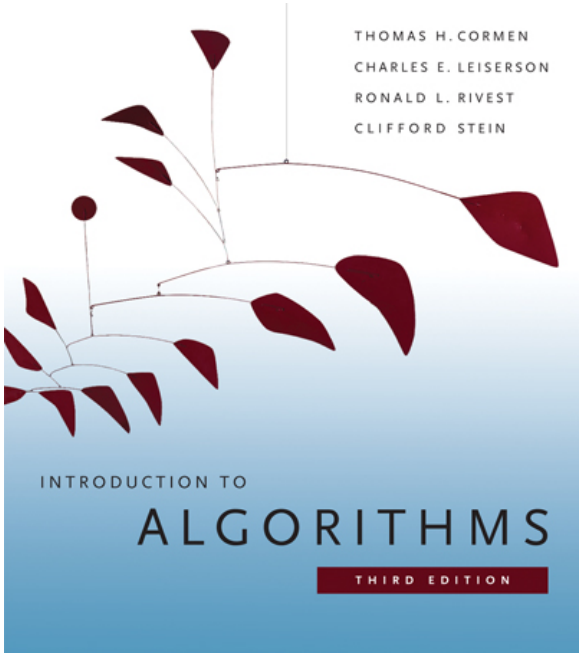
6.006- *Introduction to Algorithms*

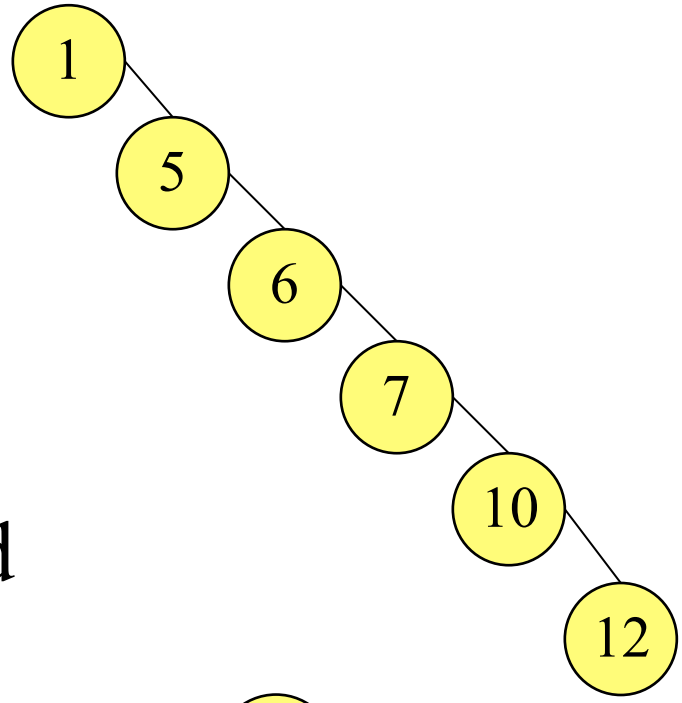


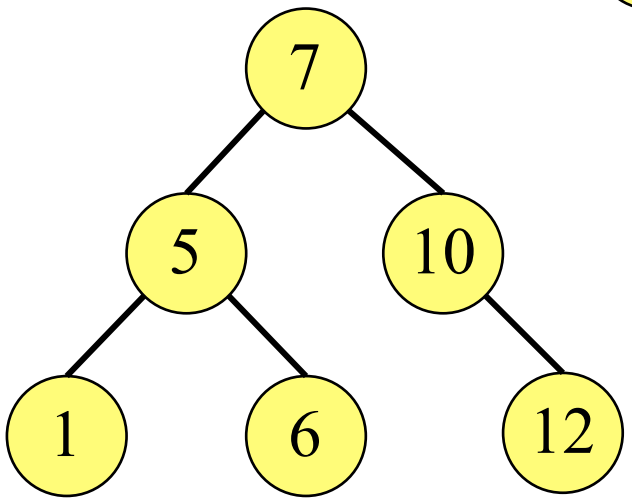
Lecture 4

Prof. Silvio Micali

Lecture Overview

- Review: Binary Search Trees
- Importance of being balanced
- Balanced BSTs
  - AVL trees
  - Other balanced trees

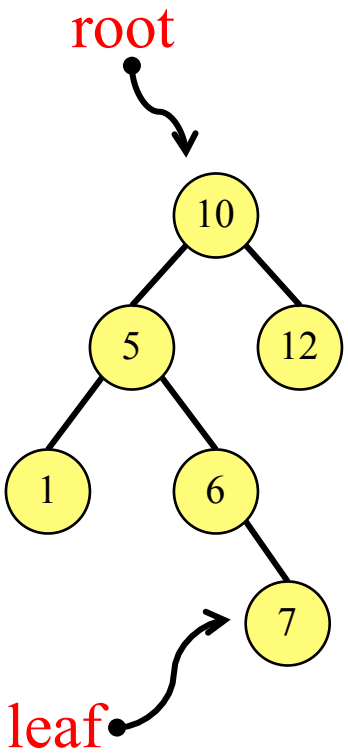




Binary Search Trees (BSTs)

- Each node x has:
  - key[x]
  - Pointers: left[x], right[x], p[x]

- Property: for any node x:
  - For all nodes y in the **left** subtree of x:
    $$key[y] \leq key[x]$$
  - For all nodes y in the **right** subtree of x:
    $$key[y] \geq key[x]$$

