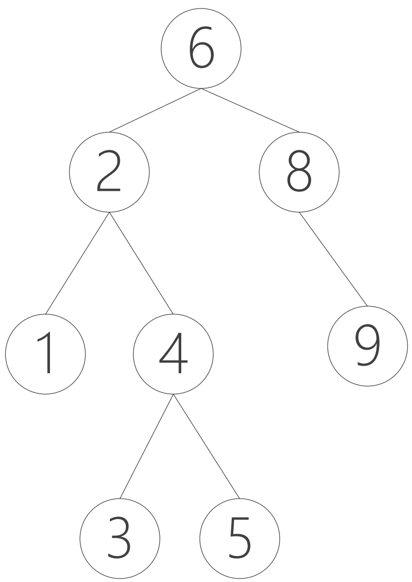
- Suggestion 1: the left and right subtrees of root have the same height
  - Tree height is O(n)
- Suggestion 2: every node must have left and right subtrees of the same height
  - Too rigid to be useful
- Our choice: for each node, the height of the left and right subtrees can differ at most 1

AVL Tree

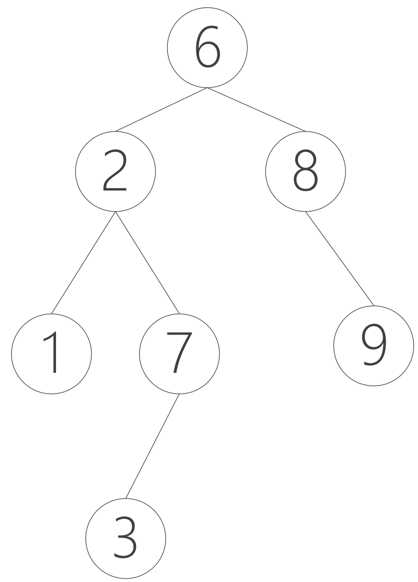
AVL Tree

- An AVL tree is a binary search tree in which
  - for every node in the tree, the height of the left and right subtrees differ by at most 1.
- Height of subtree
  - Number of edges to the deepest leaf
- Height of an empty subtree
  - -1

