

# CSC 172– Data Structures and Algorithms

Lecture #21

Spring 2018

Please put away all electronic devices



# Student Feedback

- Workshop Problems vs. Problems on Quiz
  - As you have noticed, for the last couple of quizzes most of the problems are similar to those you have done in workshop.
  - As we will cover less coding problems, this will be the case until the end of the semester.
  - But, don't forget Labs are also part of the quizzes.

Note: Currently, we use Workshops for concepts and Labs for coding

Please let me know whether you are interested in doing more coding problems during workshops or you prefer to continue working on the concepts.

# Student Feedback

- Slides before Lecture:
  - Most of the time, slides are uploaded **within 30 minutes** of the lecture.
  - Also, slides have many ‘surprise’ elements.
  - I understand that it might be useful for a few student to review the slides beforehand. But we encourage students to take note on paper and discourage using electronic devices in class.
  - Still, if you want to access the slides beforehand, I would suggest visiting <http://www.cs.rochester.edu/courses/172/fall2017/>

That being said, I will make more conscious effort to post the slides ‘just before’ lectures.

# Scheduling an appointment

- Feel free to come during office hours
  - No Appointment required
- Stop by my office anytime and if I am free I will meet
- Send me an email with your availability. Usually, Friday afternoon works best for me. But, please check with me before coming.

# Lecture Video

- Slides are not readable because of the light
  - Please interrupt me and let me know if that is the case.

# Extra Credit

- If you have any suggestion, let me know.
- Curving of course grade:
  - Grading based on Distribution shown in Lecture 1
  - I **may** curve the course (and/or particular components) if required
    - No negative curving
    - Median would be in the range of B (between B- and B+)

# (Self) Balanced Search Trees



# Variations of Balanced Binary Search Tree

- AVL
- Red-black
- 2-4
- Splay
- B+ Tree

# BSTs are Potentially Good

- In  $O(h)$ -time, where  $h$  is the height of the tree, we can perform
  - Search
  - Minimum, maximum
  - Predecessor, successor
  - Insert, Delete
- An  $n$ -node binary tree must have height  $h = \Omega(\log n)$ 
  - The best we can hope for is  $h = O(\log n)$

# Intuitively, How to Keep a Tree's Height Small?

- For every internal node  $v$ 
  - $|\text{left branch of } v| \approx |\text{right branch of } v|$
  - Exactly the same reason quick sort needs balanced partition
- *AVL trees* maintain this property by
  - Keeping the heights of left and the right subtrees roughly equal
- AVL is more rigid, faster search

Named after

- Georgy Adelson-Velsky and Evgenii Landis
- First self-balancing binary search tree
- 1962
- <http://professor.ufabc.edu.br/~jesus.mena/courses/mc3305-2q-2015/AED2-10-avl-paper.pdf>

Idea: rebalance the tree after an insert/delete

# AVL TREES

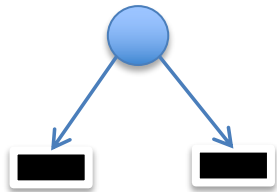
# AVL Trees

- *Balanced node:*
  - A node  $v$  is “balanced” if its left subtree and right subtree have heights differ by **at most 1**
- An **AVL tree** is
  - a BST in which every internal node is balanced.

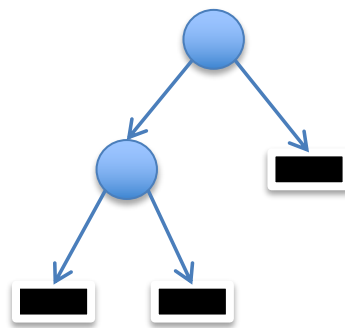
**Theorem:** an AVL tree on  $n$  nodes has  $O(\log n)$ -height

# Recurrence for $n(h)$

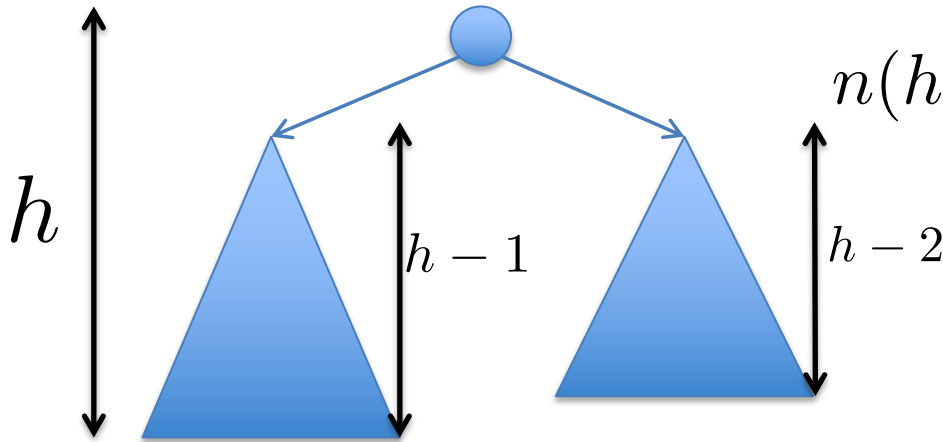
For convenience, heights measured to the NULLs



$$n(1) = 1$$



$$n(2) = 2$$



$$n(h) = 1 + n(h-1) + n(h-2)$$

# Old Friend: Fibonacci

$$n(h) = \frac{\phi^{h+2} - (-1/\phi)^{h+2}}{\sqrt{5}} - 1$$

$$\phi = \frac{1 + \sqrt{5}}{2} \approx 1.618$$

$$n(h) = \Omega(1.6^h) \Leftrightarrow h = O(\log n)$$

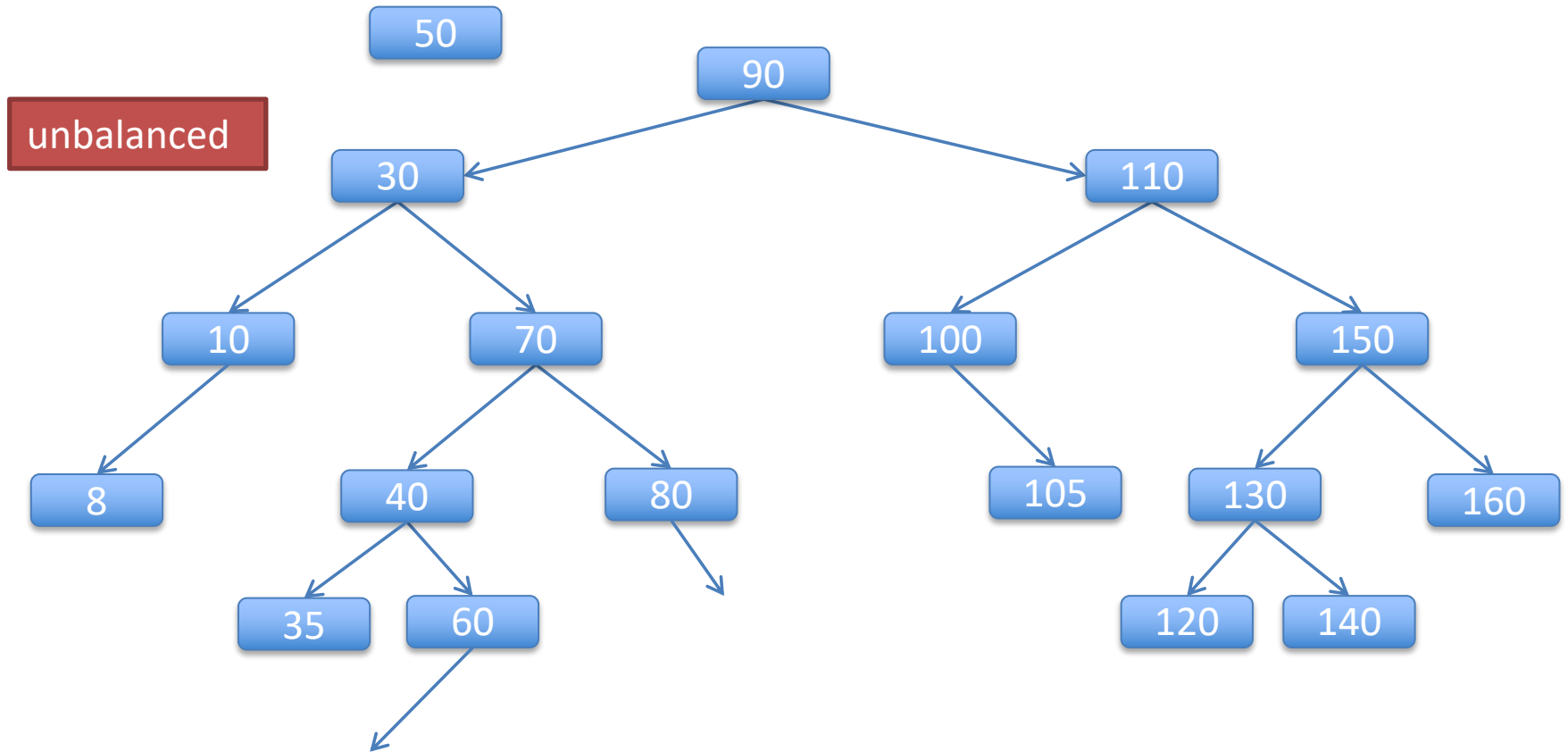
Not reqd. for Exam or Quiz

# But how do we maintain AVL property?

- After an insert
  - One subtree might be taller than the other by 2
  - Potentially affect the *balance* of all nodes up to the root
  - Rebalance
- After a delete
  - One subtree might be shorter than the other by 2
  - Potentially affect the *balance* of all nodes up to the root
  - Rebalance



