# Data Structures in Java

Session 9

Instructor: Bert Huang

http://www1.cs.columbia.edu/~bert/courses/3134

# Announcements

- Homework 1 grades posted
- Homework 2 due
- Homework 3 posted, due 10/20
- Nikhil will teach next class

# Review

- Search Tree ADT
- Binary search tree
  - implementation
  - running time analysis

# Today's Plan

- AVL Trees (Balanced BST)
  - Implementation and analysis

# Tradeoffs

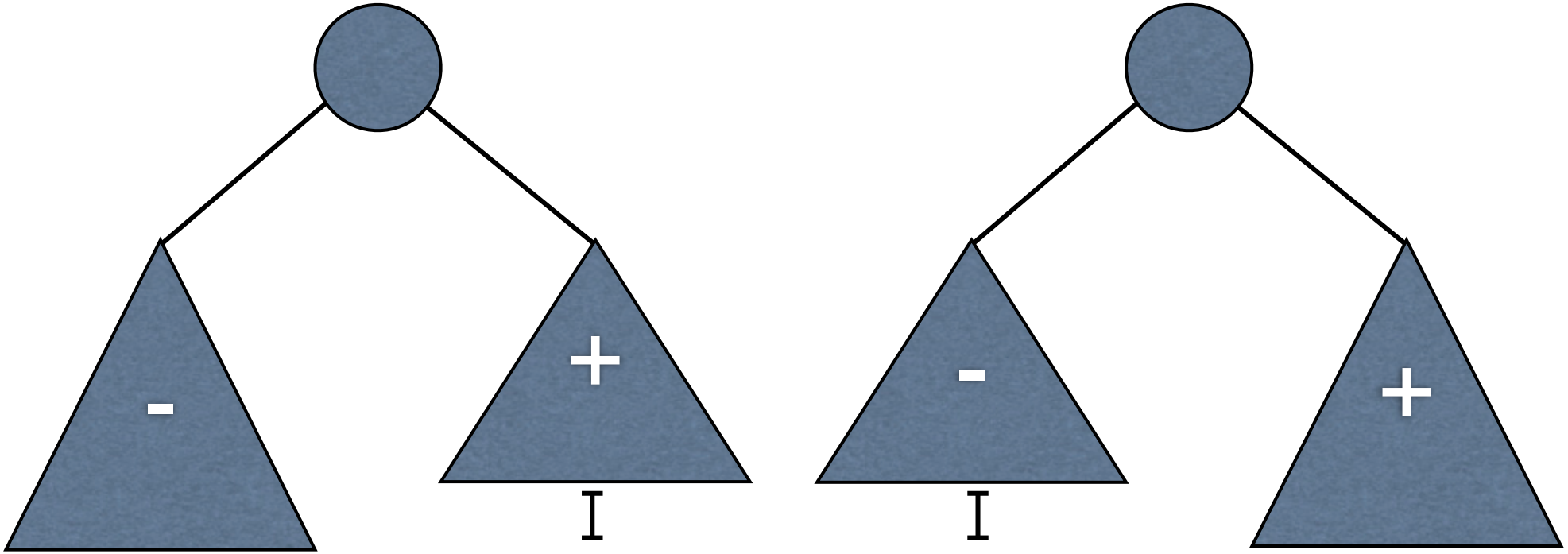
| | insert | remove | lookup | index |
|---|---|---|---|---|
| ArrayList | O(N) | O(N) | O(N) | O(1) |
| LinkedList | O(1) | O(1) | O(N) | O(N) |
| Stack/Queue | O(1) | O(1) | N/A | N/A |
| BST | O(d)=O(N) | O(d)=O(N) | O(d)=O(N) | N/A |
| AVL | O(log N) | O(log N) | O(log N) | N/A |

- There may not be free lunch, but sometimes there's a cheaper lunch

# AVL Trees

- Motivation: want height of tree to be close to log N
- AVL Tree Property:
  For each node, all keys in its left subtree are less than the node's and all keys in its right subtree are greater.
  **Furthermore, the height of the left and right subtrees differ by at most 1**