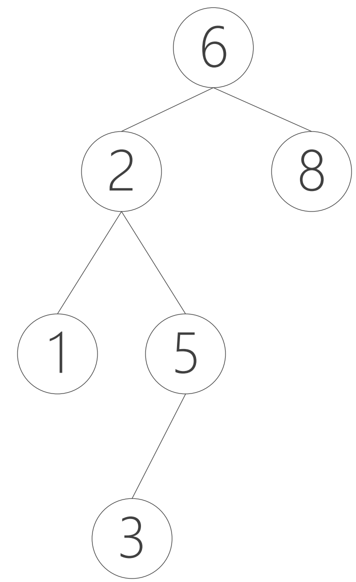
AVL Tree Examples



An AVL Tree



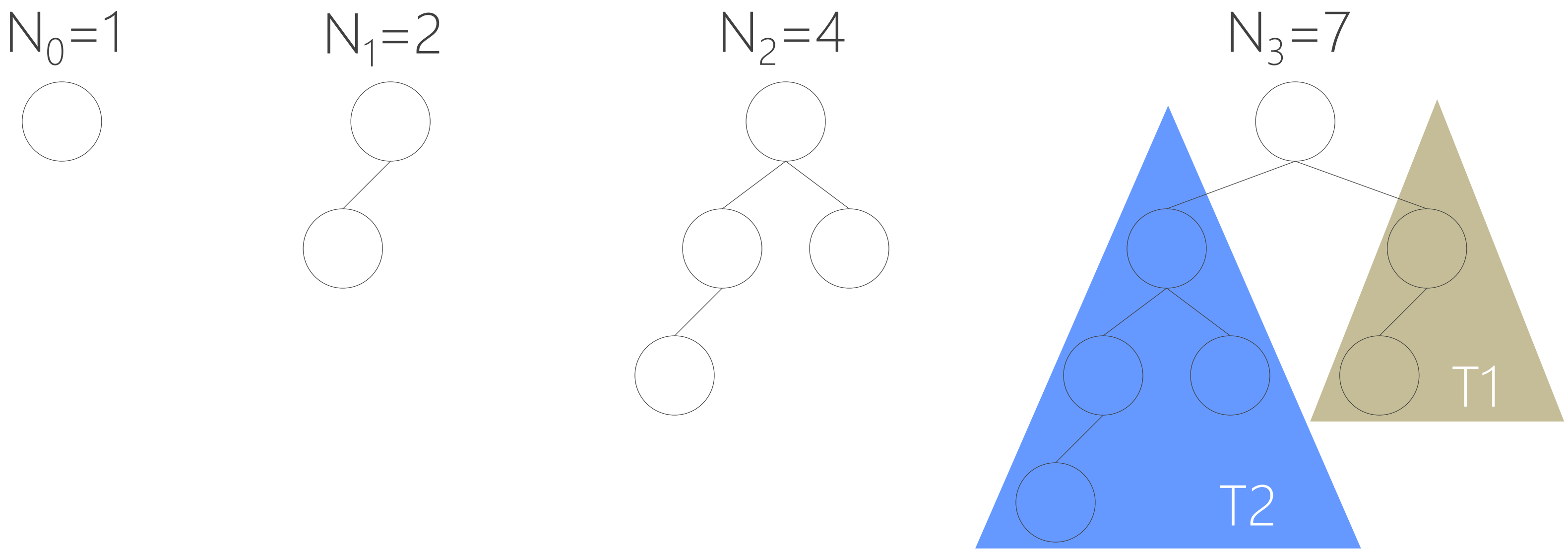
Not An AVL Tree
WHY?



Not An AVL Tree
WHY?

Height of an AVL Tree

- To analyze the relation between tree height and the number of nodes in a tree, we list the minimum AVL trees of all heights



