

Splay Trees

CSE 373

Data Structures

Readings

- Reading Chapter 10
 - › Section 10.3

Self adjusting Trees

- Ordinary binary search trees have no balance conditions
 - › what you get from insertion order is it
- Balanced trees like AVL trees enforce a balance condition when nodes change
 - › tree is always balanced after an insert or delete
- Self-adjusting trees get reorganized over time as nodes are accessed
 - › Tree adjusts after insert, delete, or find

Splay Trees

- Splay trees are tree structures that:
 - › Are not perfectly balanced all the time
 - › Data most recently accessed is near the root. (principle of locality; 80-20 “rule”)
- The procedure:
 - › After node X is accessed, perform “splaying” operations to bring X to the root of the tree.
 - › Do this in a way that leaves the tree more balanced as a whole

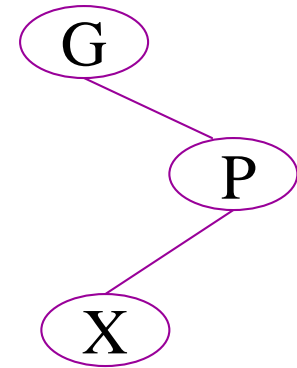
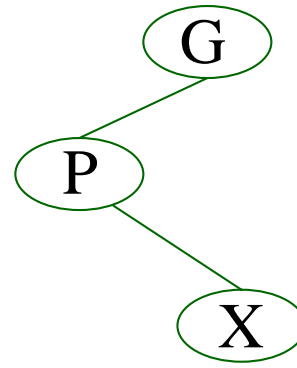
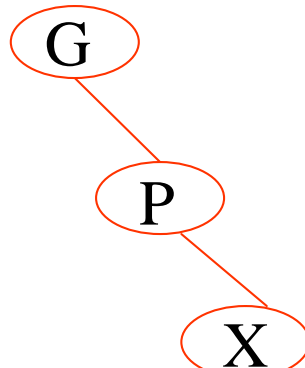
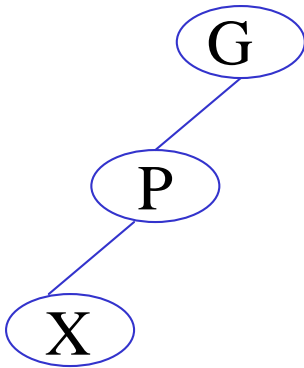
Splay Trees (1985)

- Daniel Sleator (1954 -) & Robert Tarjan (1948 -)



Splay Tree Terminology

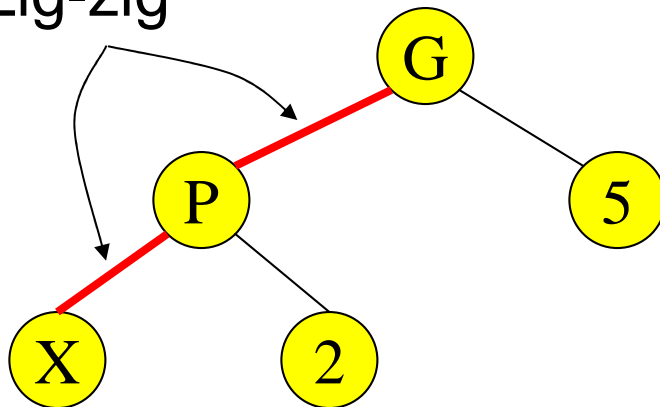
- Let X be a non-root node with ≥ 2 ancestors.
 - P is its parent node.
 - G is its grandparent node.