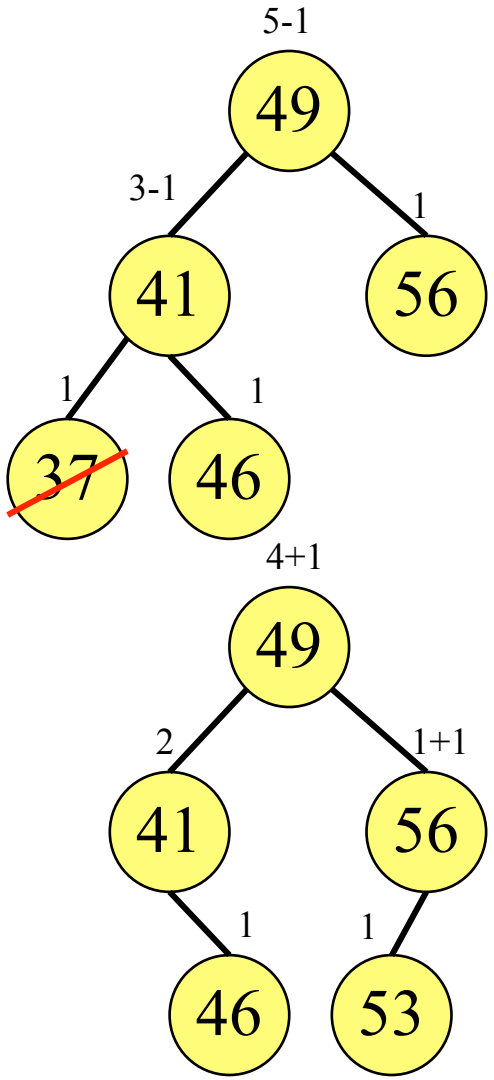


height = 3

BST Basic Operations

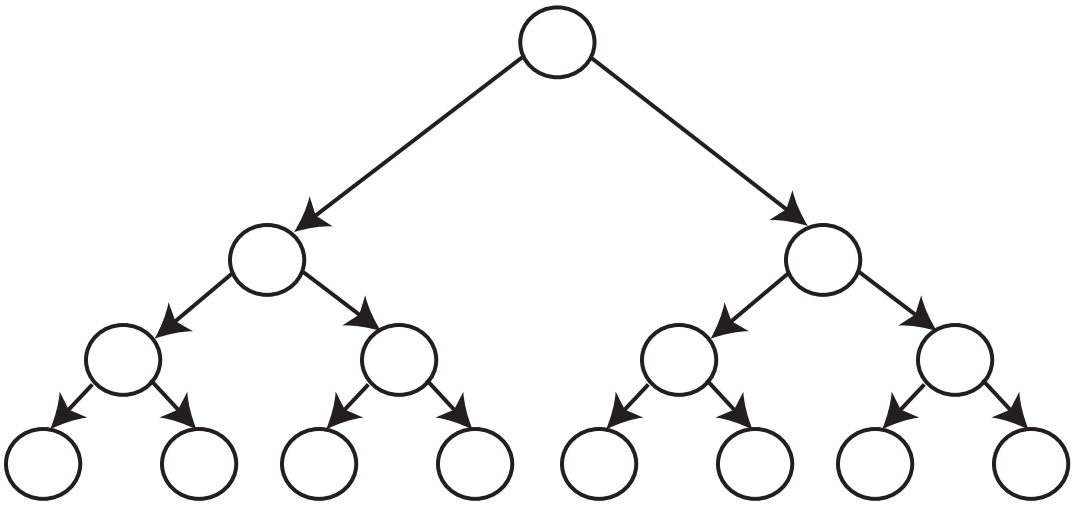
- Find, successor, min, …
- Remove an element (e.g., 37)
- Insert new element (e.g., 53)
- Delete & insert: O(h),

  where h is the height of the tree
- Useful to "augment" a BST (e.g., w/ tree size)

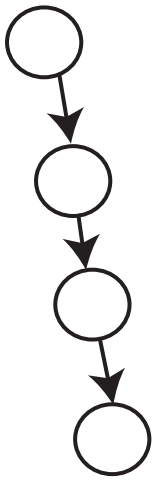


The importance of being balanced

for n nodes:



vs.



Perfectly Balanced

Path

$h = \Theta(\log n)$

$h = \Theta(n)$

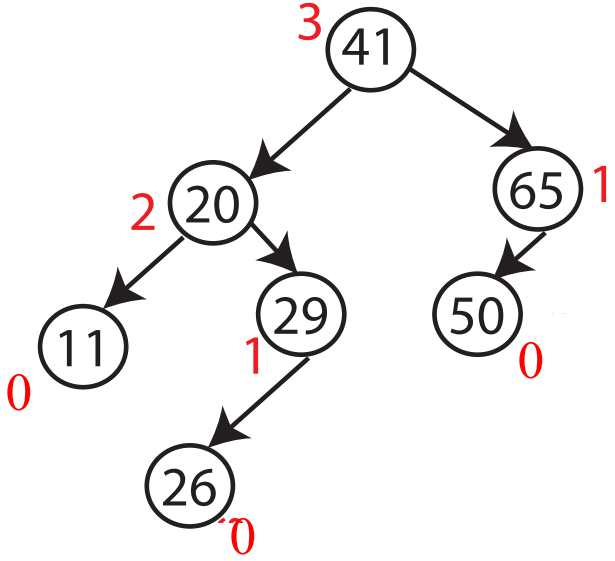
Balanced BST Strategy

- Augment every node with useful INFO
- Define a local invariant on INFO
- Show that invariant guarantees $\Theta(\log n)$ height
- Design algorithms to maintain INFO & invariant