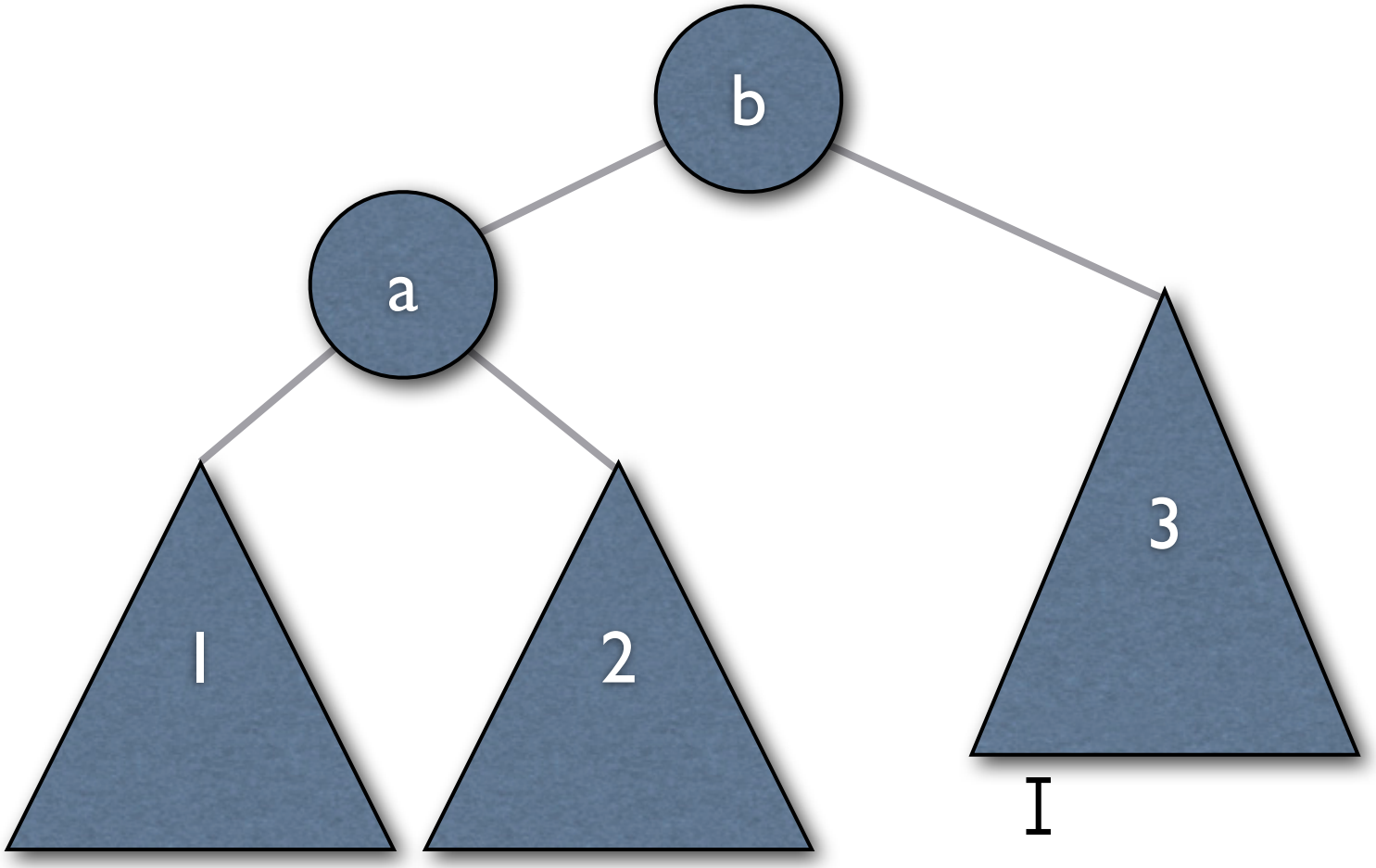
# AVL Tree Visual



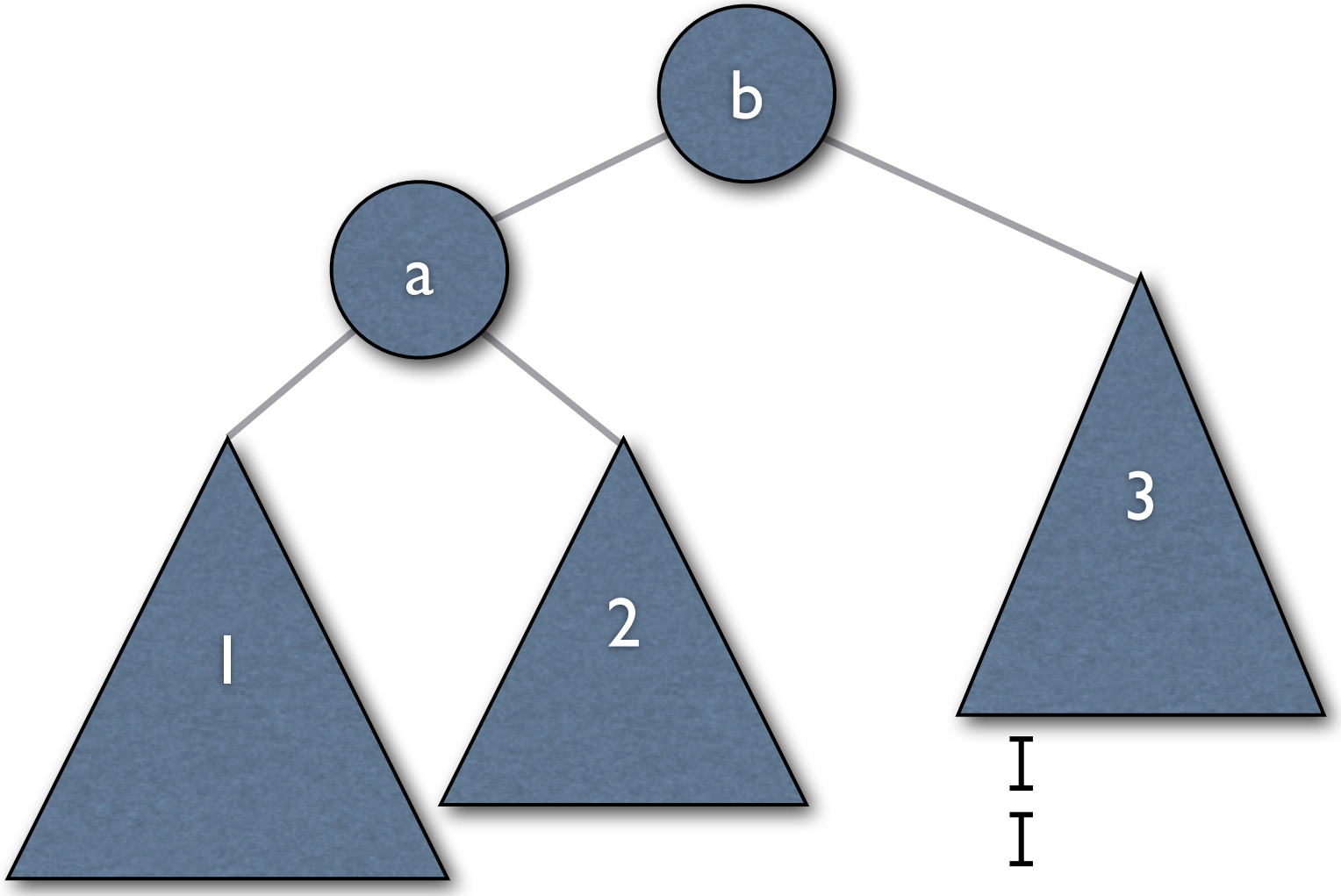
# Tree Rotations

- To balance the tree after an insertion violates the AVL property,
  - rearrange the tree; make a new node the root.
  - This rearrangement is called a **rotation.**
  - There are 2 types of rotations.