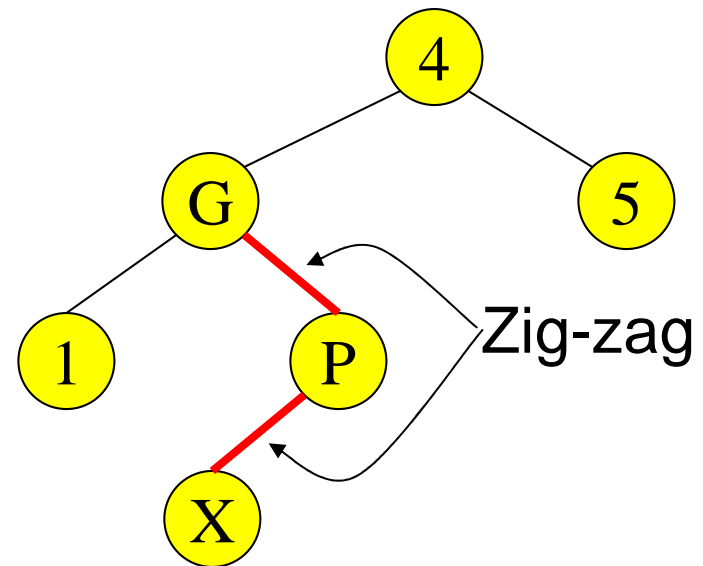
Zig-Zig and Zig-Zag

Parent and grandparent
in same direction.

Zig-zig

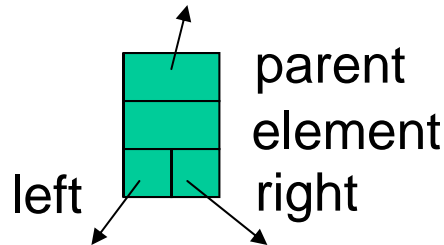


Parent and grandparent
in different directions.



Splay Tree Operations

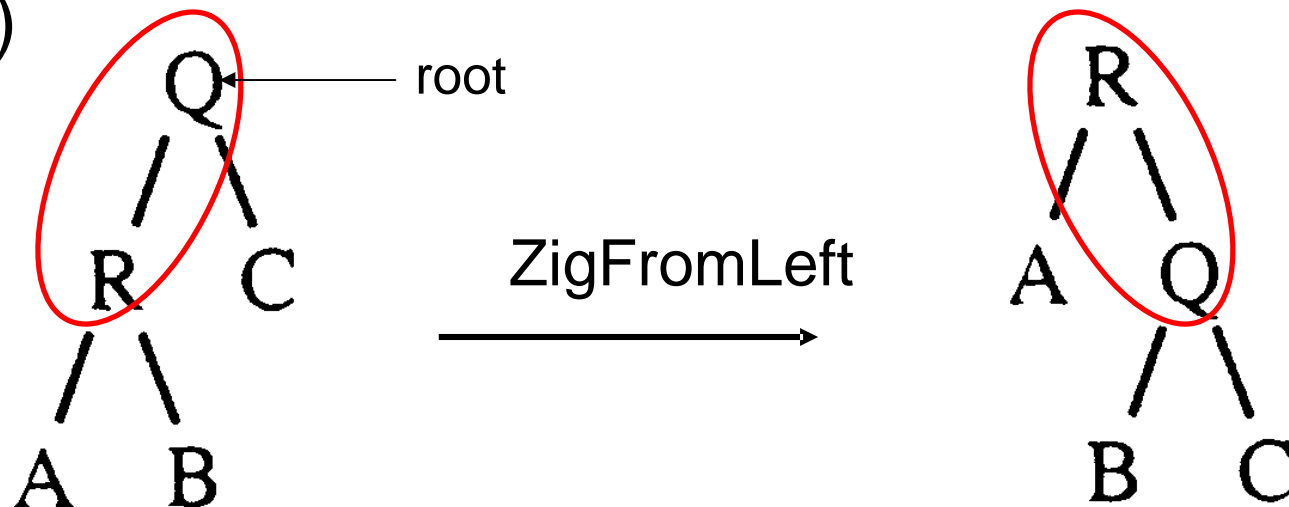
1. Helpful if nodes contain a **parent** pointer.



2. When X is accessed, apply one of **six** rotation routines.
 - Single Rotations (X has a P (the root) but no G)
ZigFromLeft, ZigFromRight
 - Double Rotations (X has both a P and a G)
ZigZigFromLeft, ZigZigFromRight
ZigZagFromLeft, ZigZagFromRight

Zig at depth 1 (root)