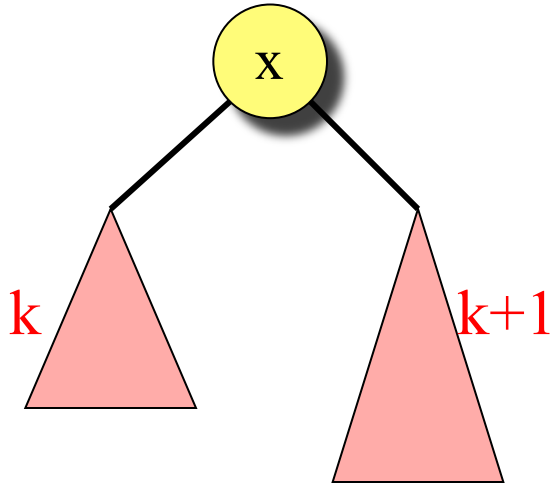
AVL Trees: Definition

[**A**delson-**V**elskii and **L**andis'62]

- **INFO**: for every node, store its height ("augmentation")
  - Leaves have height 0
  - NIL has "height" -1



- **Invariant**: for every node x, the heights of its left child and right child differ by at most 1



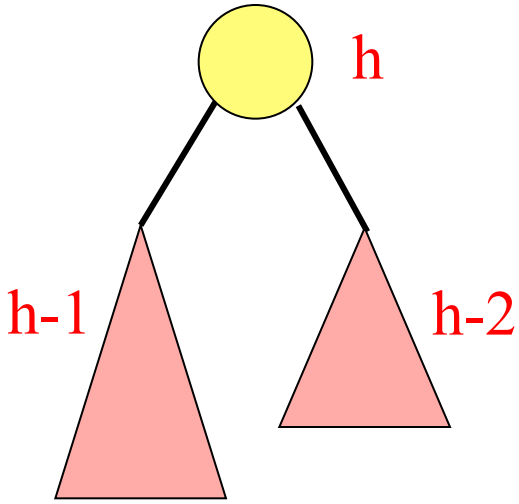
AVL trees have height $\Theta(\log n)$

- Let n_h be the minimum number of nodes of an AVL tree of height h
- We have $n_h \geq 1 + n_{h-1} + n_{h-2}$

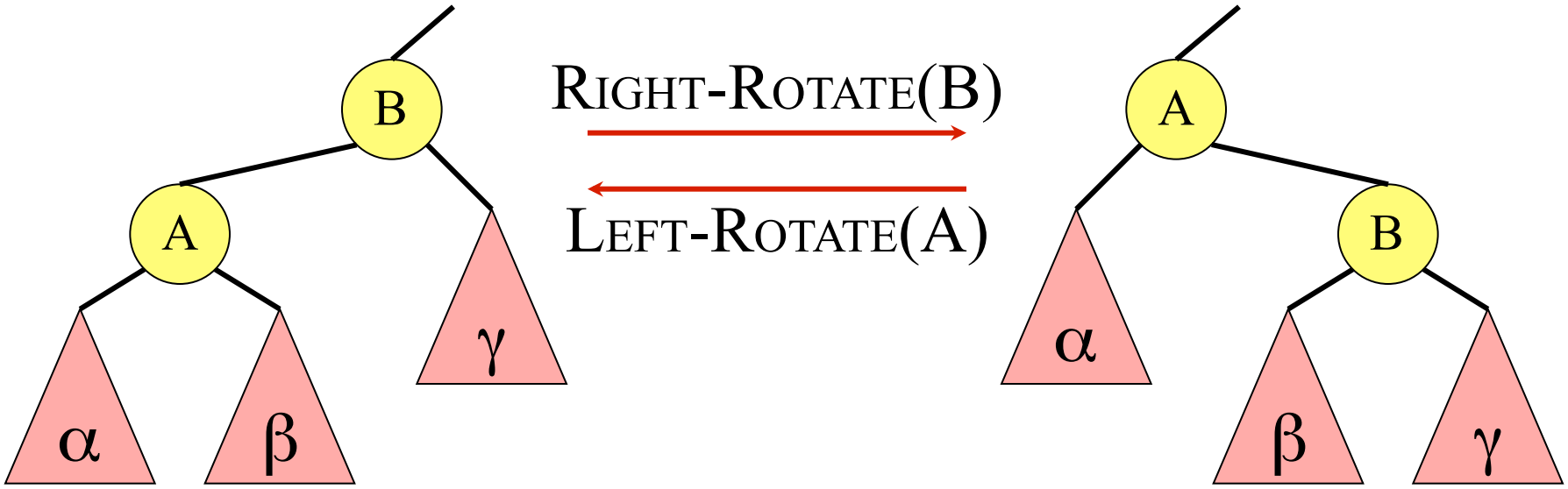
  $\Rightarrow n_h > 2n_{h-2}$

  $\Rightarrow n_h > 2^{h/2}$

  $\Rightarrow h < 2 \lg n_h$
- Optimal?

