Sorting IV: AVL Trees and AVL Sort

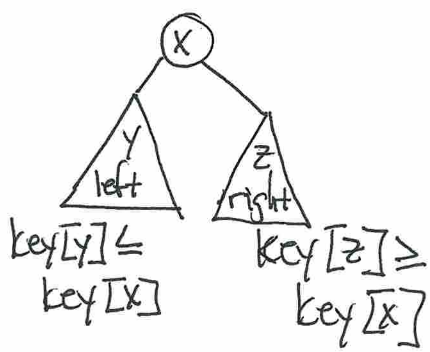
Reading: CLRS 13.1-2, Chpt 14

Admin:
- Final Exam, Monday 16 May 2016, 1:30-4:30 PM, W35
- Cookie Challenge, PS 2 Part B submitted 48 hrs early

Last Time: Binary Search Trees

BST Property

For any node x
- all nodes y in left subtree key[y] ≤ key[x]
- all nodes z in right subtree key[z] ≥ key[x]



Operations

insert/delete, find-min/find-max, sucessor/predecessor, find all are $O(h)$ for any particular case and $\Theta(h)$ worst case

Inorder-Tree-Walk $\Theta(n)$

BST Sort

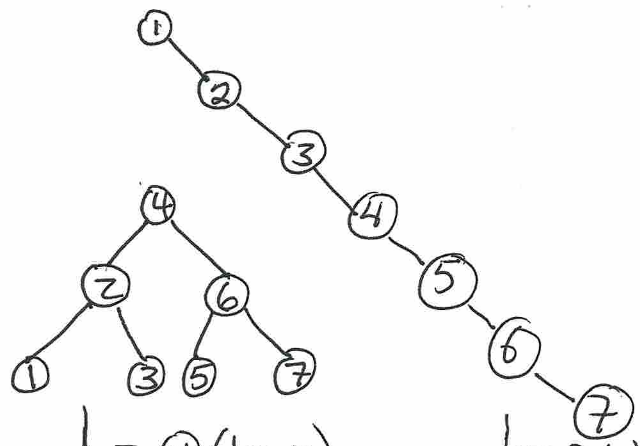
Create-BST-from-input
Inorder-Tree-Walk(root[Tree])

last time	today
$\Theta(n^2)$	$\Theta(n \log n)$
$+\ \Theta(n)$	$+\ \Theta(n)$
$\Theta(n^2)$	$\Theta(n \log n)$

TODAY

The "problem" with BSTs is the relationship between h (height) and n (# of nodes).

Worst Case $h \approx n$



$h = \Theta(n)$

Improvement:
Somehow balance trees so $h = \Theta(\log n)$

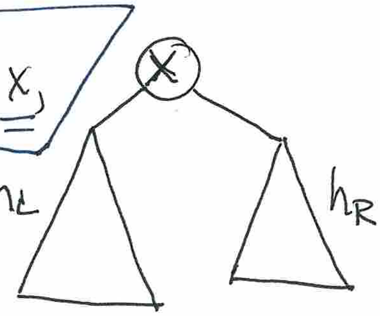
$h = \Theta(\log n)$

Need to do this in a way that balance can be maintained in $O(\log n)$ time. $\longrightarrow$ Then operations become $\Theta(\log n)$

Many possibilities – we will discuss AVL Trees as BSTs with extra condition

AVL Trees (Adelson–Velskii and Landis, 1962)

Invariant: For every node x, the heights of its left child and right child differ by at most 1.

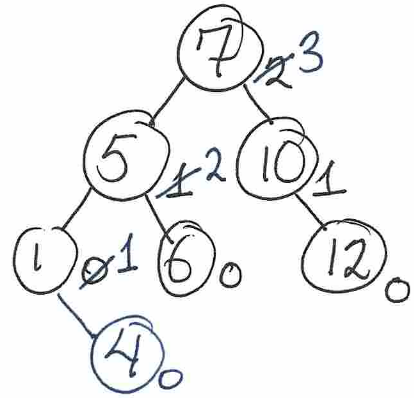


$|h_L - h_R| \le 1$

How to keep track of heights?