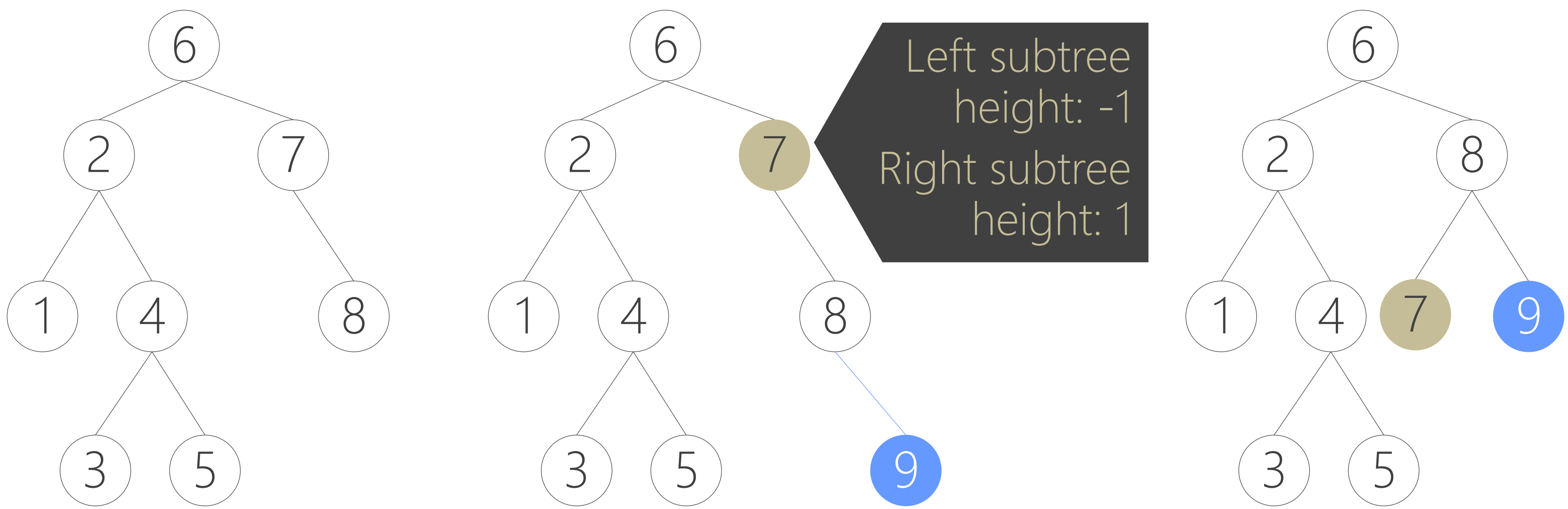
Relation: $N_h=1+N_{h-1}+N_{h-2}$

Height of an AVL Tree

- It can be proved that the height of an AVL tree is O(log(n))
  - HOW?
- Further, an AVL tree is a BST
- Therefore, all the operations on an AVL tree take O(log(n)) time
- The AVL tree property must be maintained on each
  - insert
  - delete

Insertion

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



Valid AVL tree

Insert 9

Restore AVL property

Some Observation

- 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
  - Insertion increases the height of the altered subtree by 1
  - Rebalance decreases the height of the altered subtree by 1
  - No height change!

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
- Cases 1&4 are mirror image symmetries with respect to α, as are cases 2&3

Rotations

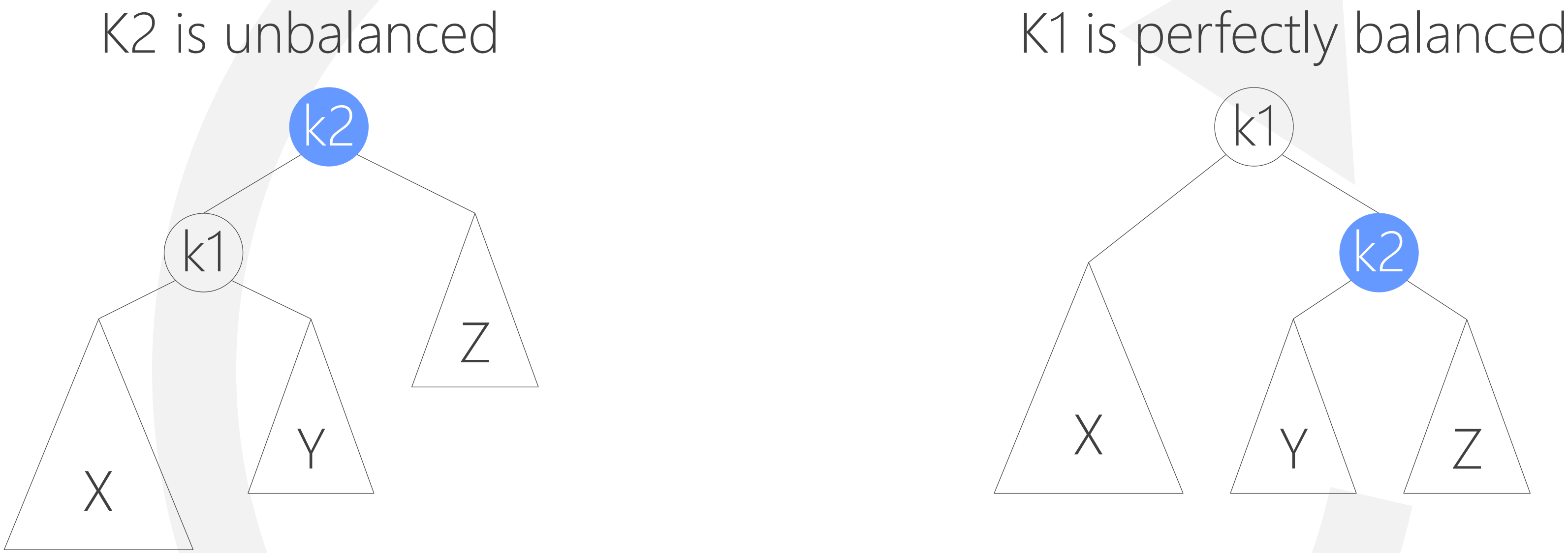
- Rebalance of AVL tree are done with simple modification to tree, known as rotation
- Insertion occurs on the "outside" (i.e., left-left or right-right) is fixed by single rotation of the tree
- Insertion occurs on the "inside" (i.e., left-right or right-left) is fixed by double rotation of the tree
- Animation: https://visualgo.net/bn/bst