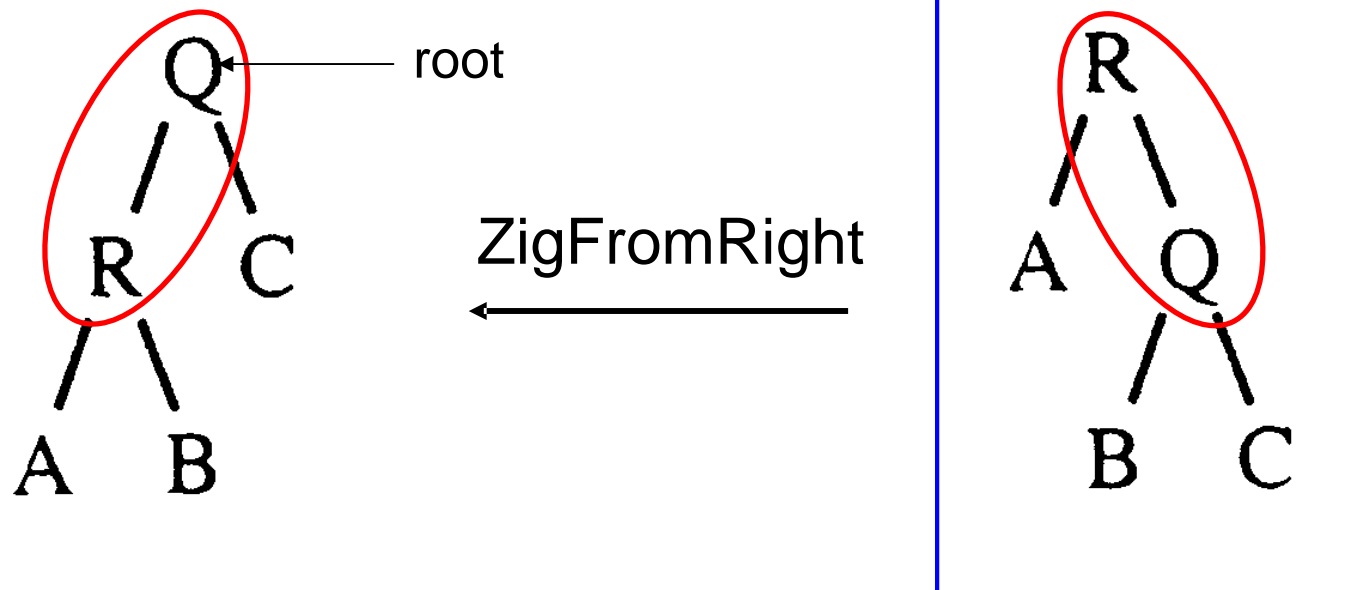
- “Zig” is just a **single rotation**, as in an AVL tree
- Let R be the node that was accessed (e.g. using Find)