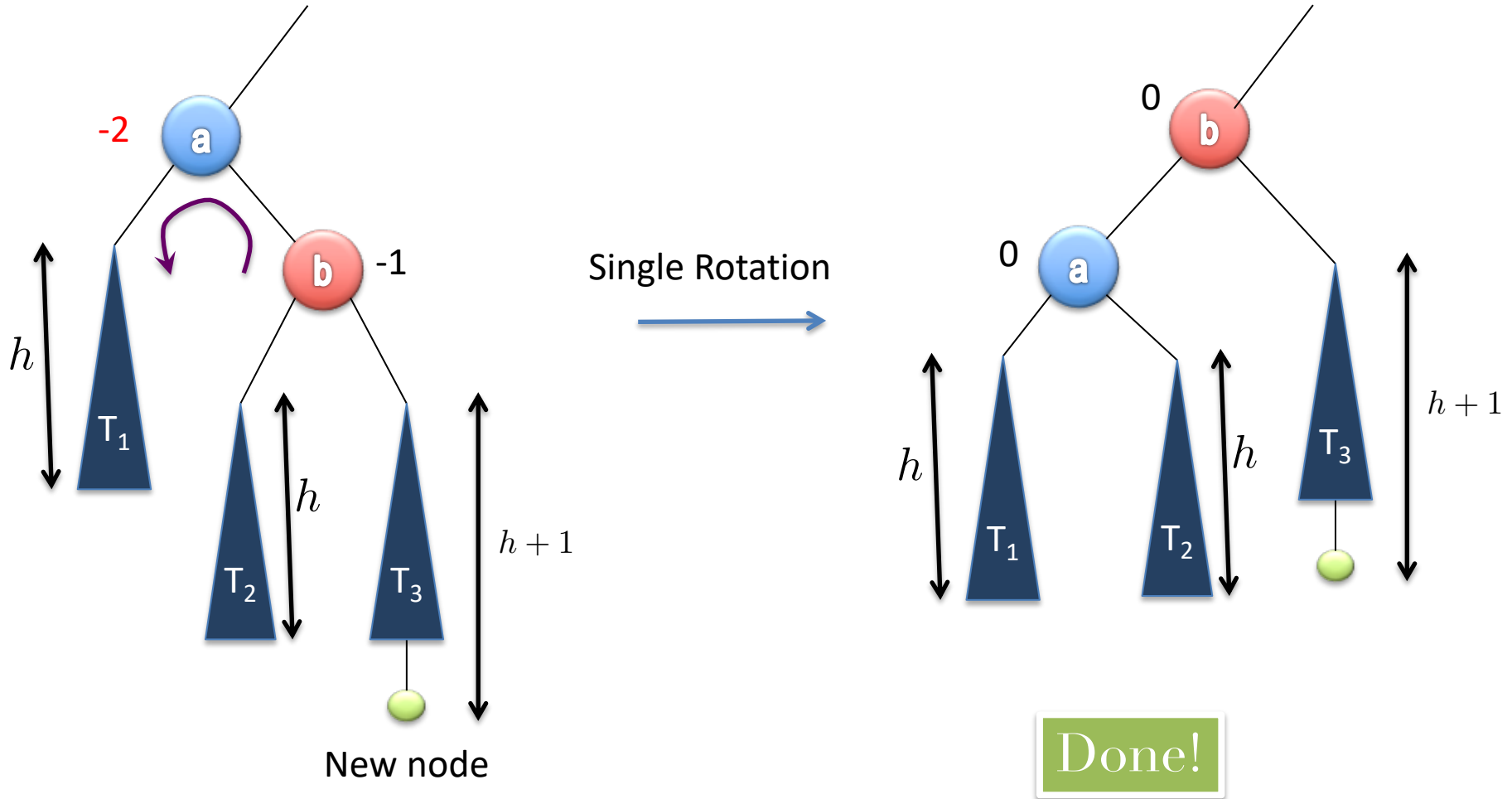
# Insert



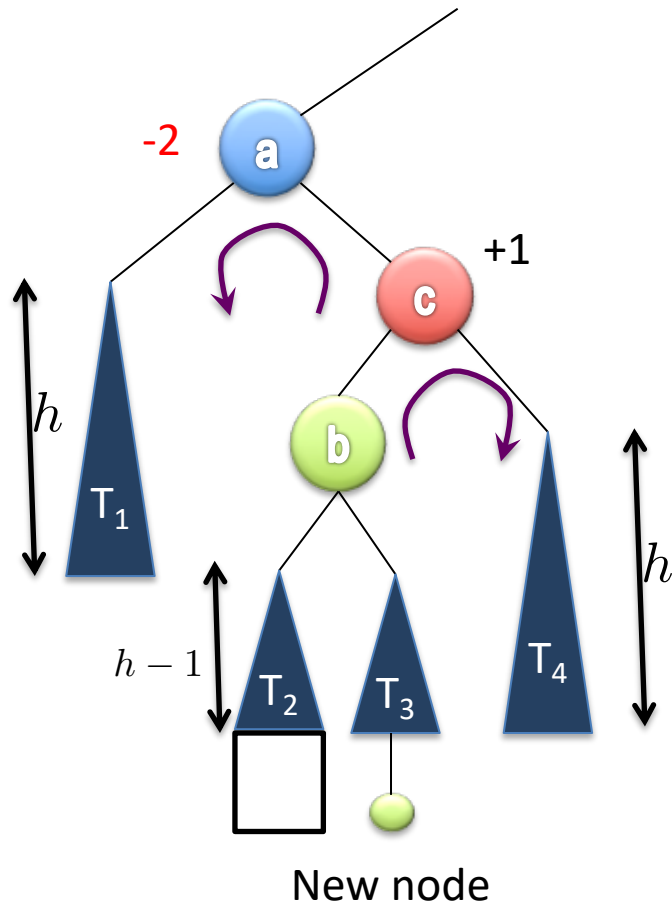
# Balance

- Let's define the “balanceness” of a node
  - $\text{Balance}(v) = \text{height}(v.\text{left}) - \text{height}(v.\text{right})$
- We want  $v$ 's balance to be in  $\{-1, 0, 1\}$ 
  - $\text{balance} = 1$  means  $v$  is “left heavy”
  - $\text{balance} = -1$  means  $v$  is “right heavy”
- After inserting a new node
  - Let  $a$  be the first node on the path back to the root that's not balanced, then  $a$ 's new balance is 2 or -2