Insertion Algorithm

1. Insert the new key as a new leaf just as in ordinary binary search tree
2. 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

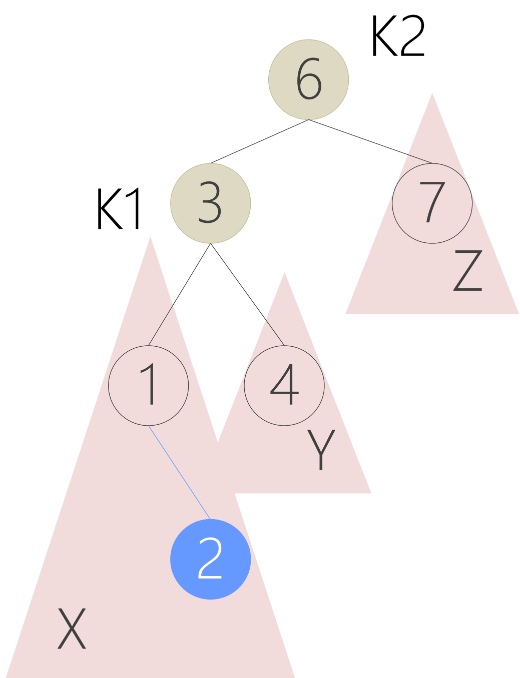
- Note: once we perform a rotation at a node x, we won't need to perform any rotation at any ancestor of x.

Single Right Rotation to Fix Case 1 (left-left)

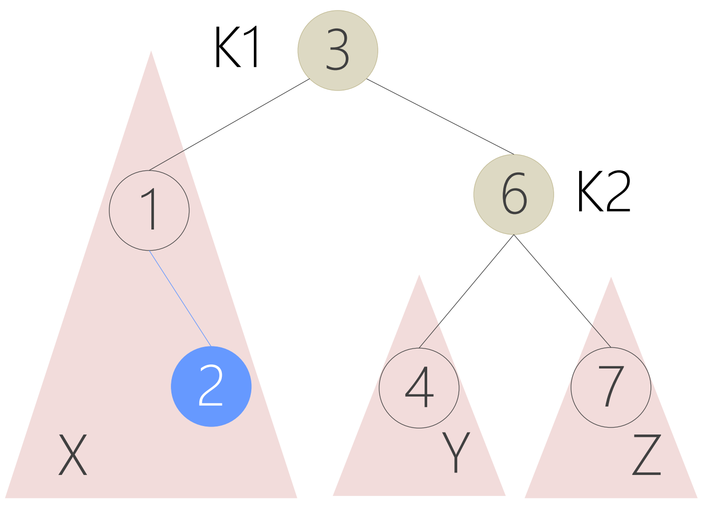


- Suppose that height(Z) = h
  - What is height(X)?
  - What is height(Y)?
- Note
  - Before insertion, the tree is balanced
  - After insertion, k_2 is case 1 unbalanced.