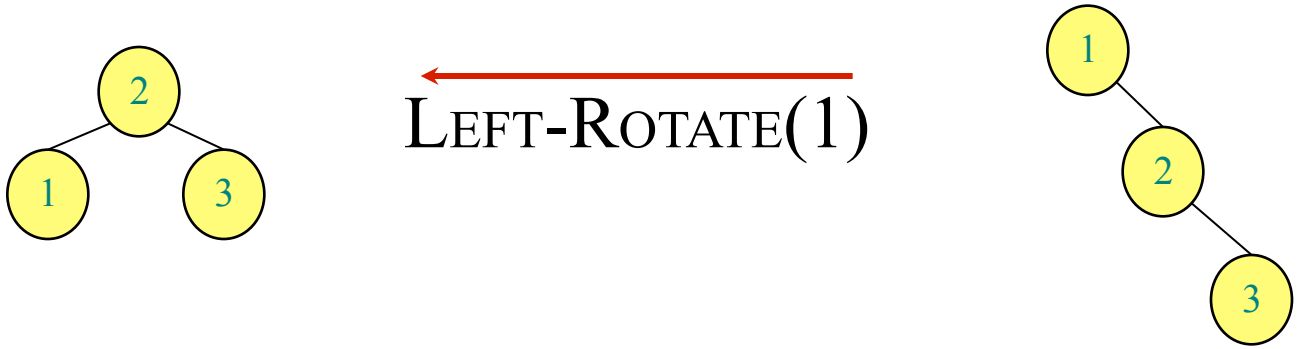


Rotations



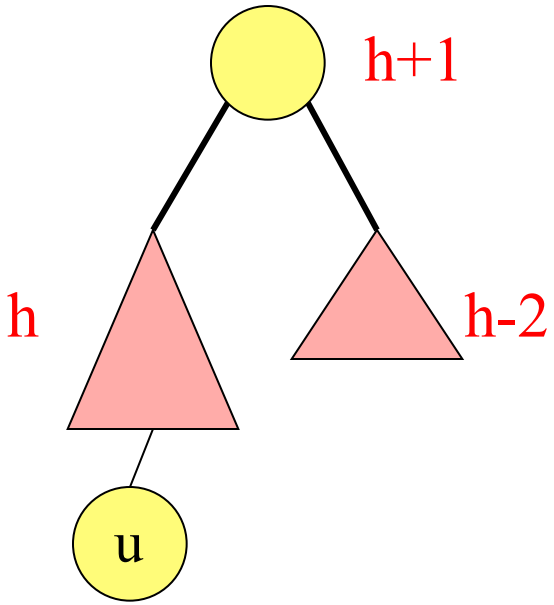
Rotations maintain the inorder ordering of keys:

- $\forall a \in \alpha \quad \forall b \in \beta \quad \forall c \in \gamma$: $a \le A \le b \le B \le c$.



Insertions

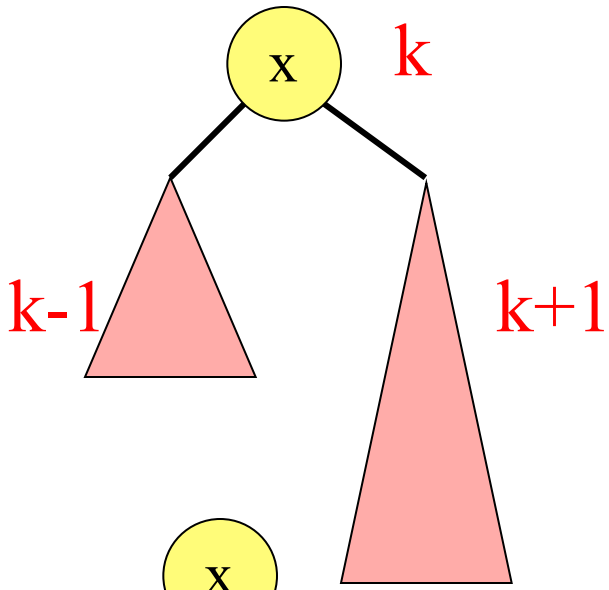
- Insert new node u as in the simple BST
  - Can create imbalance
- Work your way up the tree, restoring the balance

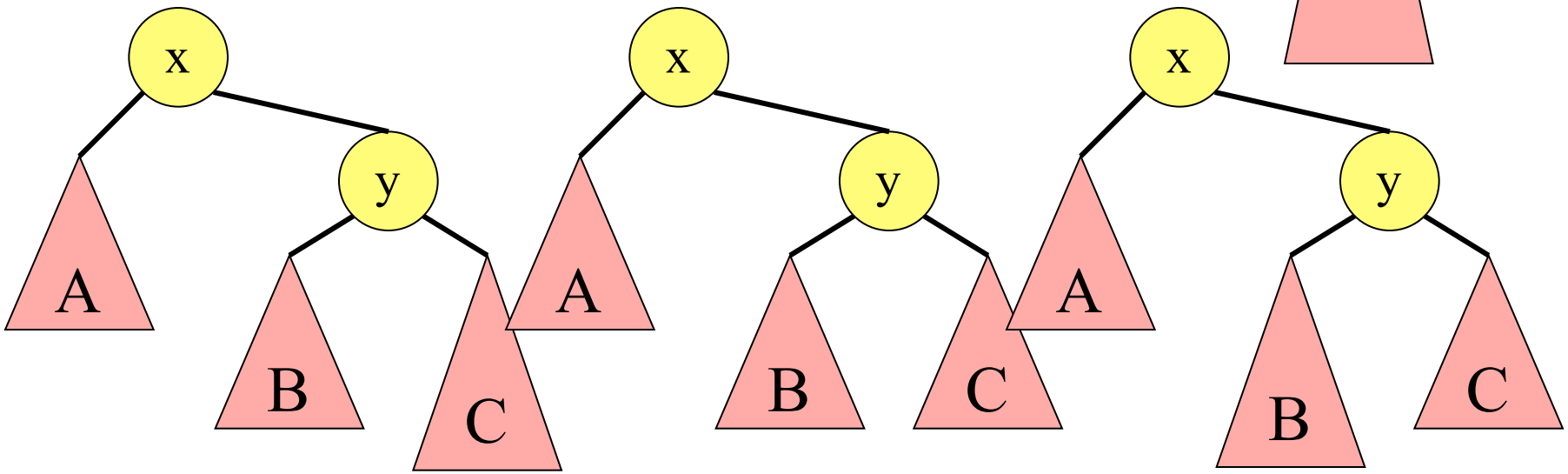


Balancing

- Let x be the lowest "violating" node
- WLOG x is "right-heavy":