Augment the data structure for every node with its height

- Null has height -1
- Leaves have height 0
- Node has height equal to # of edges to "deepest descendant"



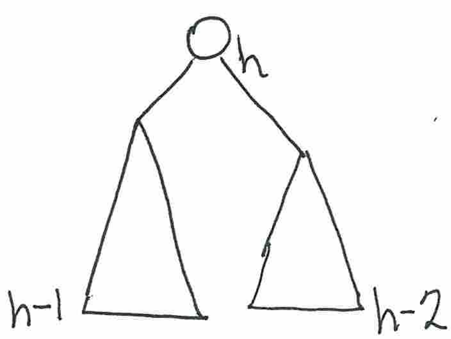
Prove: AVL trees have height $\Theta(\log n)$

"Most compact" tree: Complete binary tree of height h ← Is AVL tree

has $n = 2^{h+1} - 1$, so $n \le 2^{h+1} - 1$

and $h \ge \log_2(n+1) - 1$, so $h = \Omega(\log n)$

"Least compact" tree: n_h = minimum # nodes in AVL tree of height h



$$n_h \ge 1 + n_{h-1} + n_{h-2} > 2n_{h-2}$$

$$n_h > 2^{h/2} \rightarrow h < 2\log_2(n_h)$$

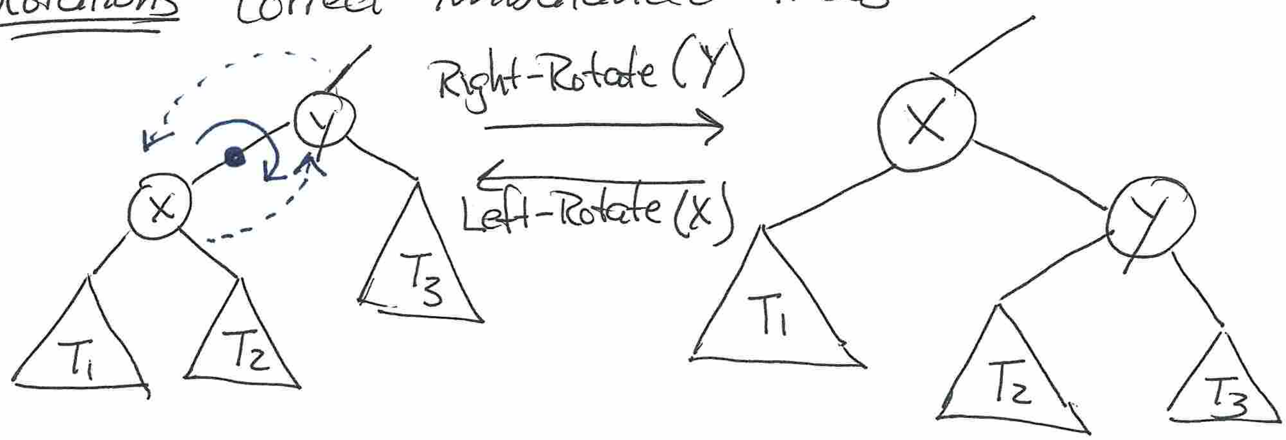
$$h = O(\log n)$$

$$\boxed{\therefore h = \Theta(\log n)}$$

But how can the invariant be maintained? (in log n time)

First, note that changes to tree structure (insert/delete) only change heights of nodes on direct route between site of change and root.

Rotations correct imbalanced trees



Right-Rotate

- X rises and Y drops (reversing their parent-child relationship)
- X would have 3 children ⇒ left[Y] ← Root[T2]
- Y would have 2 parents ⇒ p[X] ← Previous p[Y]

Proper Key Order Maintained

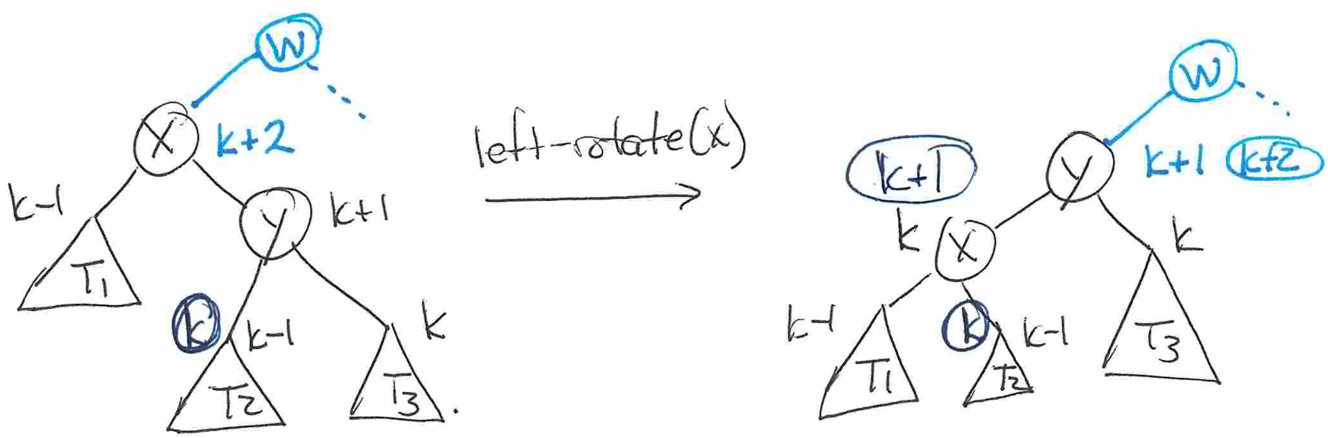
$$keys[T_1] \le key[X] \le keys[T_2] \le key[Y] \le keys[T_3]$$