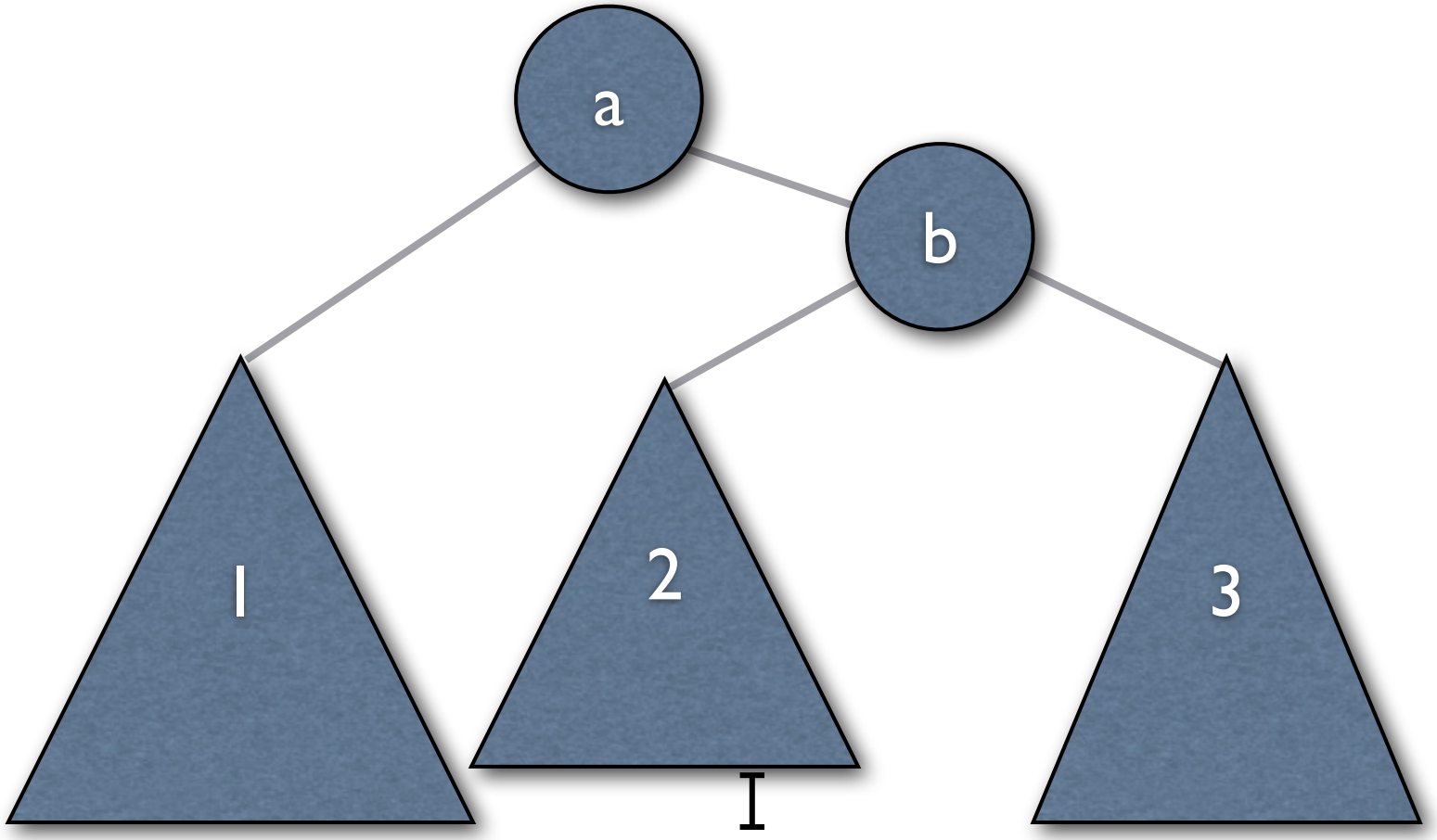
# AVL Tree Visual: Before insert



# AVL Tree Visual: After insert