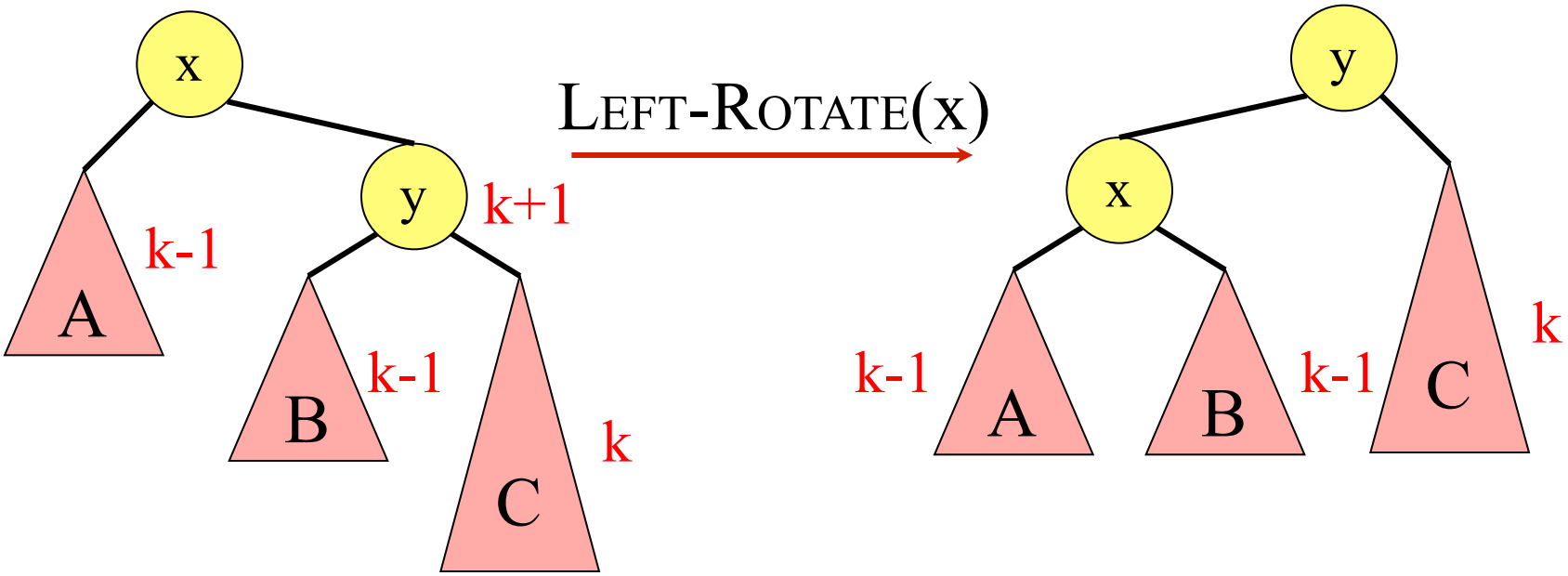
  Right[x] deeper Left[x]
- 3 Cases (others are symmetric):