Single Right Rotation Example

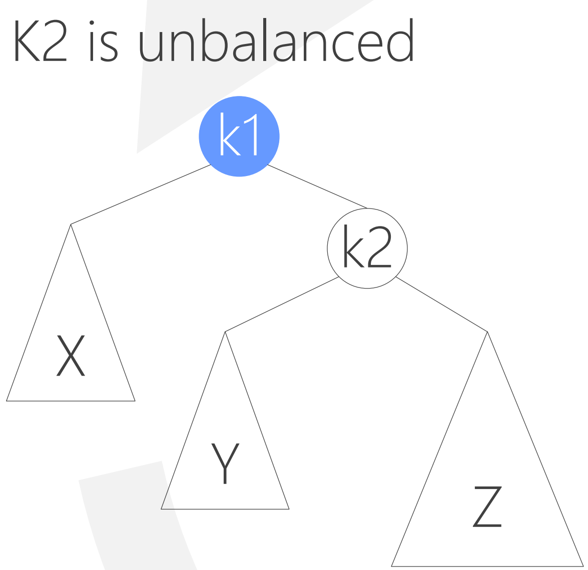


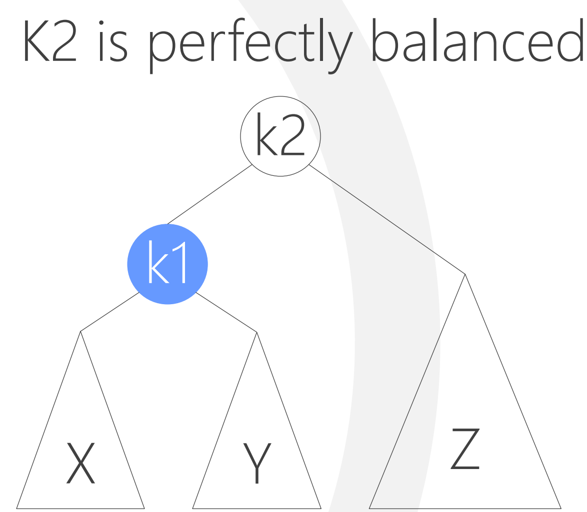
Node 2 is new



Case 1:
Single right Rotation

Single Left Rotation to Fix Case 4 (right-right)