- ZigFromLeft moves R to the top → faster access next time

Zig at depth 1

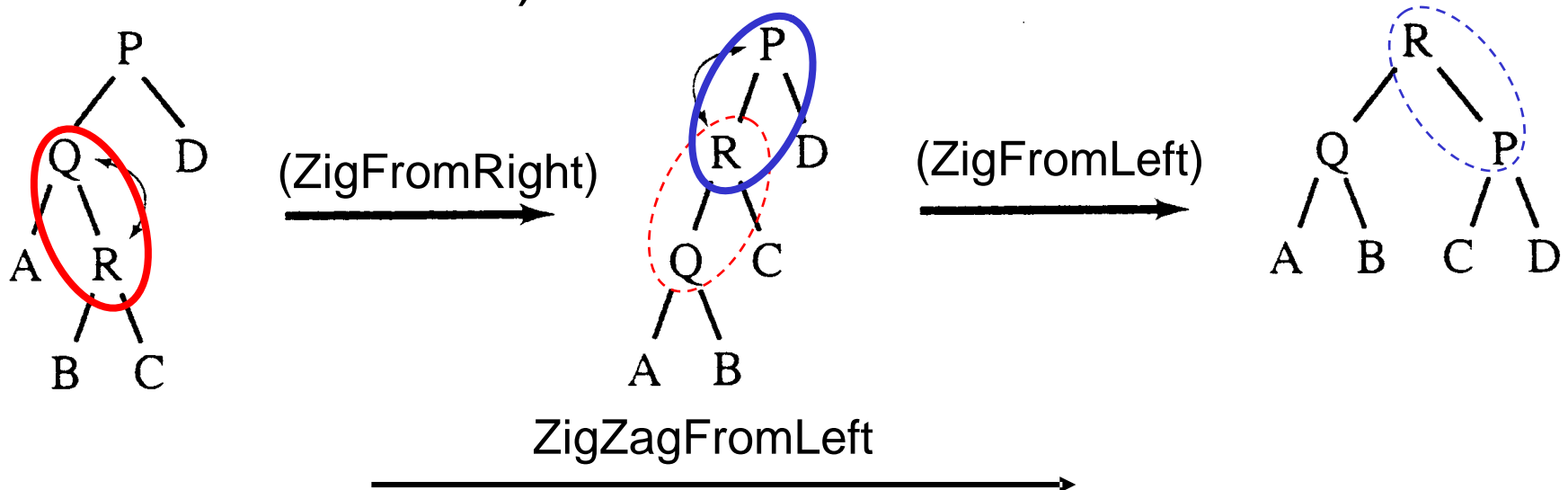
- Suppose Q is now accessed using Find



- ZigFromRight moves Q back to the top

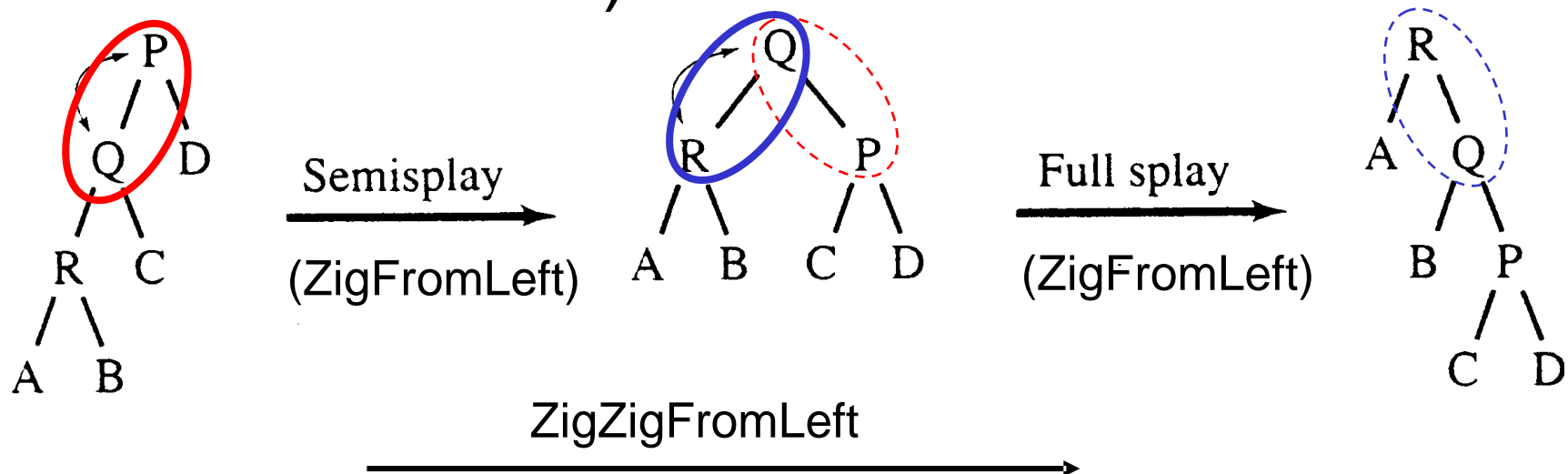
Zig-Zag operation

- “Zig-Zag” consists of **two rotations of the opposite direction** (assume R is the node that was accessed)

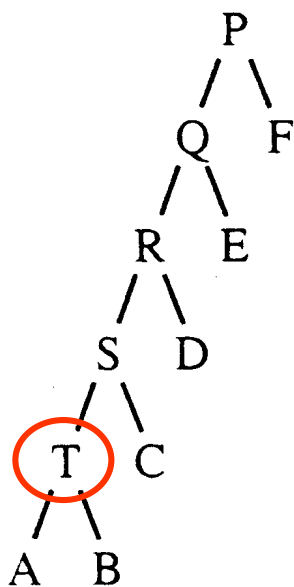


Zig-Zig operation

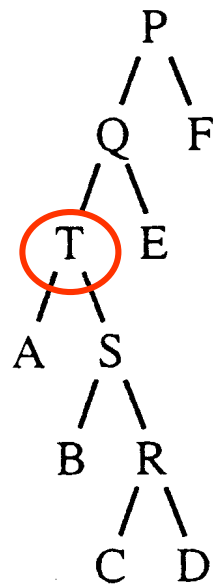
- “Zig-Zig” consists of **two single rotations of the same direction** (R is the node that was accessed)