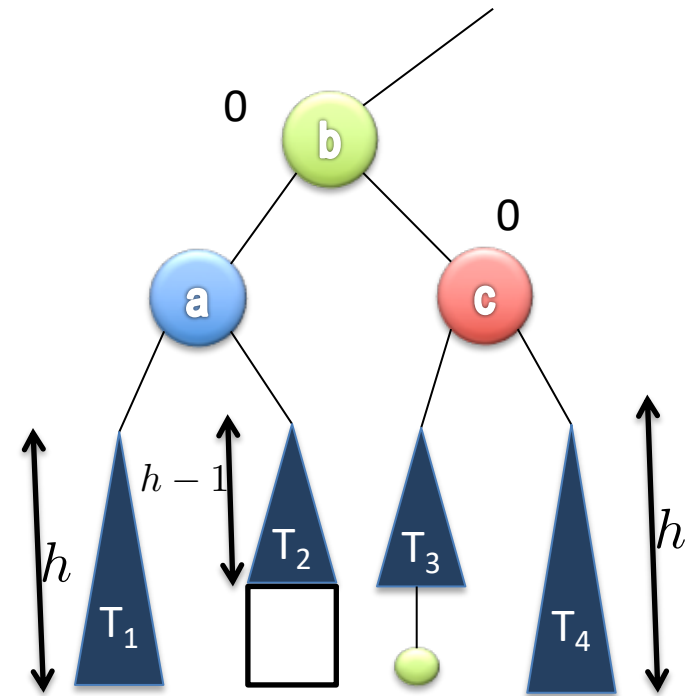
# Example: RR case



# Example: RL case



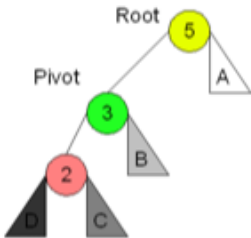
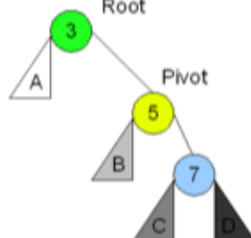
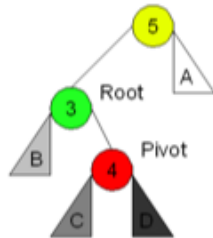
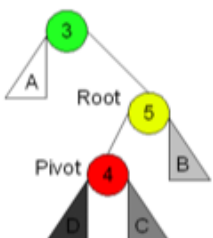
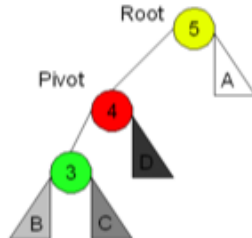
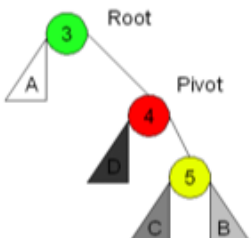
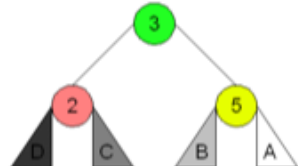

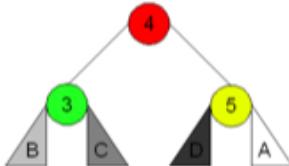
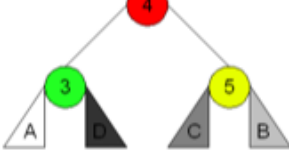
Double Rotation



Done!

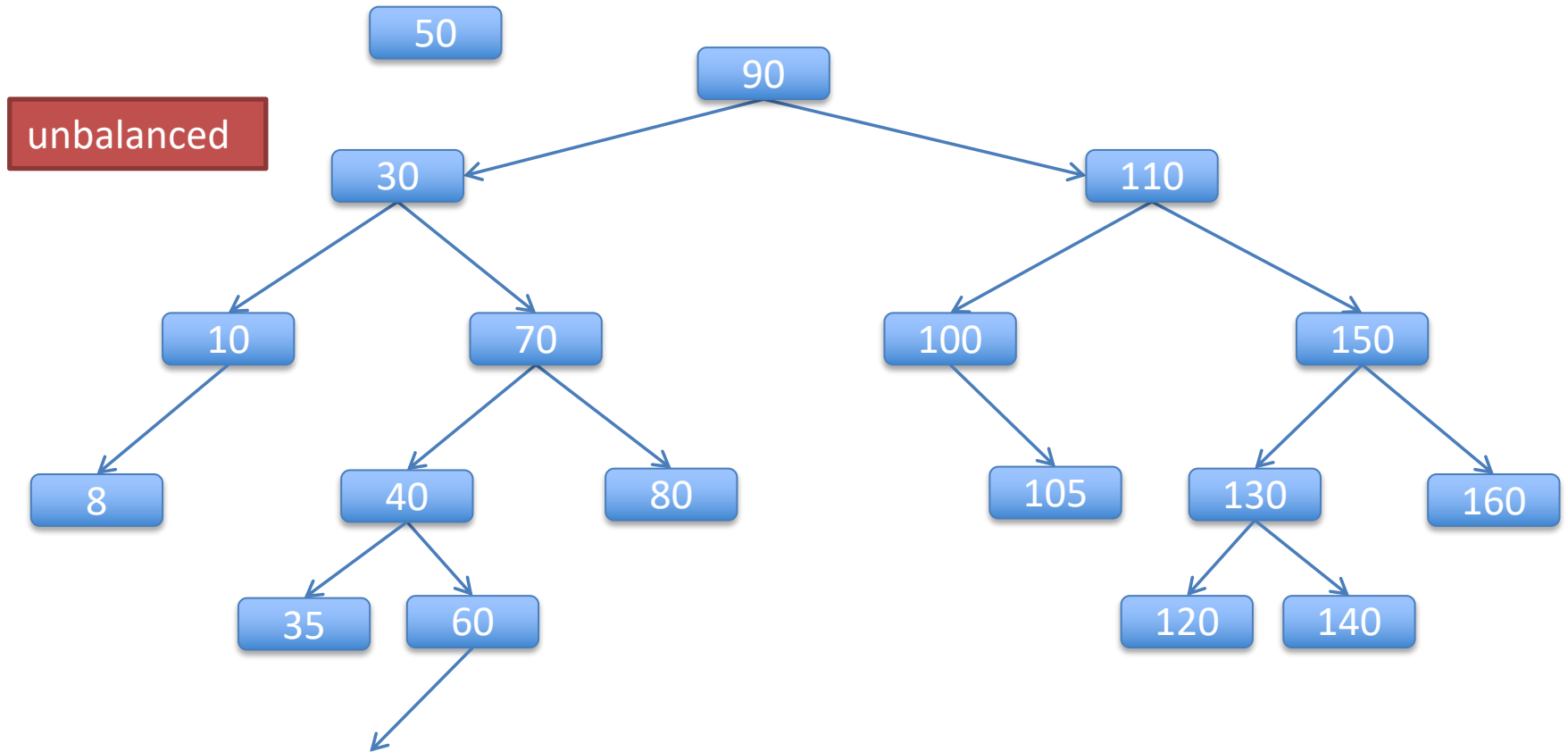
There are 4 cases in all, choosing which one is made by seeing the direction of the first 2 nodes from the unbalanced node to the newly inserted node and matching them to the top most row.

**Root** is the initial parent before a rotation and **Pivot** is the child to take the root's place.

| Left Left Case   | Right Right Case   | Left Right Case   | Right Left Case   |
|--|--|---|---|
|  <p>Root Rotation</p> |  <p>Left Rotation</p> |  <p>Left Rotation</p>  |  <p>Right Rotation</p> |
|  |  |  <p>Right Rotation</p> |  <p>Left Rotation</p>  |
|                     |                    |                      |                      |