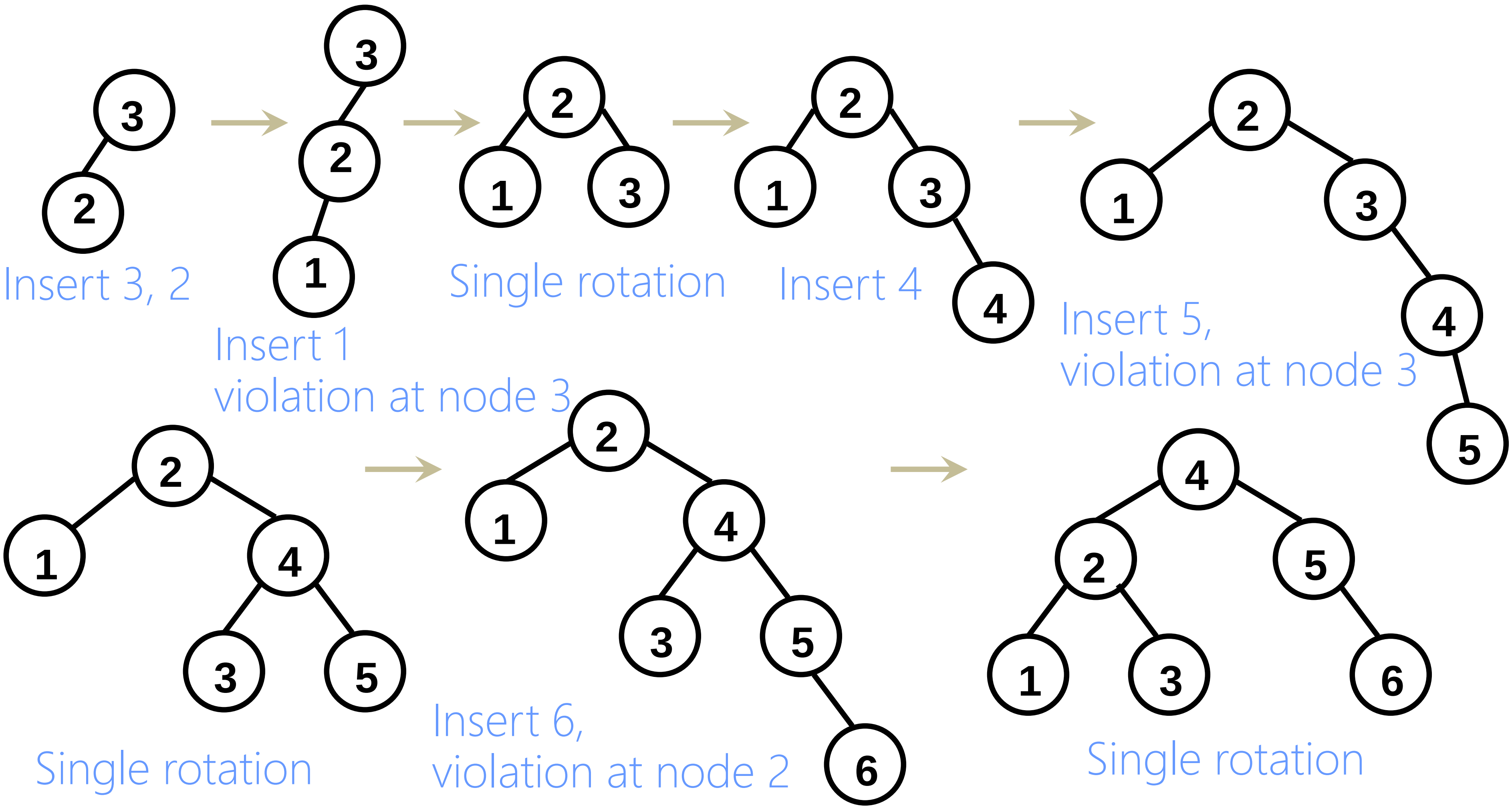


- Case 4 is symmetric to case 1
- Insertion takes O(log(n)) time
- Single rotation takes O(1) time

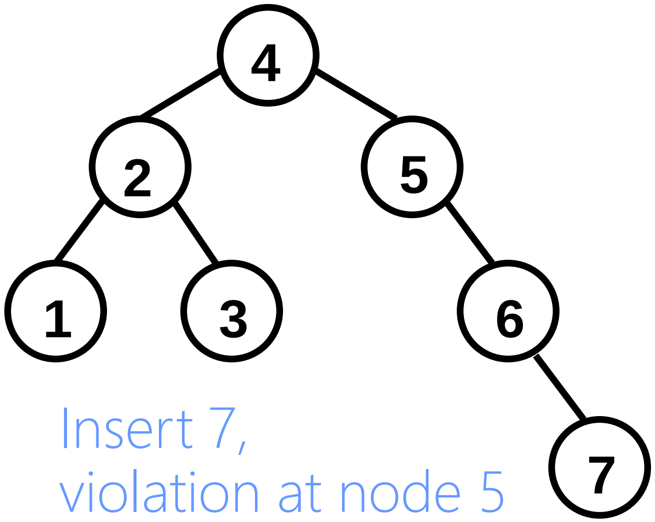
Total cost: O(log(n))

Single Rotation Example

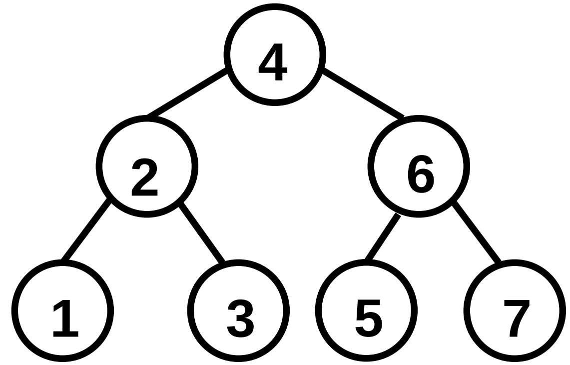
- Sequentially insert 3, 2, 1, 4, 5, 6 to an AVL Tree