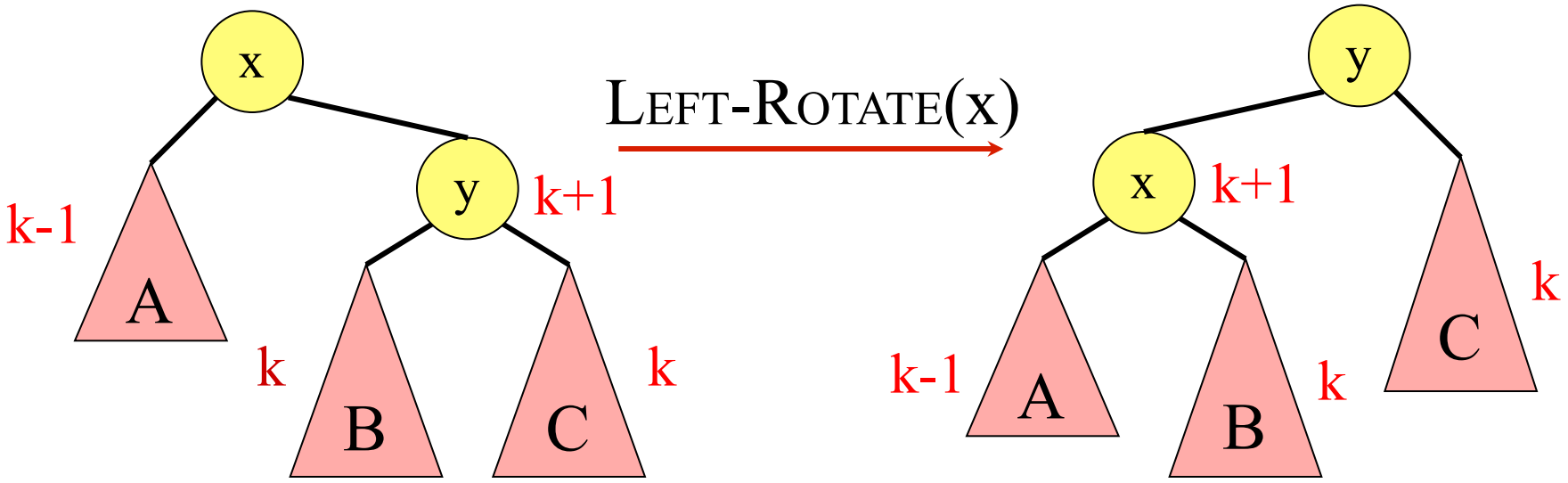


1. y right-heavy 2. y balanced 3. y left-heavy

Case 1: y is right-heavy

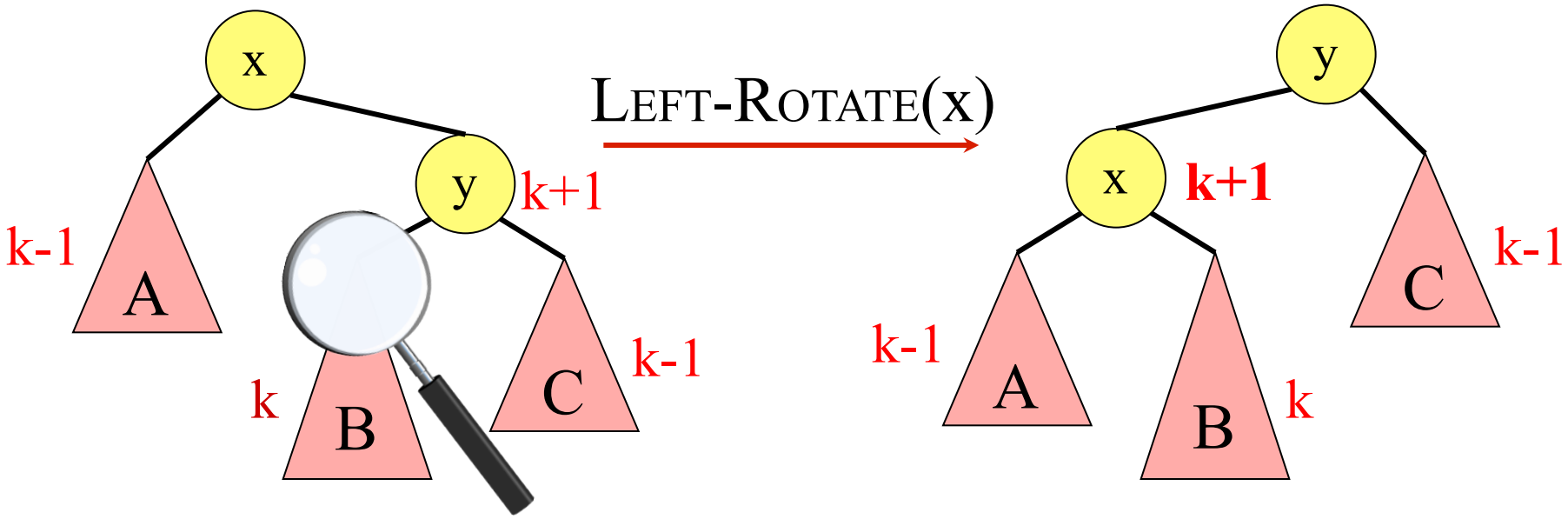


Case 2: y is balanced



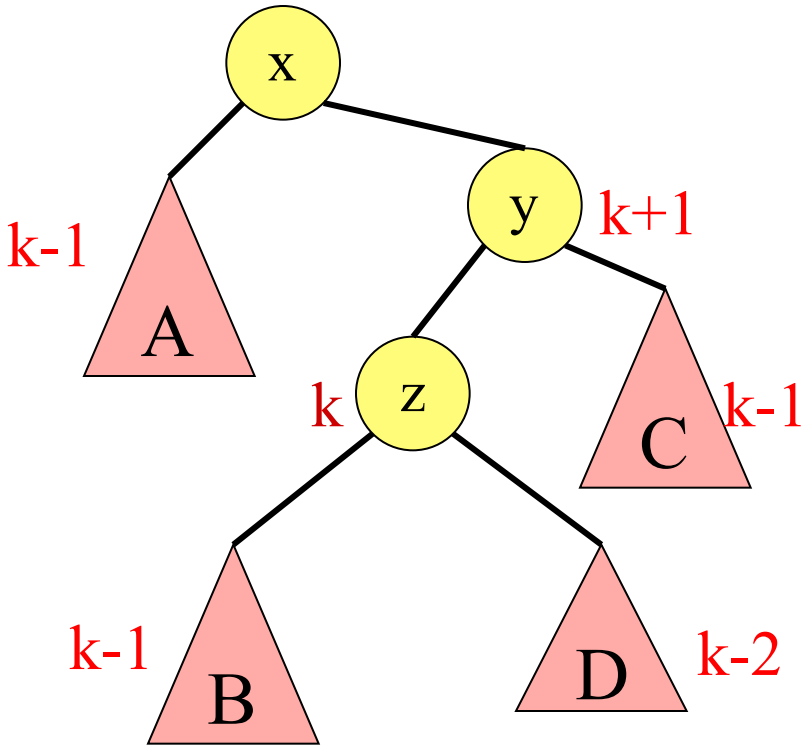
(Same as Case 1)