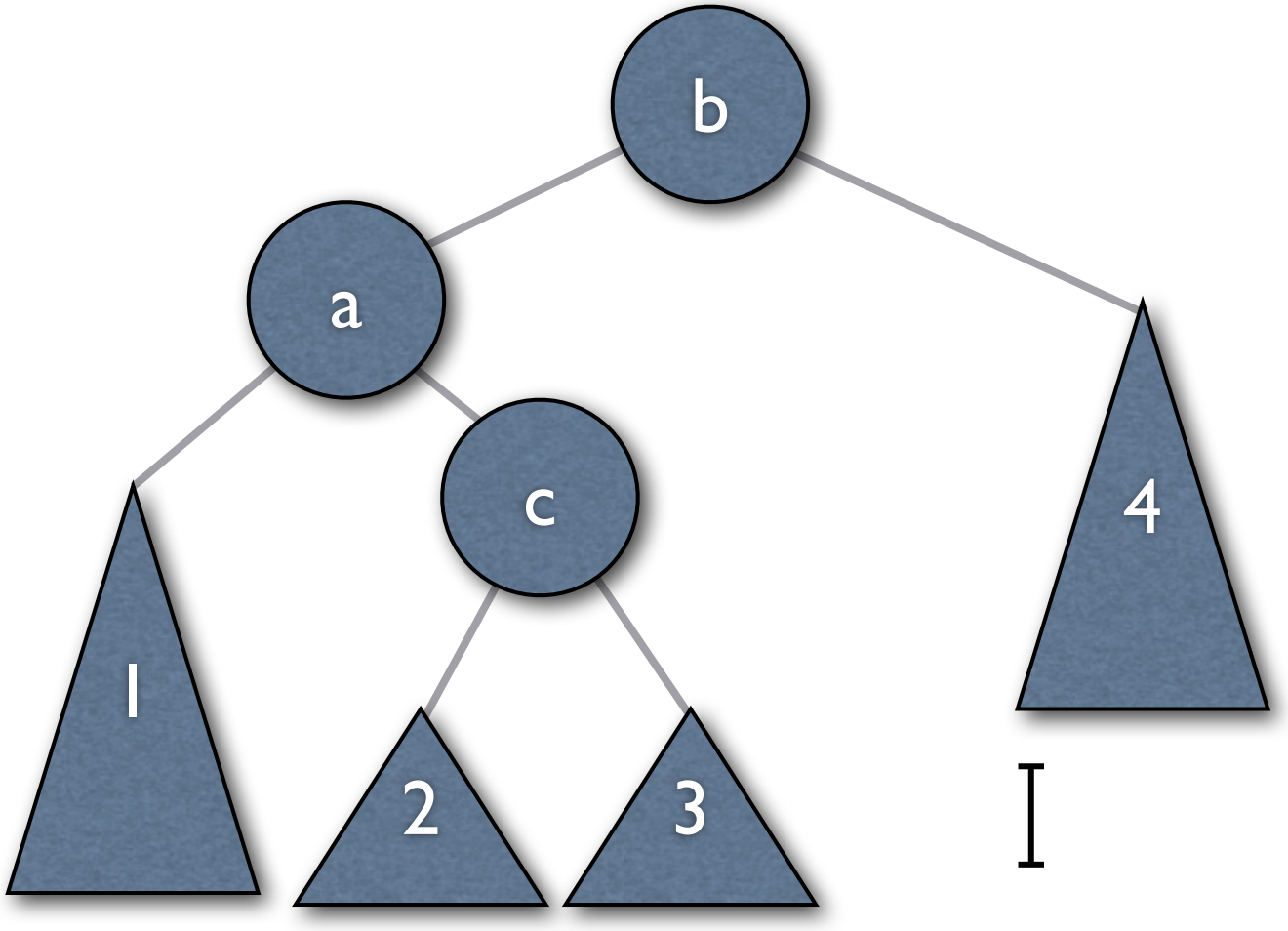
# AVL Tree Visual: Single Rotation



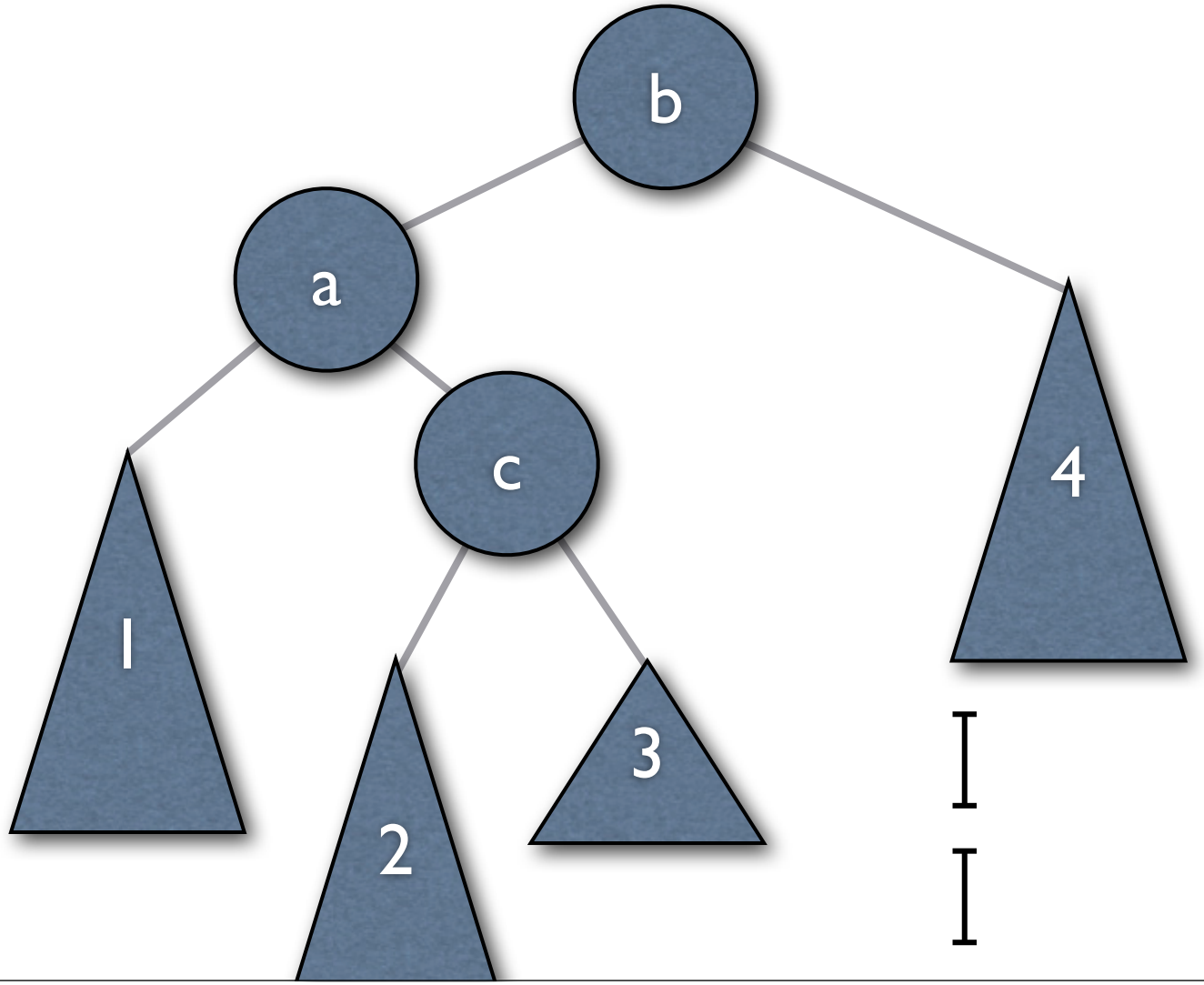
# AVL Tree
# Single Rotation

- Works when new node is added to outer subtree (left-left or right-right)
- What about inner subtrees? (left-right or right-left)