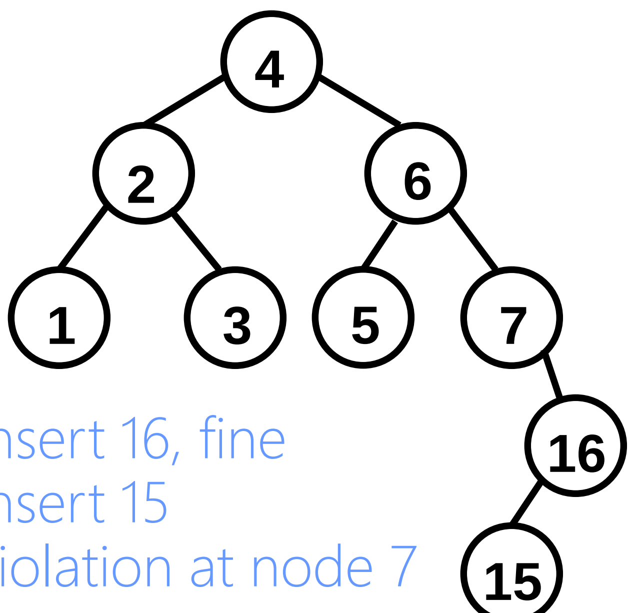
- If we continue to insert 7, 16, 15, 14, 13, 12, 11, 10, 8, 9



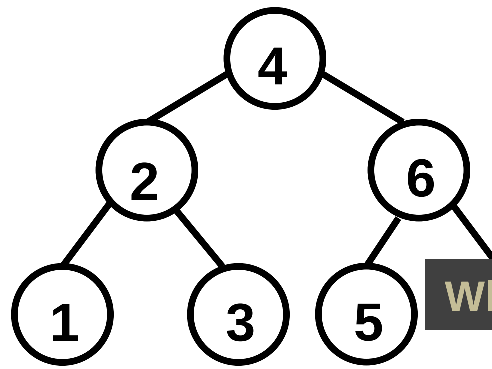
Insert 7,
violation at node 5



Single rotation



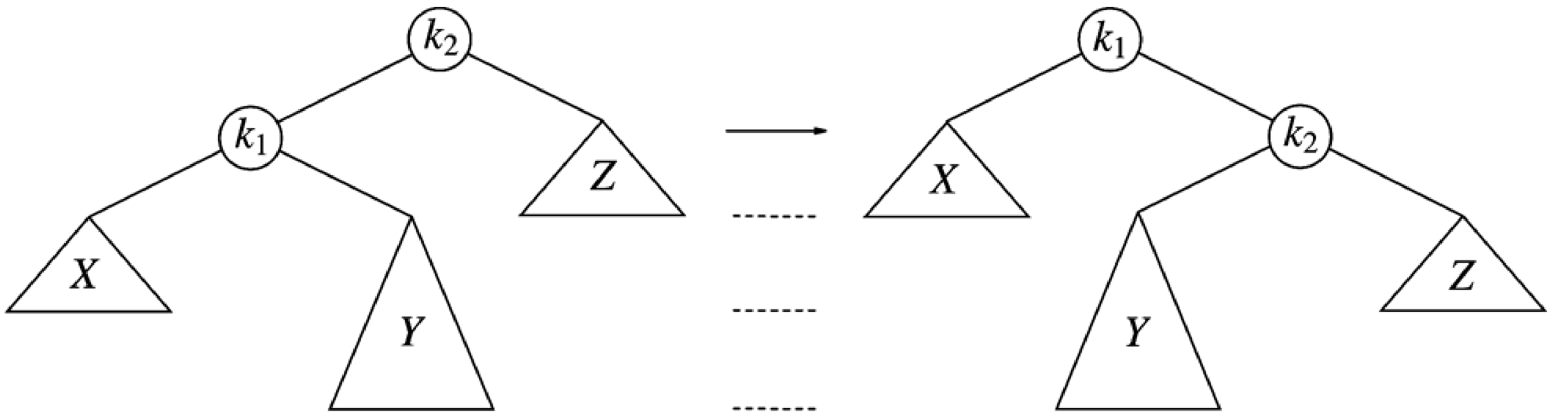
Insert 16, fine
Insert 15
violation at node 7



What is the result?

Single rotation
But….
Violation remains

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