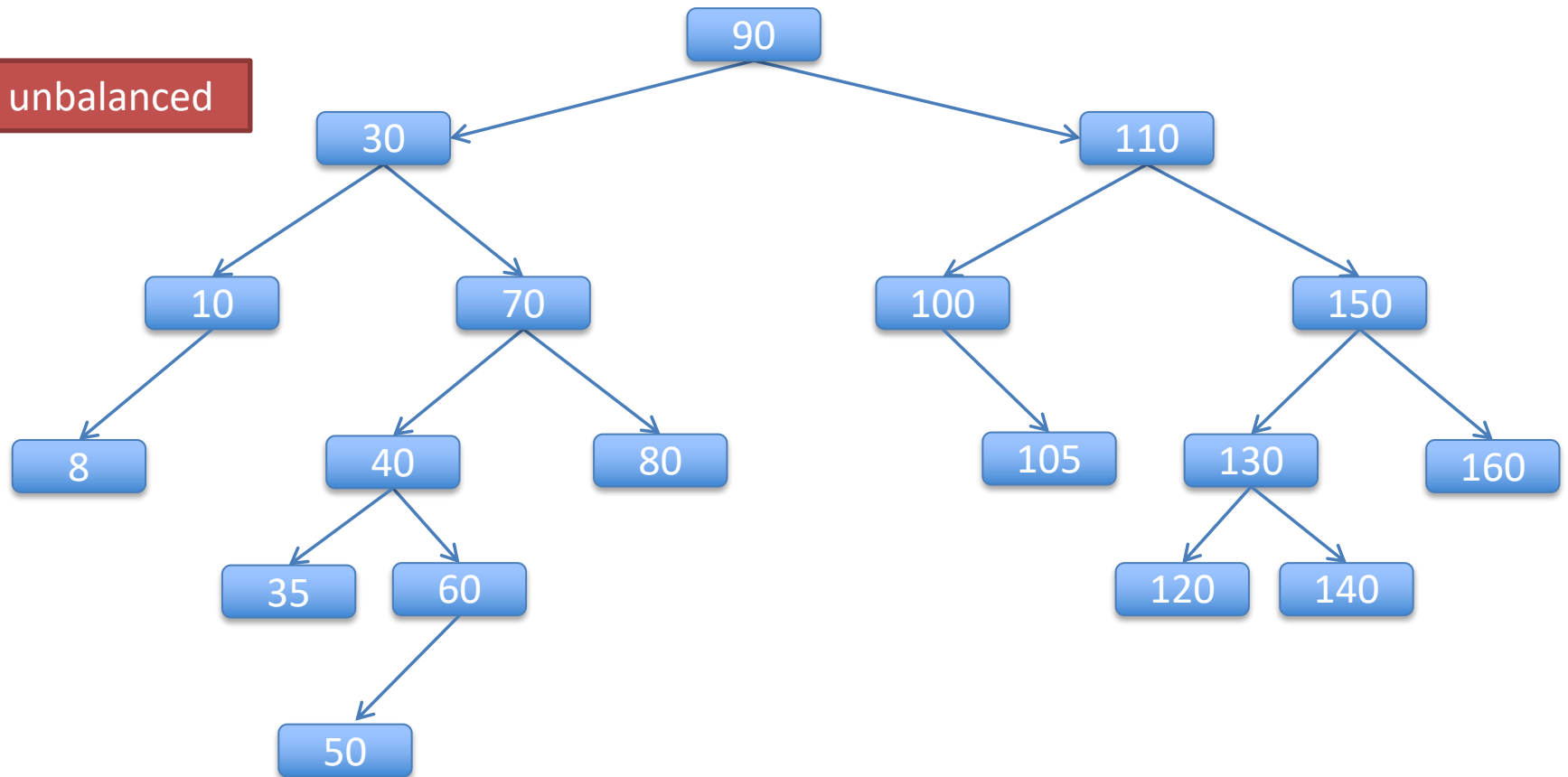
Picture from Wikipedia

# Insert



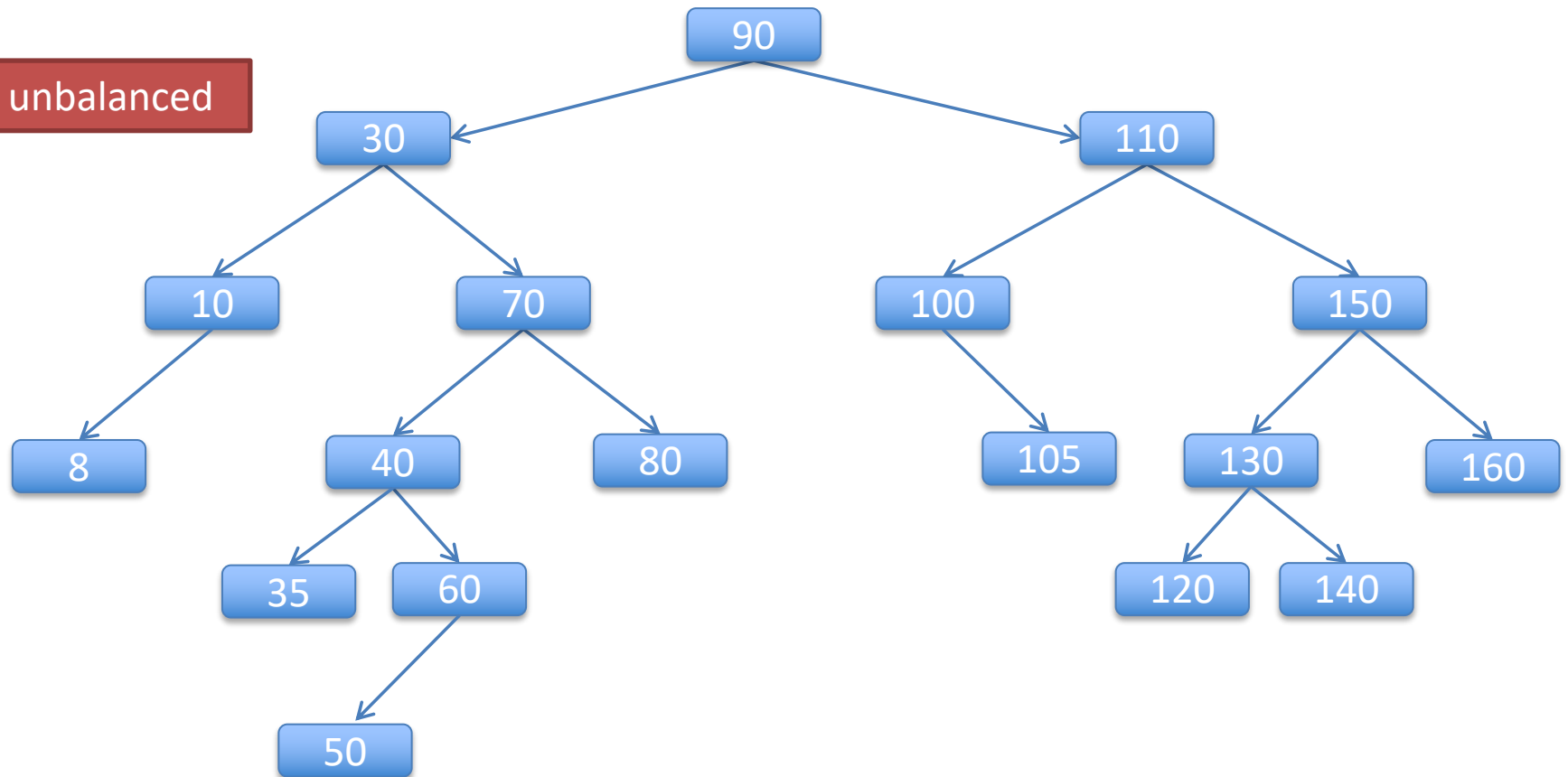
# Insert

unbalanced