Case 3: y is left-heavy

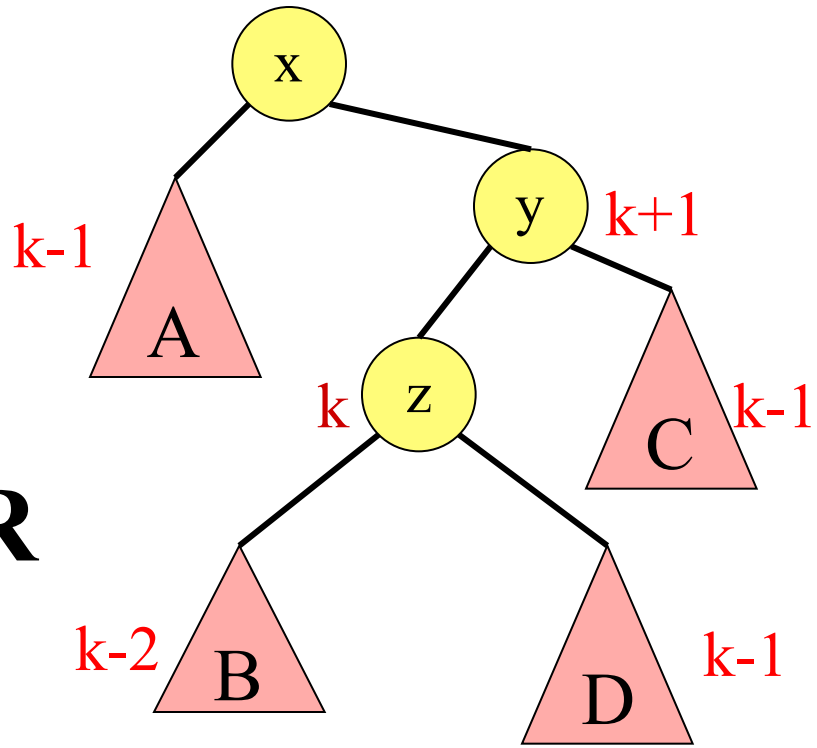


Need to do more …

Case 3: y is left-heavy



OR

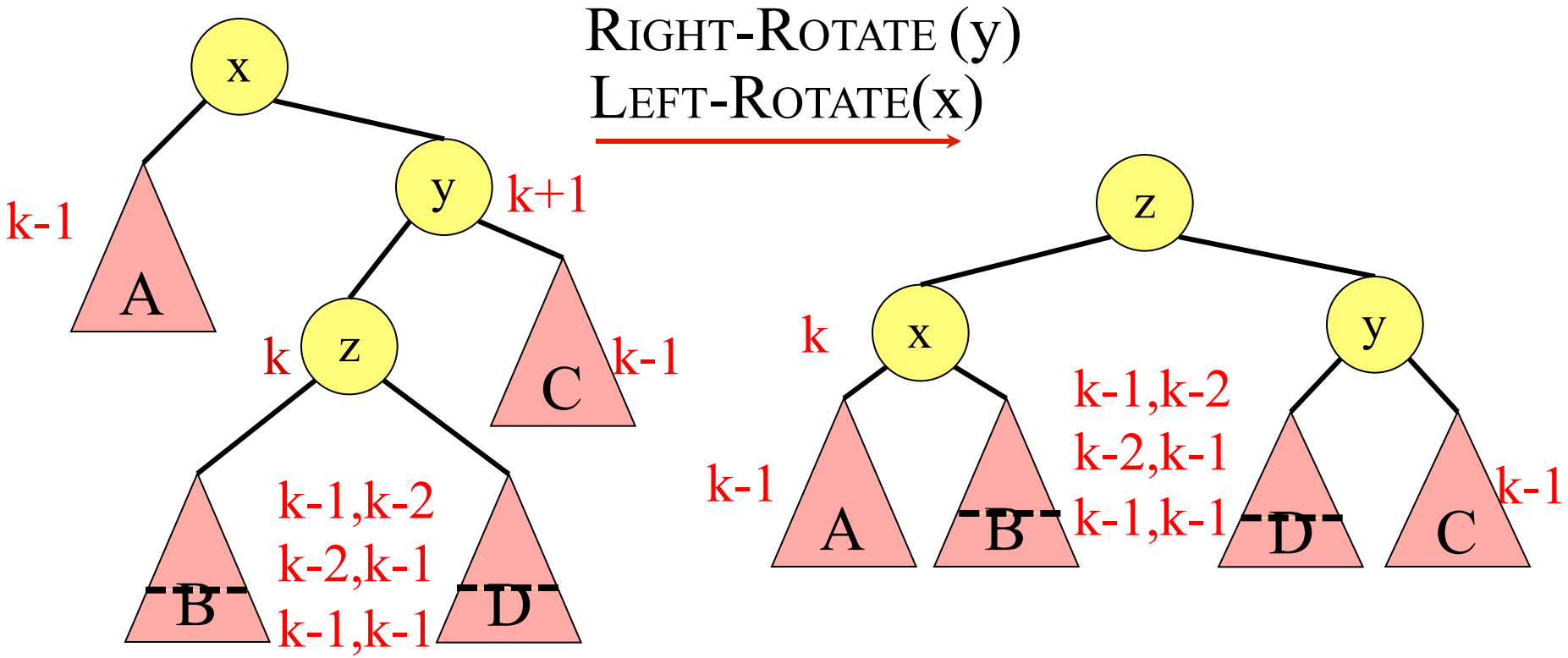


& z is left-heavy

& z is right-heavy

OR …

Case 3: y is left-heavy



And we are done!