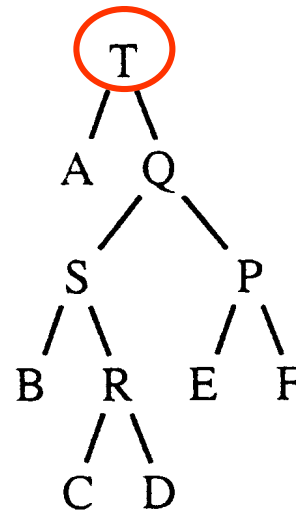
Decreasing depth - "autobalance"



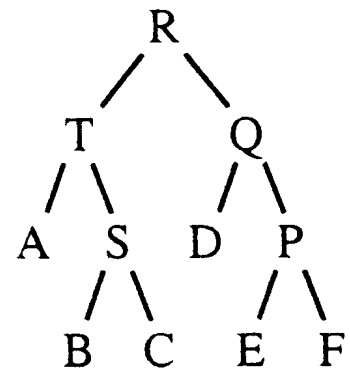
(a)



(b)



(c)



(d)

Find(T)



Find(R)

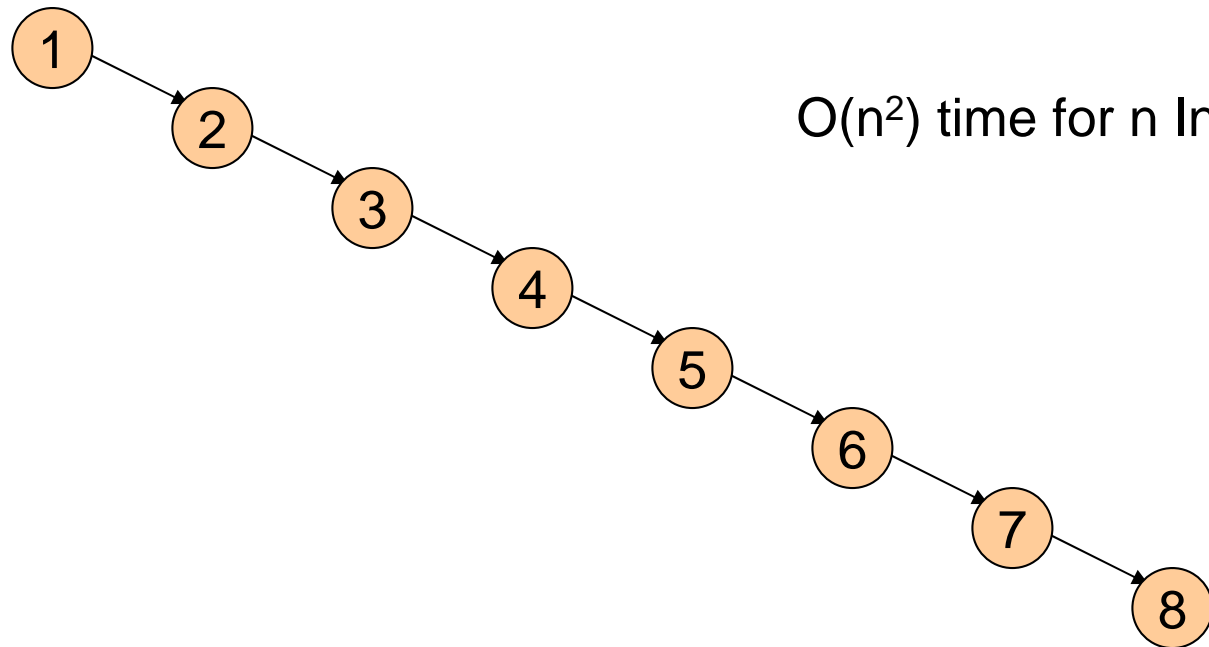


Splay Tree Insert and Delete

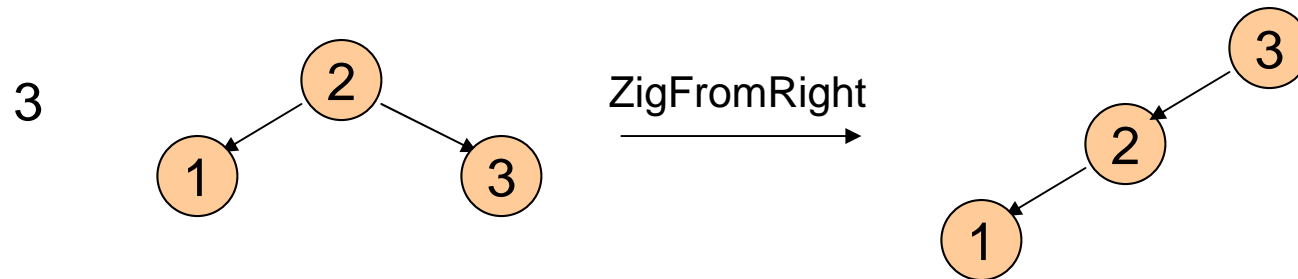
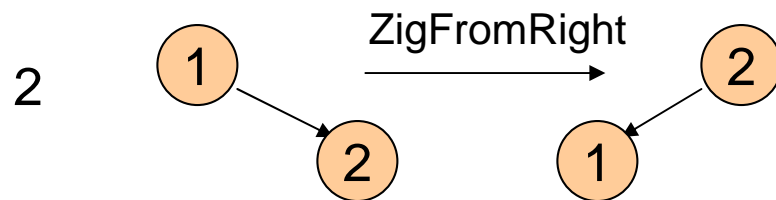