Re-balancing

- Let x be lowest violating node — fix subtree and move up
- Assume X is "right heavy" (x's right child is deeper than left)
- Exist 3 cases
  1) X's right child y is right-heavy
  2) y is balanced
  3) y is left-heavy

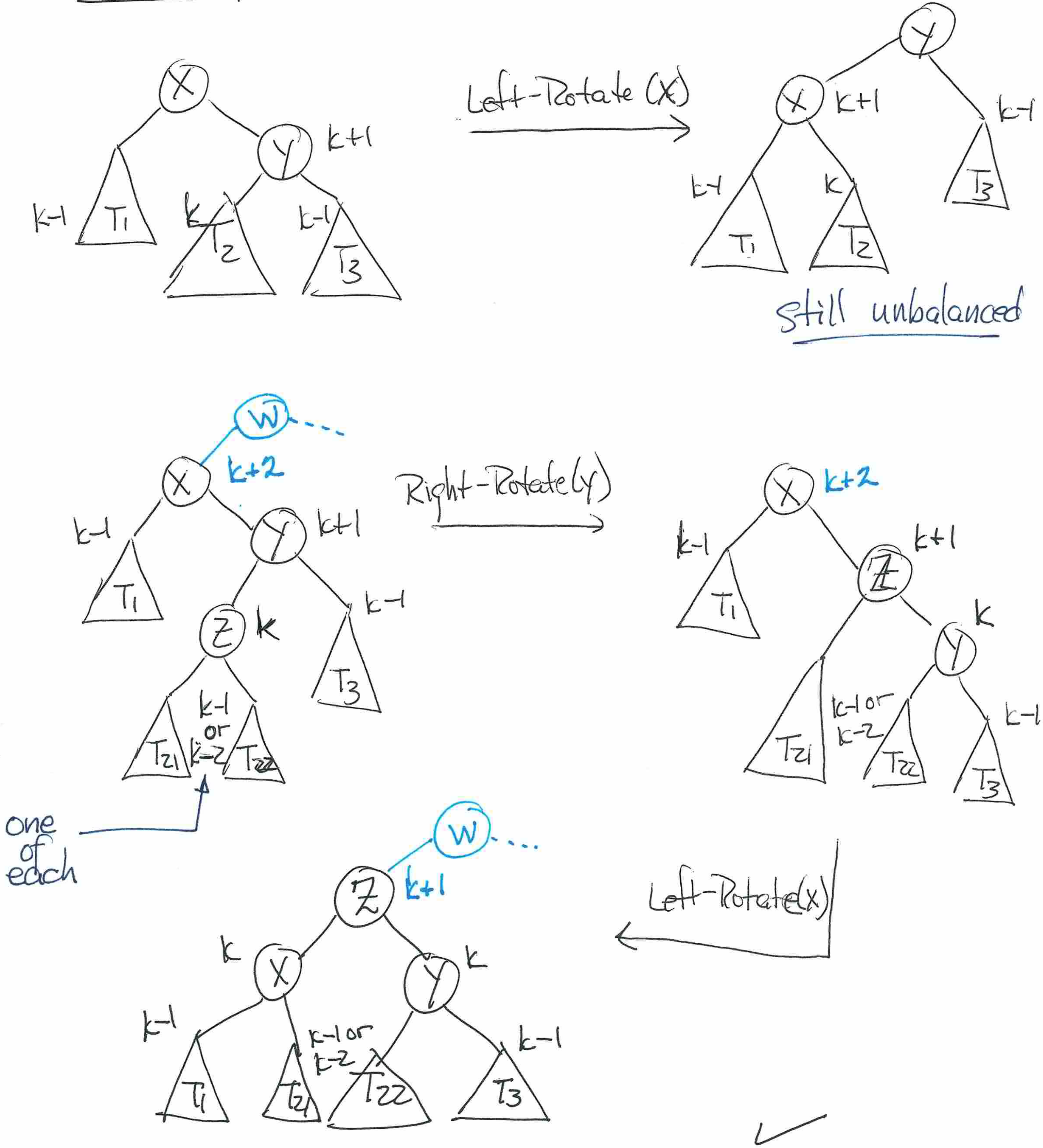
Case 1

Case 2



same solution as case 1

Case 3: Y is left-heavy



Operations can be handled $\Theta(\log n)$ on AVL trees because balanced BSTs can be maintained $\Theta(\log n)$

Insertion and Deletion can be carried out in the ordinary BST way, and imbalances created can then be corrected working up the tree toward the root