# Insert

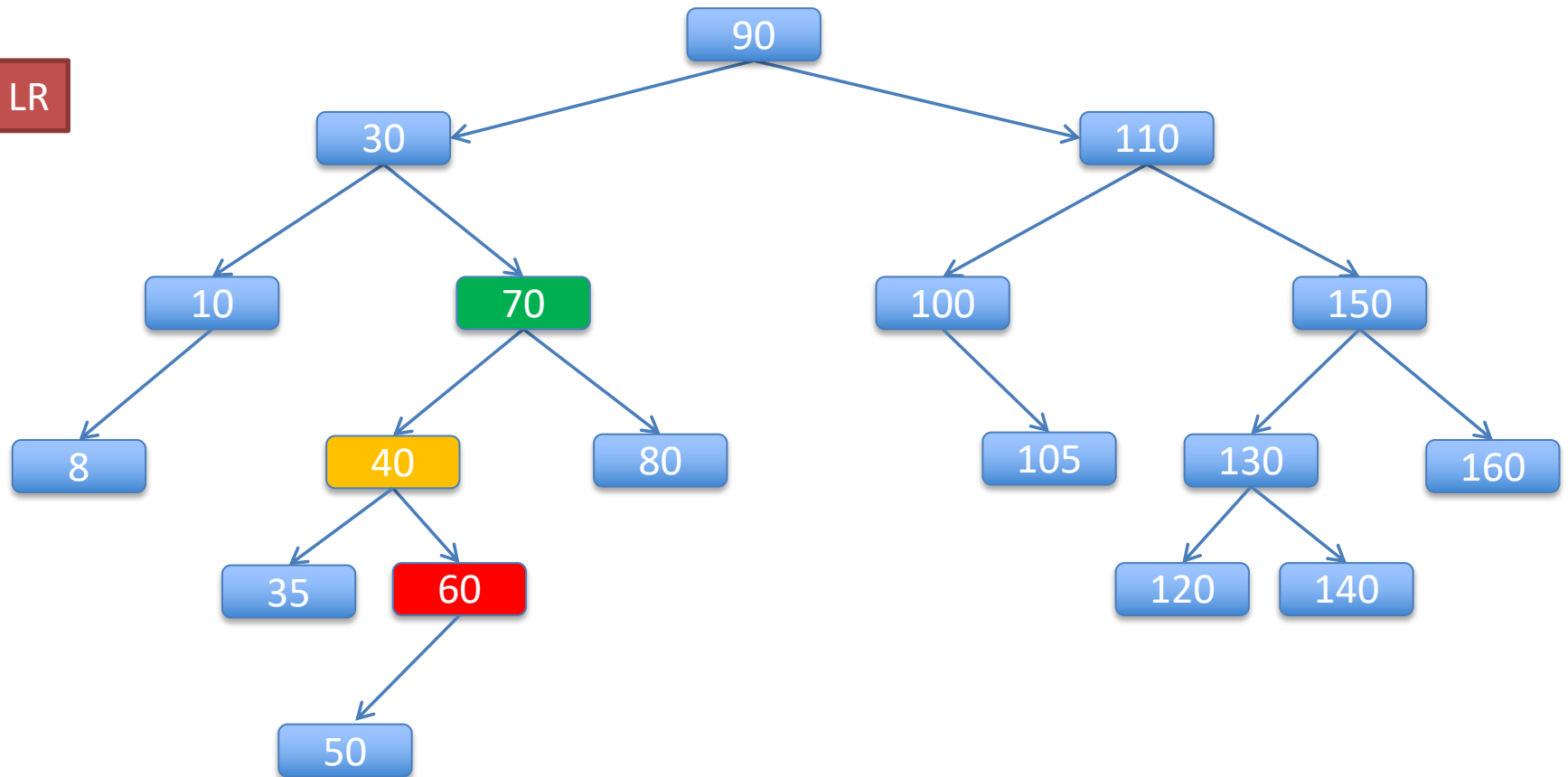
unbalanced





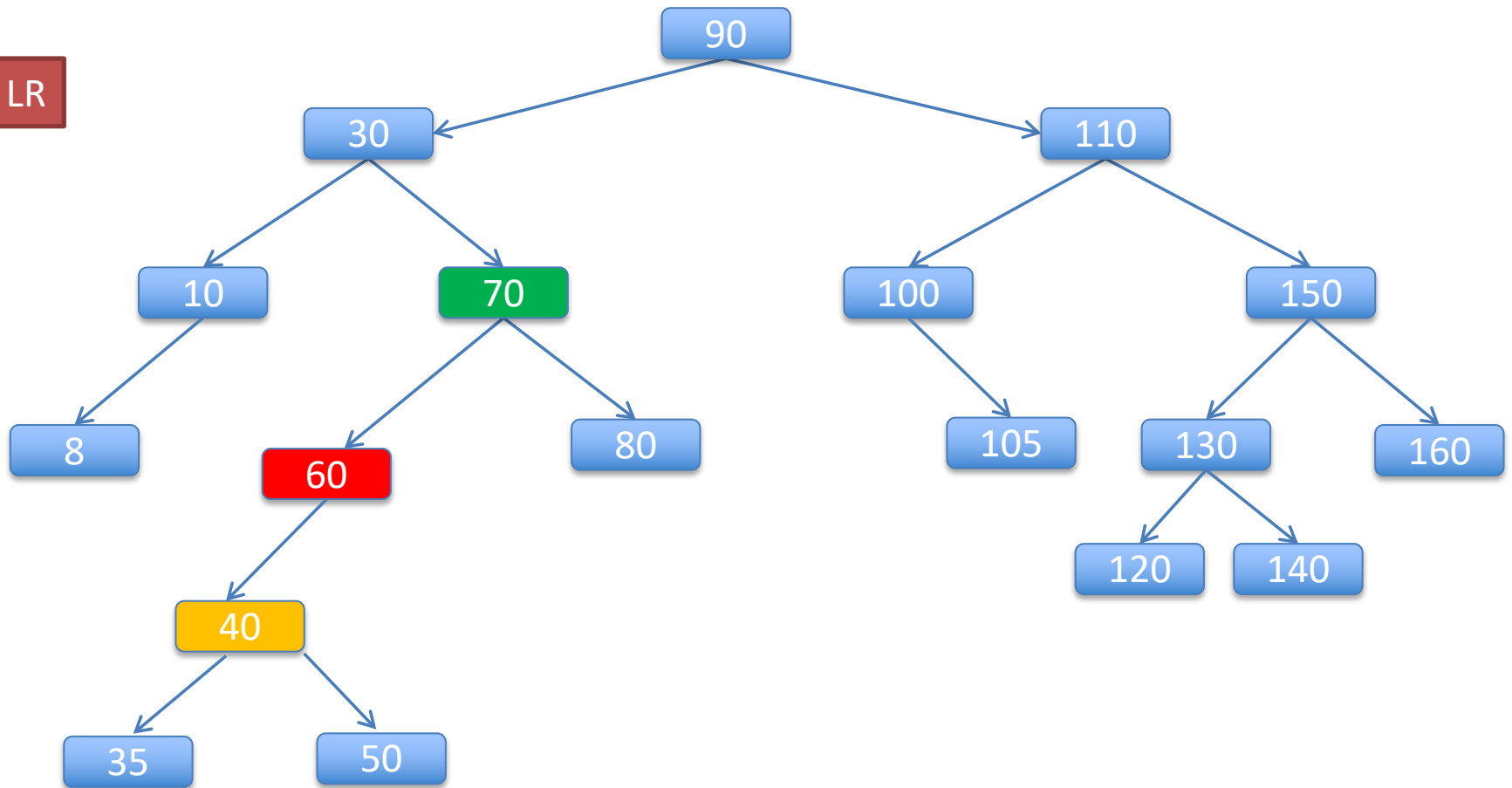
# Insert

LR



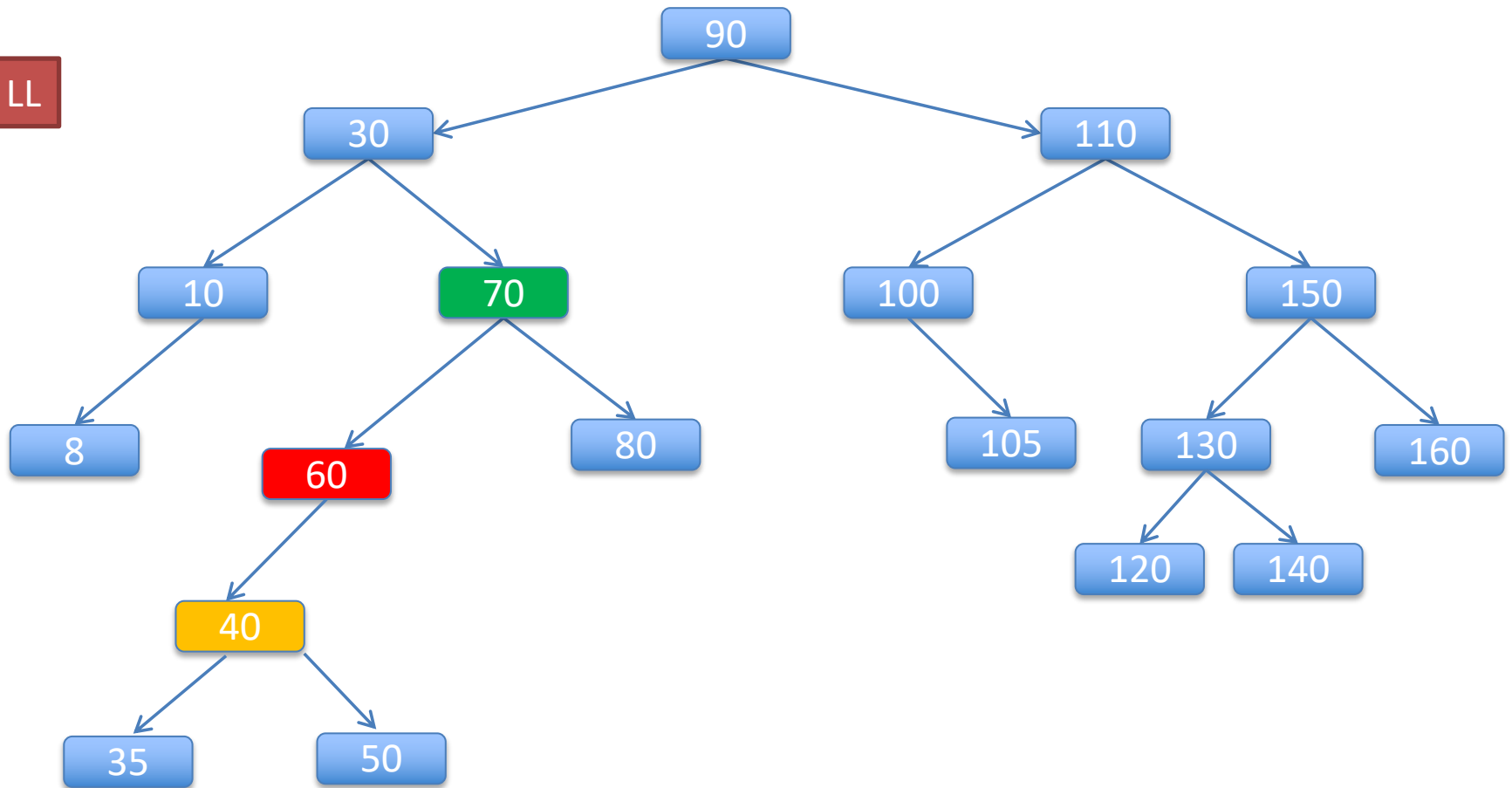