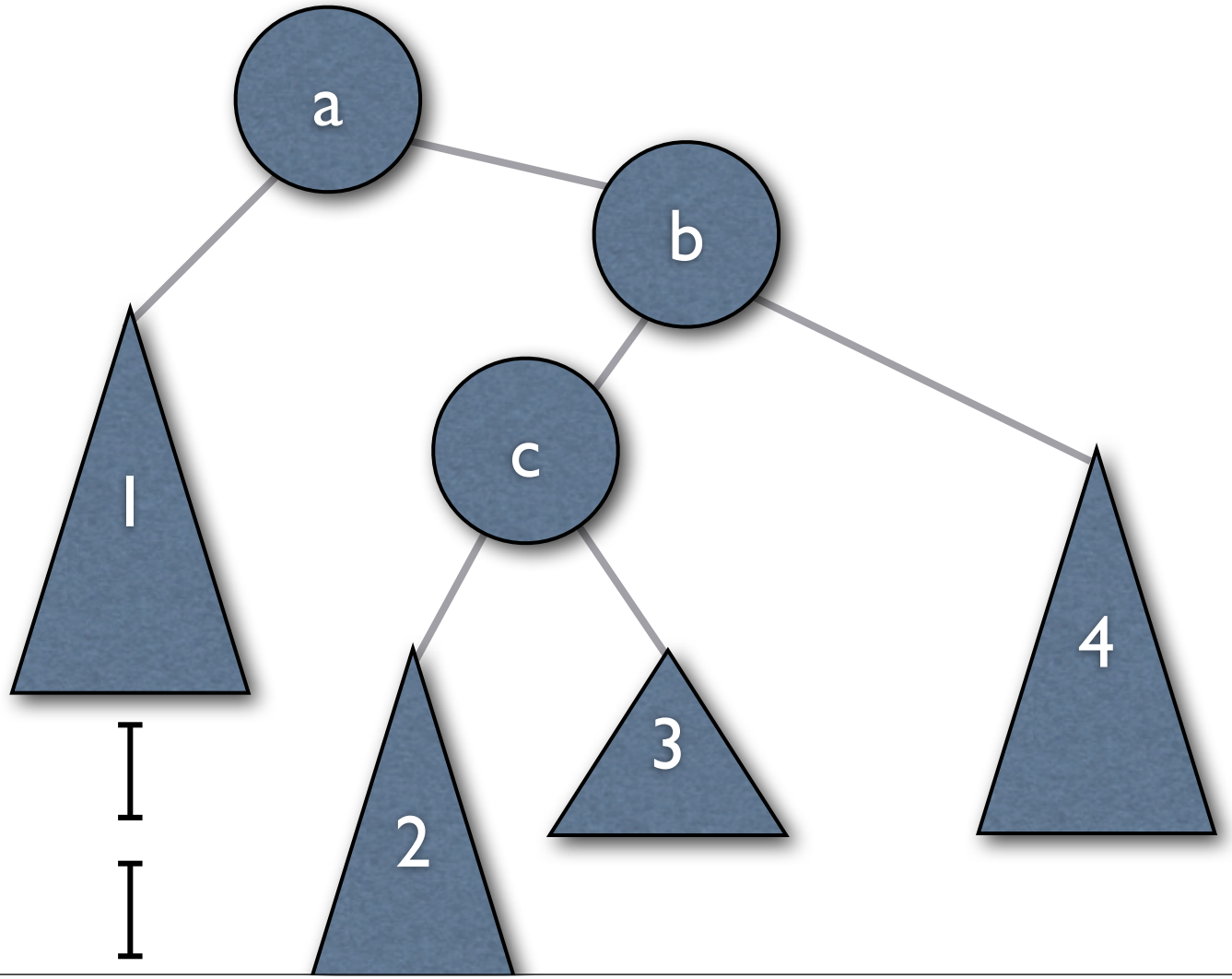
# AVL Tree Visual: Before Insert 2



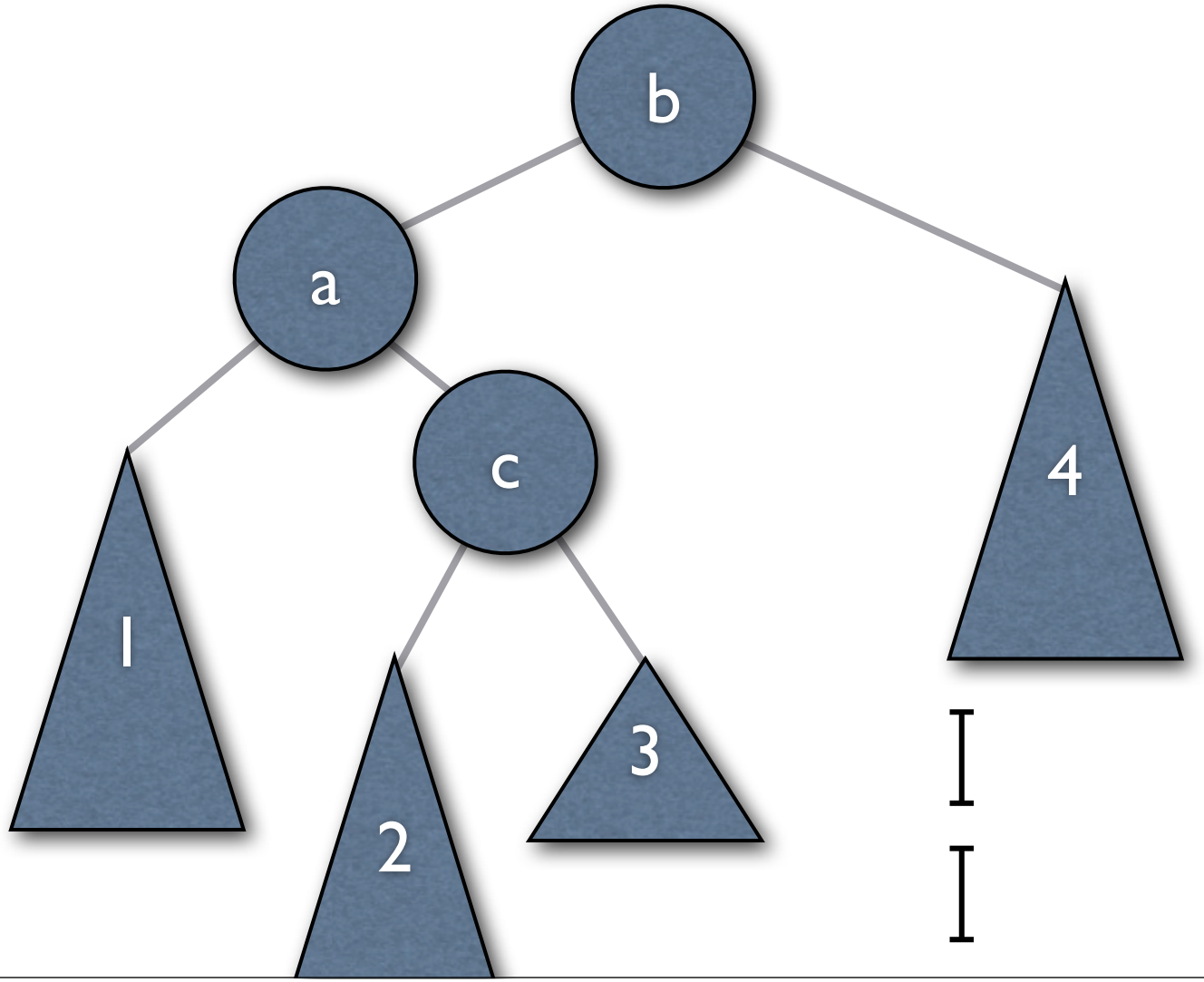