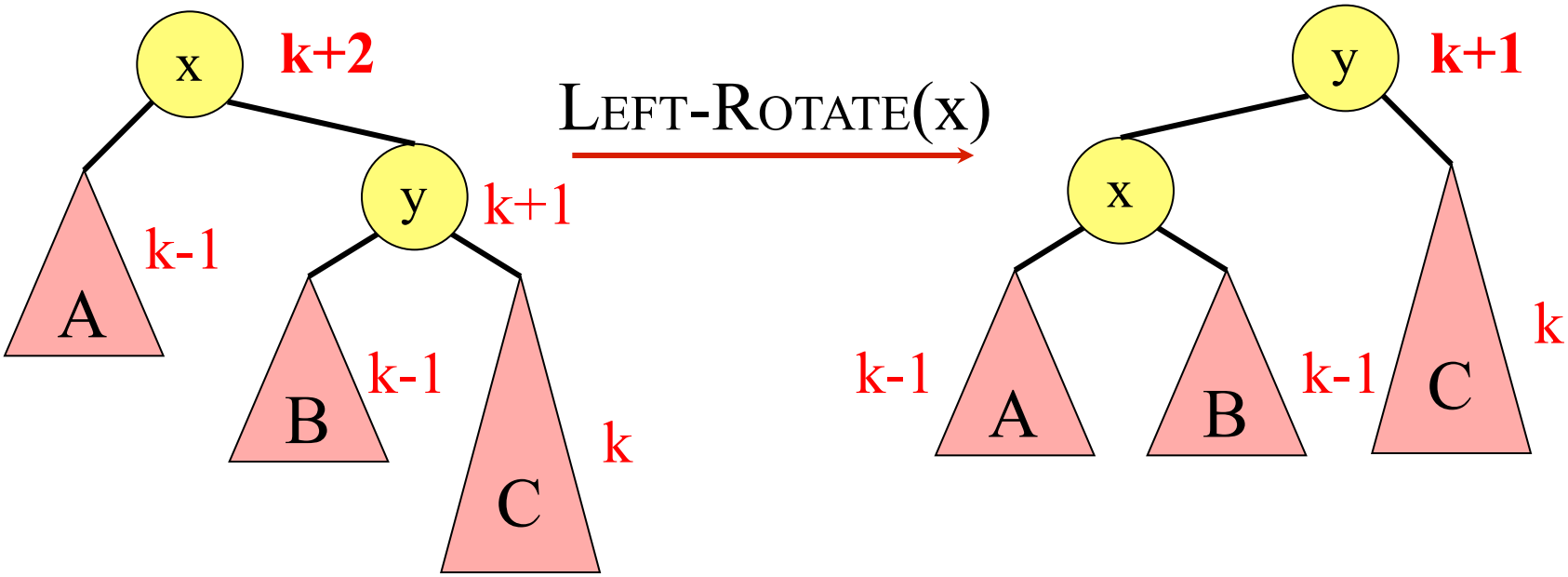
Complexity

Insertion:

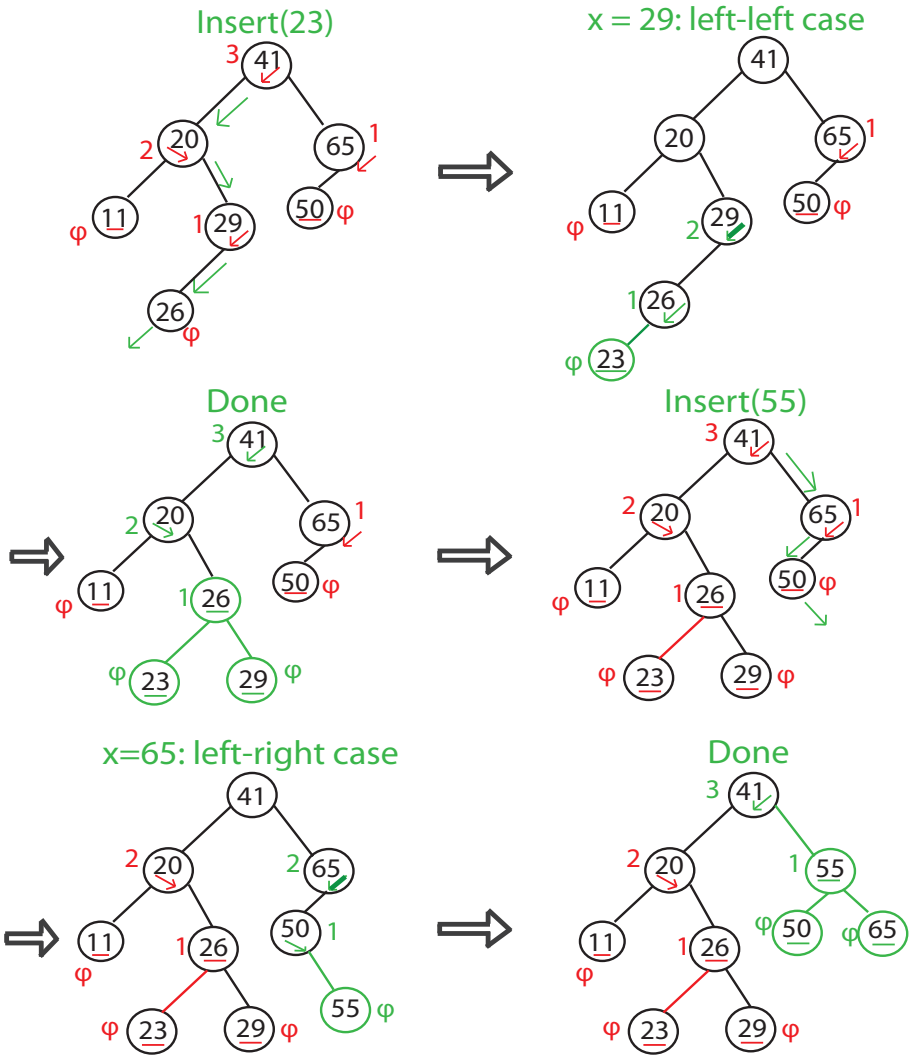
Local rebalancing:

How many local rebalancings after one insertion?

Recall Case 1: y is right-heavy



Examples of insert/balancing



Balanced Search Trees …

- AVL trees (Adelson-Velsii and Landis 1962)
- Red-black trees (see CLRS 13)
- Splay trees (Sleator and Tarjan 1985)
- Scapegoat trees (Galperin and Rivest 1993)
- Treaps (Seidel and Aragon 1996)
- ….

...

US capitol Mississippi Arkansas ?

Who invented the Multiplication Algorithm?