- Insert x
 - › Insert x as normal then splay x to root.
- Delete x (there are several options)
 - › “Delete” x as in a BST . This yields a node y that is really disappearing
 - › Splay y ’s parent to the root

Example Insert

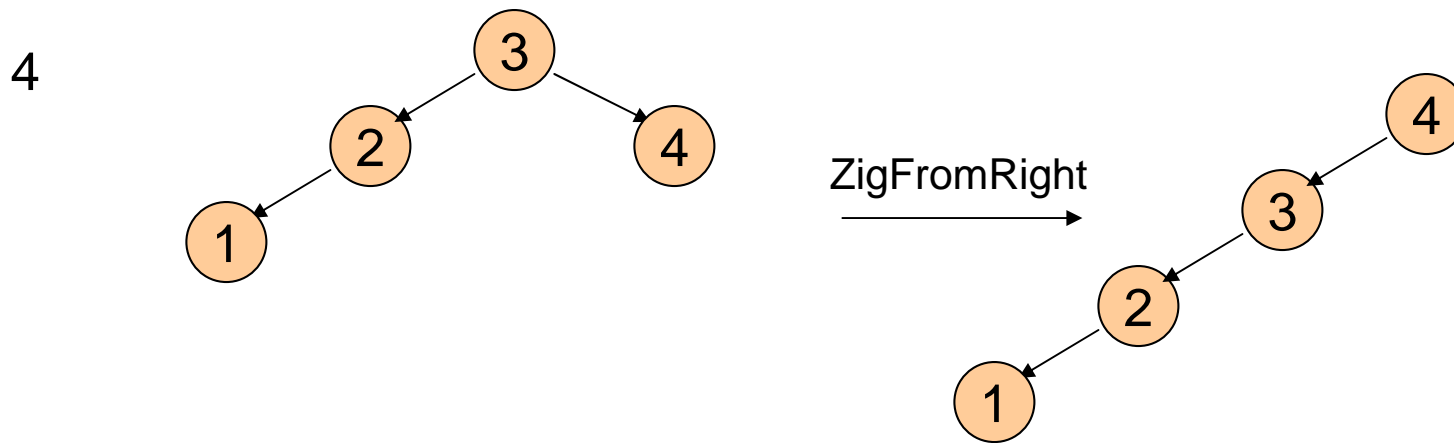
- Inserting in order 1,2,3,...,8
- Without self-adjustment