# Insert

LR



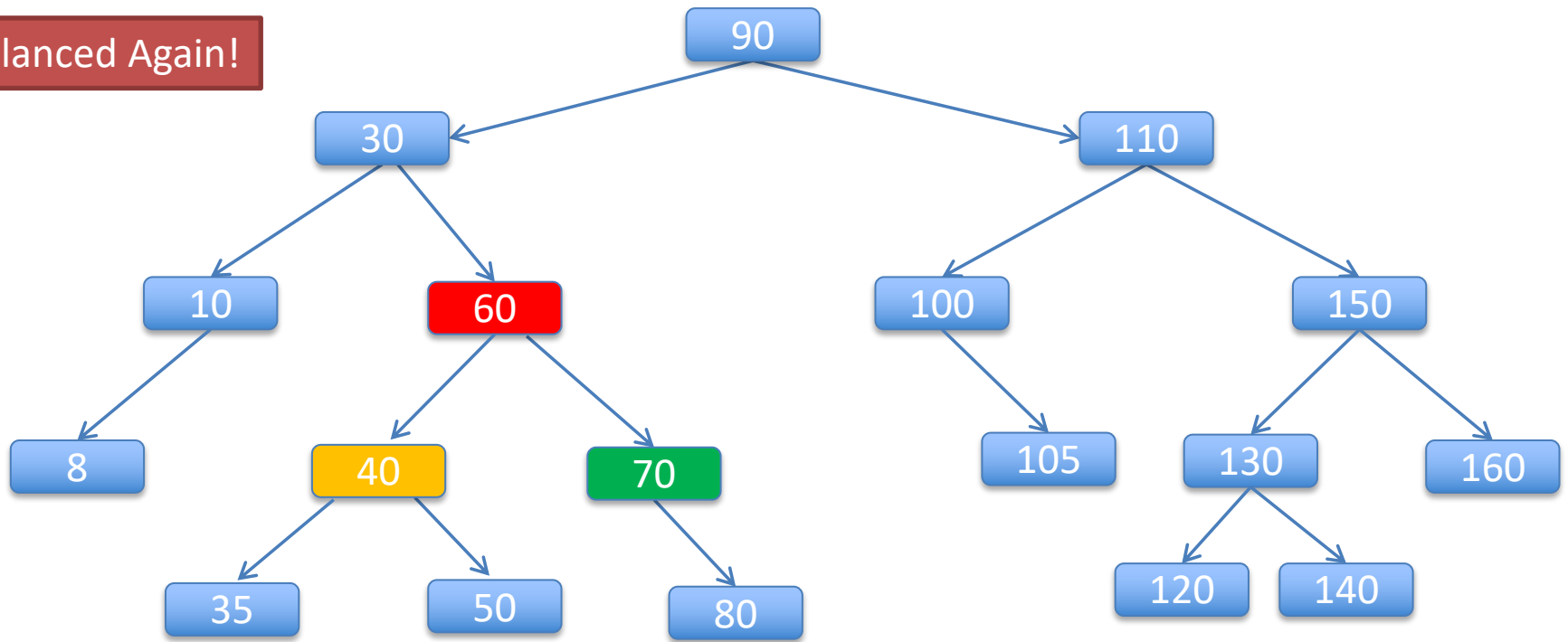
# Insert

LL



# Insert

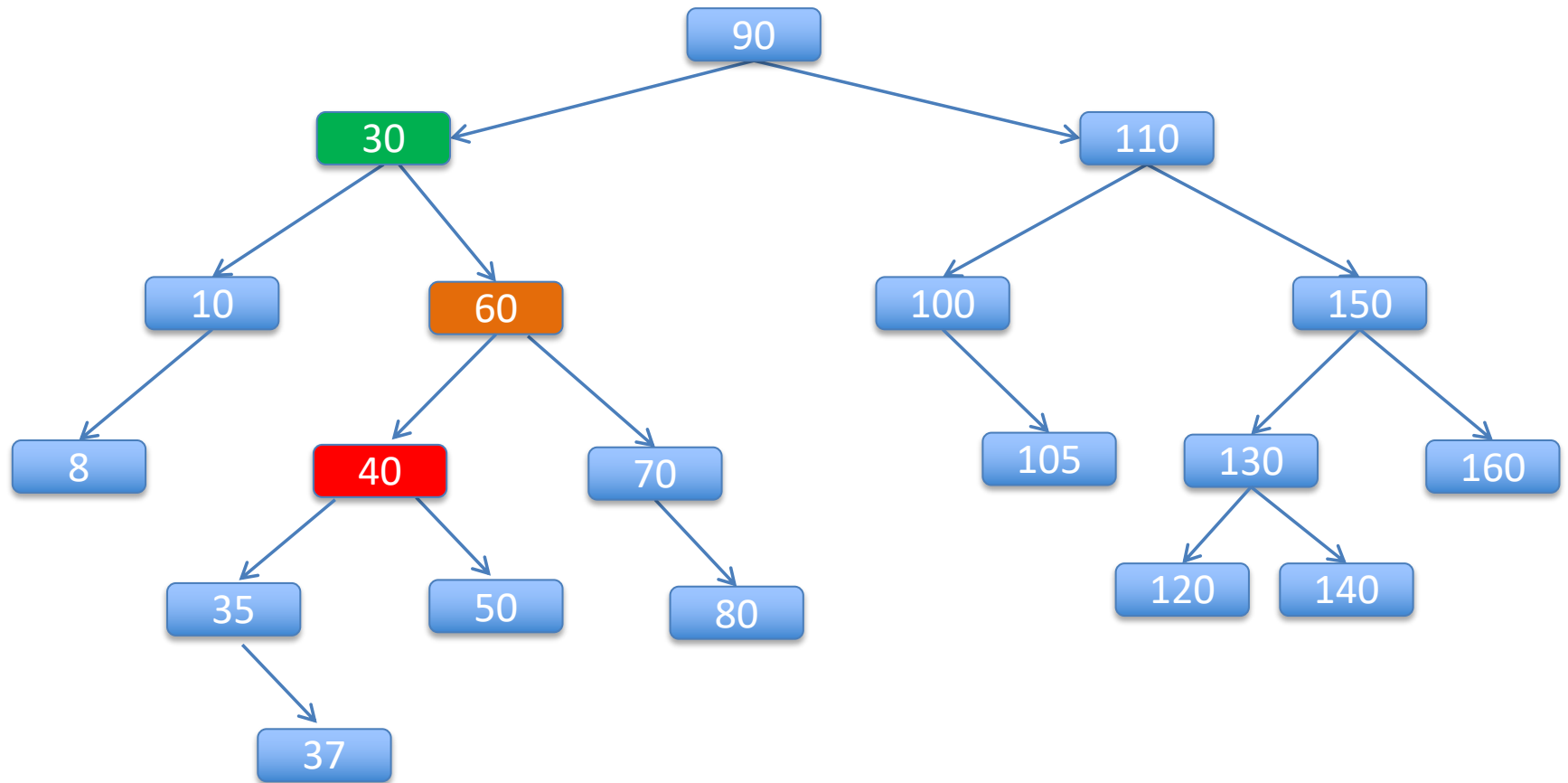
Balanced Again!



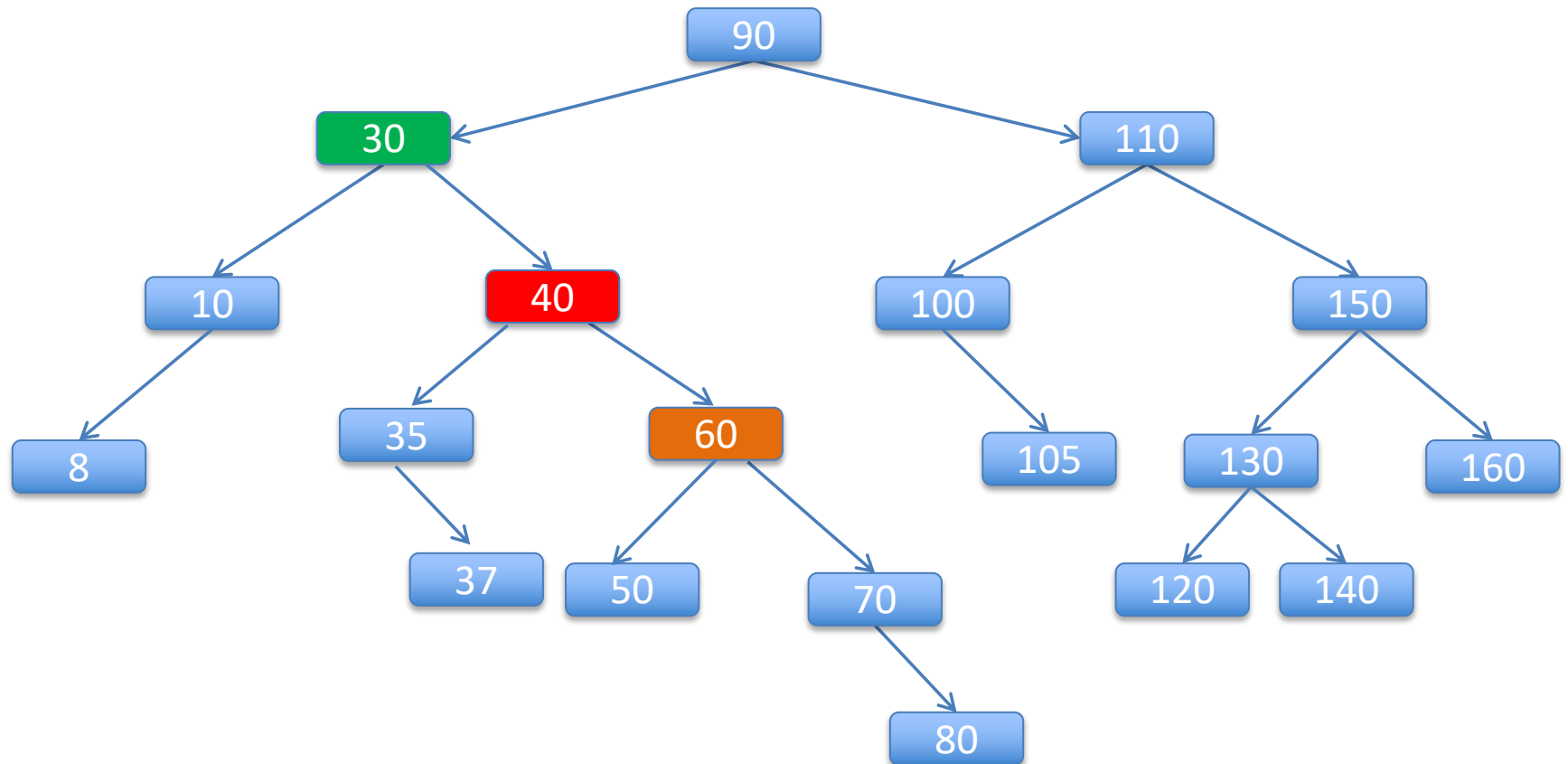
# In-class Exercise

- Adding 37 to the earlier AVL tree

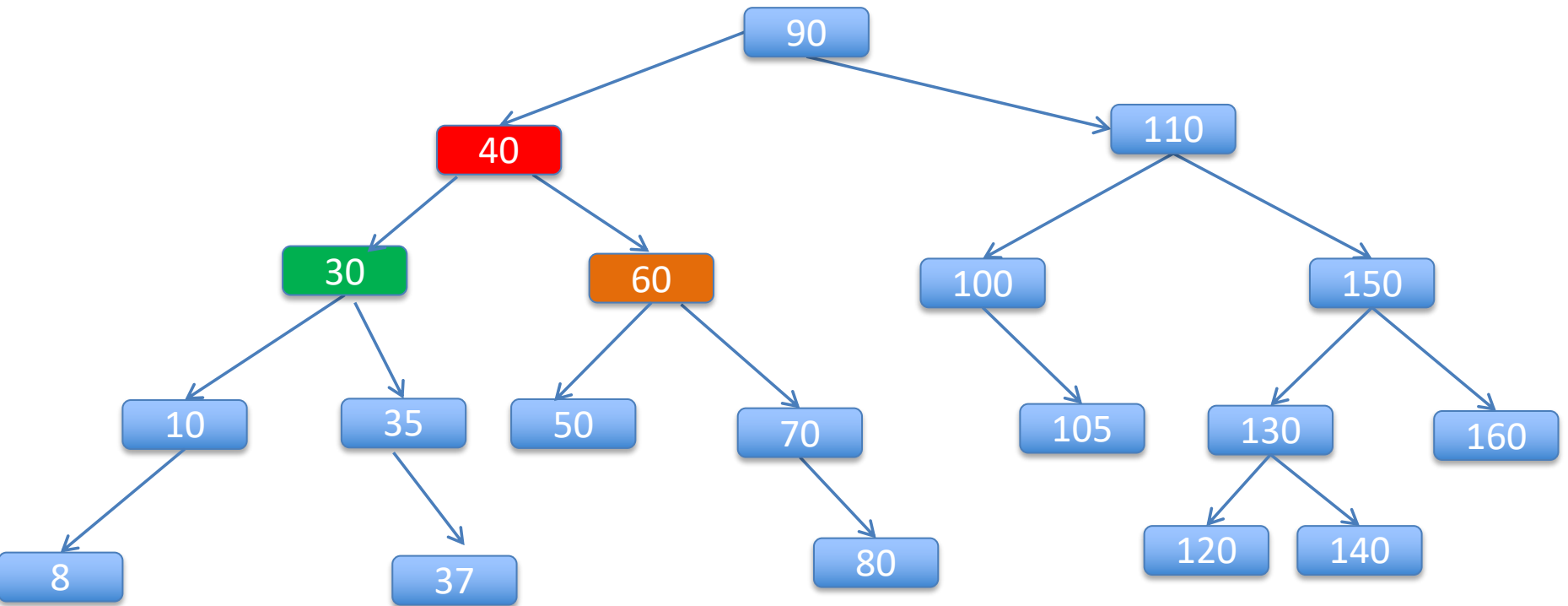
# Insert



# Insert



# Insert

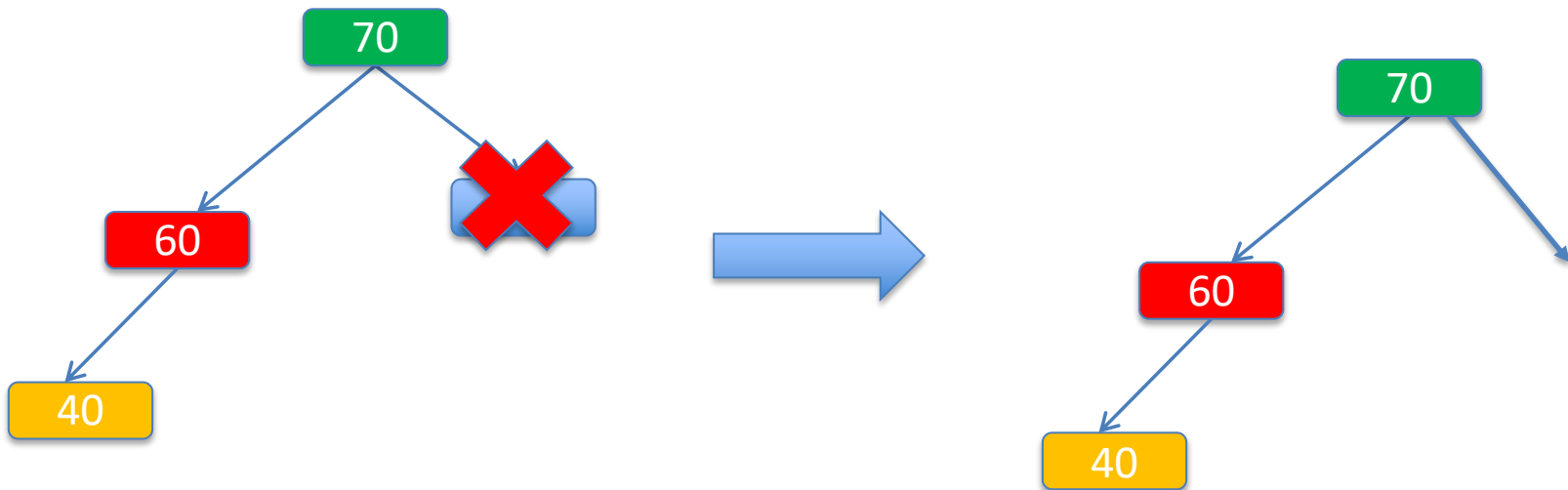




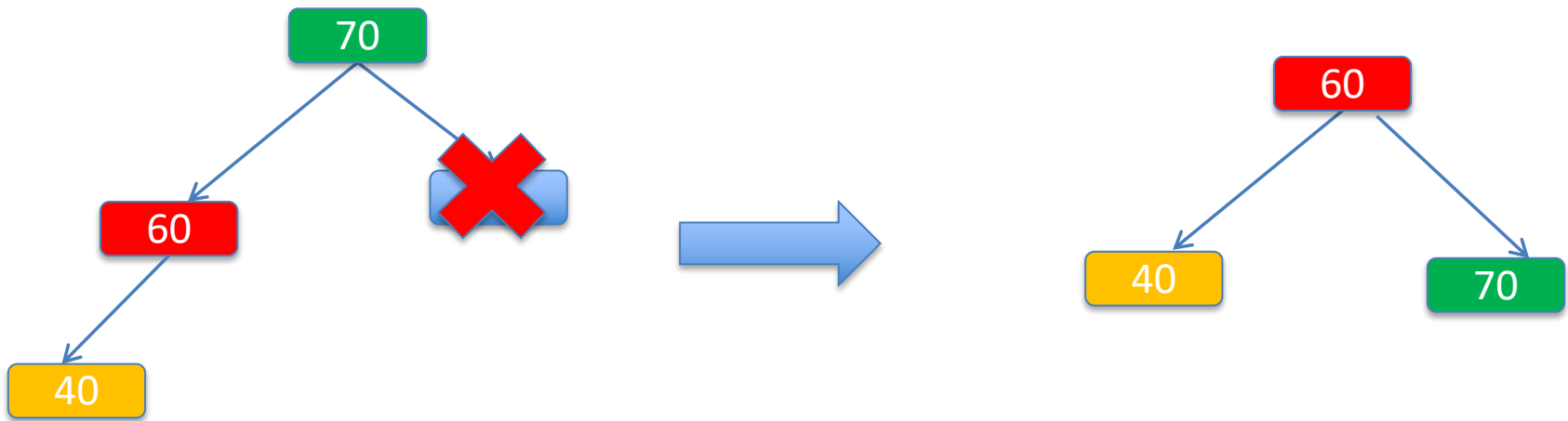
# Delete

- First, delete as in normal BST
  - But nodes on path to root might become unbalanced
- Second, fix unbalanced nodes one by one using exactly the same strategy
  - Might require up to  $O(\log n)$  rotations
- Insert & delete run in time  $O(\log n)$