With Self-Adjustment

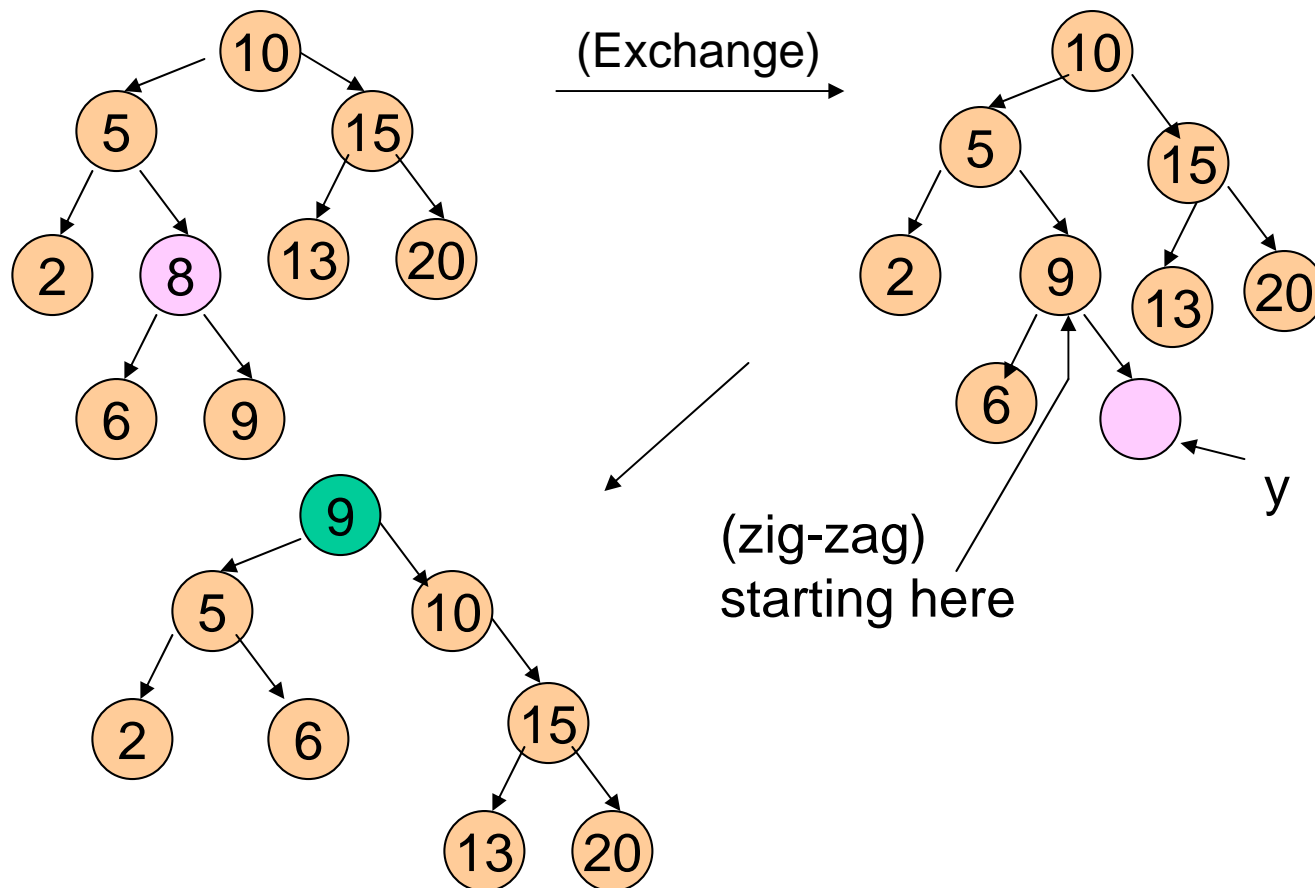


With Self-Adjustment



Each Insert takes $O(1)$ time therefore $O(n)$ time for n Insert!!

Example Deletion



Analysis of Splay Trees

- Splay trees tend to be balanced
 - › M operations takes time $O(M \log N)$ for $M \geq N$ operations on N items. (proof is difficult)
 - › Amortized $O(\log n)$ time.
- Splay trees have good “locality” properties
 - › Recently accessed items are near the root of the tree.
 - › Items near an accessed one are pulled toward the root.

This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.