# Deletion Example



# Deletion Example

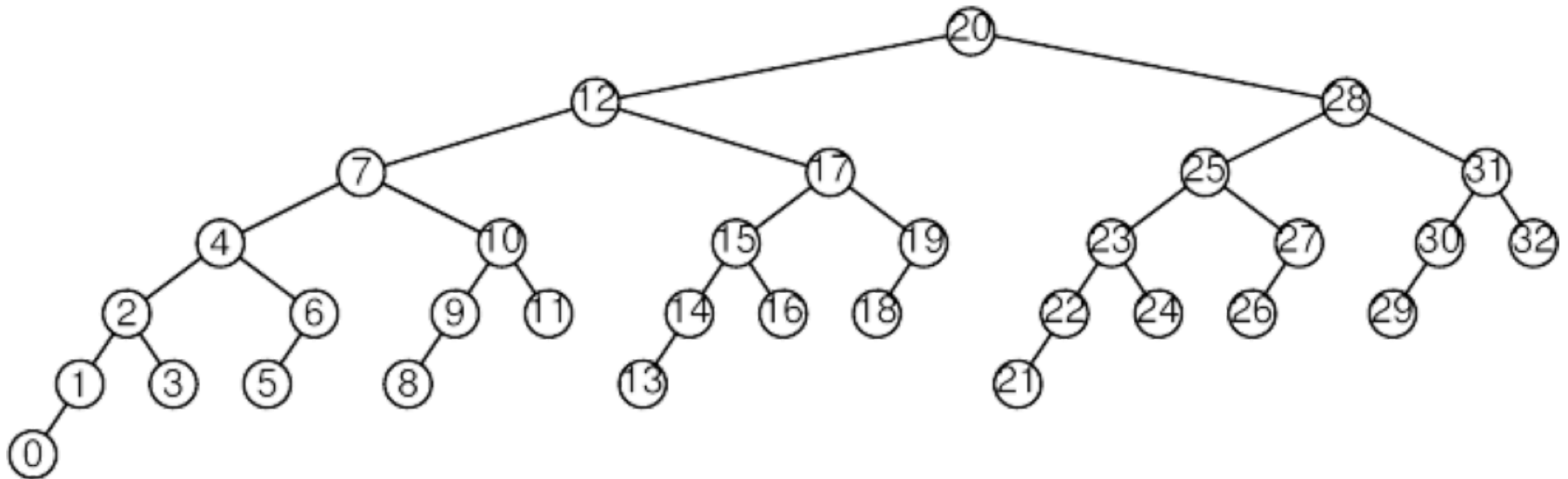


# Theorems

- Insertion:
  - After fixing one node (with a single/double rotation) the tree becomes balanced (i.e. AVL again) – why?
- Deletion:
  - Fixing one node does not necessarily balance the tree
  - Need more fixing up to the root
- Think of an example AVL-tree for which  $\Omega(\log n)$  fixes are necessary after a deletion!

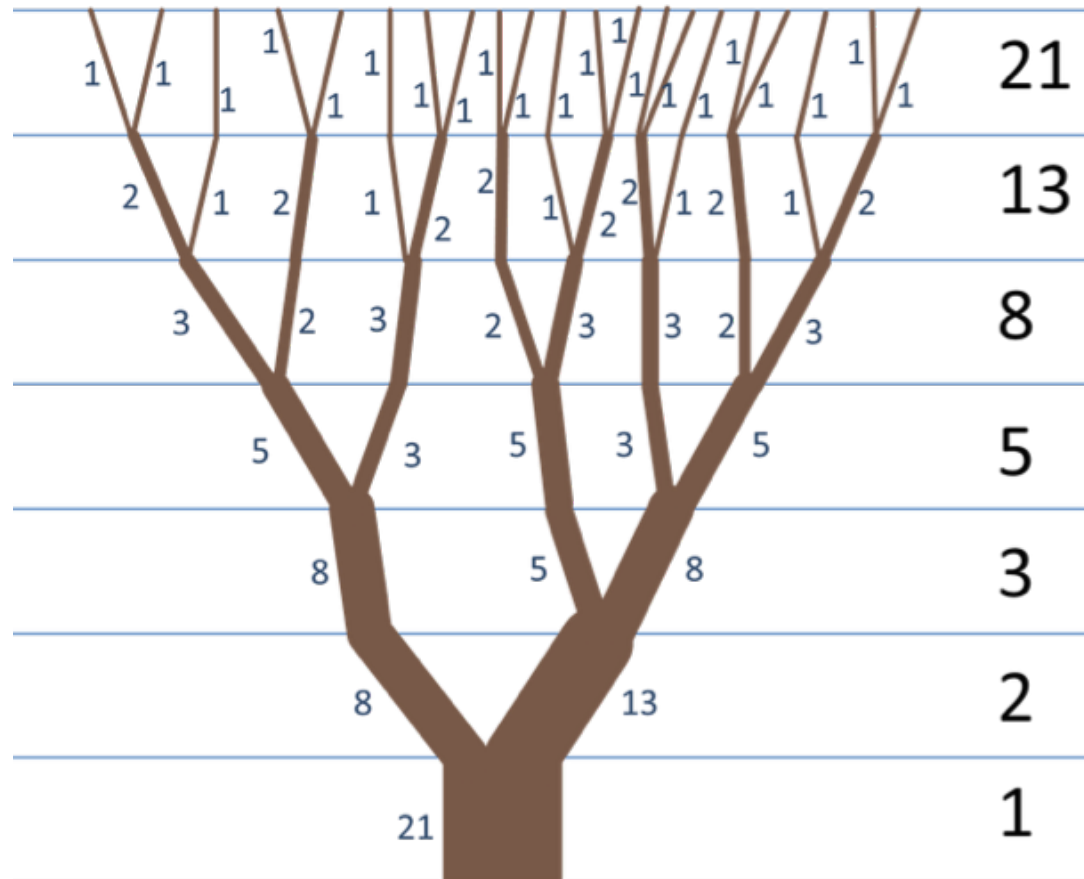
# Let's try this!

- Delete 32



# Last Example

- Fibonacci Tree



# Summary: AVL trees

- AVL trees:
  - All operations logarithmic worst-case because trees are always balanced.
  - The height balancing adds no more than a constant factor to the speed of insert and delete.
  - Arguments against AVL trees:
    - Difficult to program & debug
    - More space for height field
    - Rebalancing takes time
    - Most large searches are done in database systems on disk
      - B+ Tree is often used for that
    - If amortized logarithmic time is enough, use splay trees (not covered)

(Mostly for further reading)

# Summary (Balanced BST)

- Balanced Search Tree:
  - Insertion, Deletion, and Searching  $O(\log n)$ 
    - Improvement over  $O(h)$  where  $h$  = height of the tree
      - $h$  can  $O(n)$  in the worst case.
- Is Balanced Search Tree ever used?:
  - Yes. Java uses Red-Black tree for constructing `TreeMap`.
  - What is `TreeMap`?
    - A Map (used for accessing values using key)
      - Stores (key, value) pair
    - Not a HashMap
    - The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time



# Summary

- What did we cover?
  - Heap
  - Binary Search Tree (BST)
  - Balanced BST
- Data Structures that use these data structures:
  - Priority Queue uses Heap
  - TreeMap uses Balanced BST

# Acknowledgement

- Douglas Wilhelm Harder.
  - Thanks for making an excellent set of slides for ECE 250 *Algorithms and Data Structures* course
- Prof. Hung Q. Ngo:
  - Thanks for those beautiful slides created for CSC 250 (Data Structures) course at UB.
- Many of these slides are taken from these two sources.