# AVL Tree Visual: After Insert 2



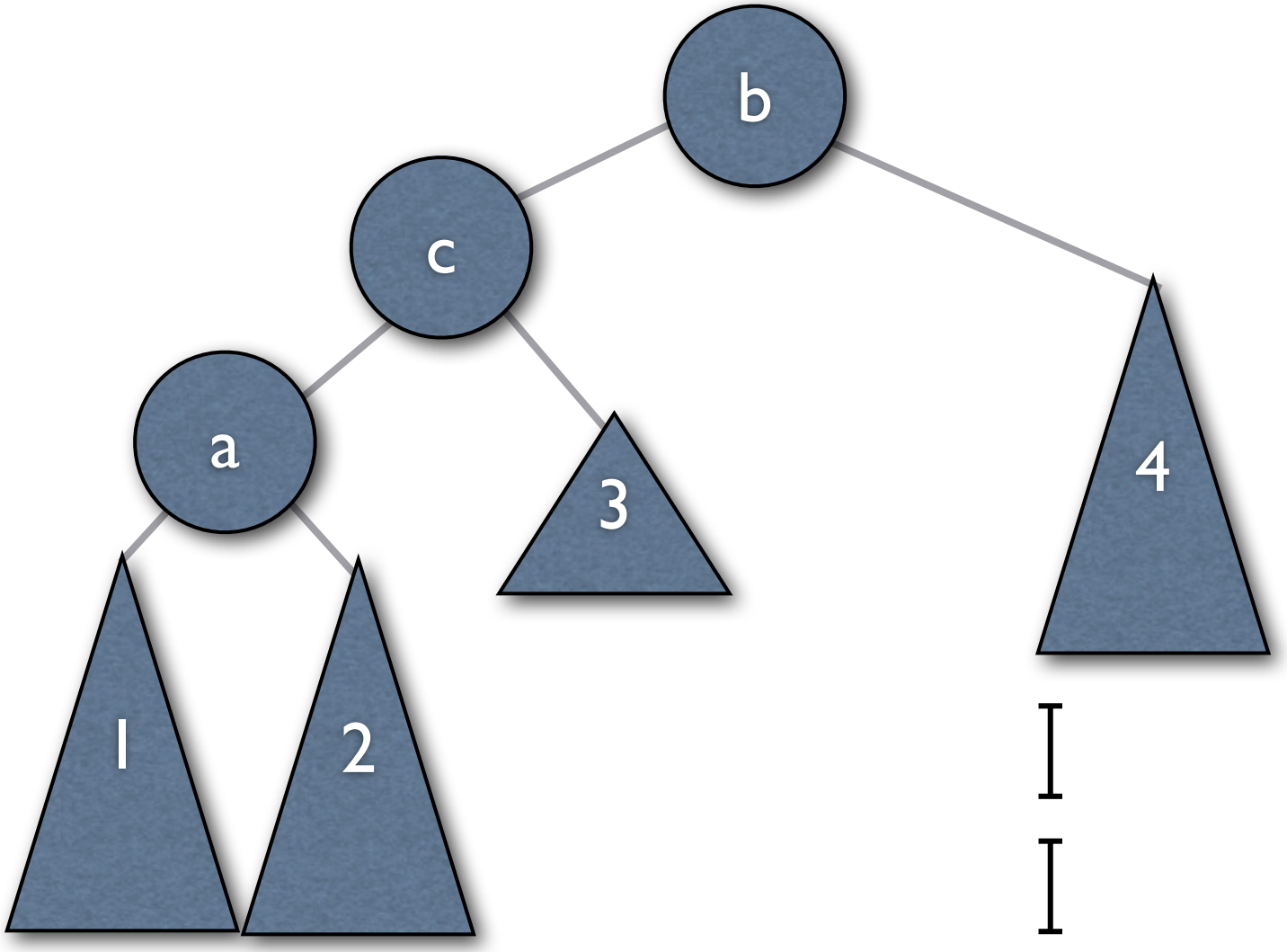
# AVL Tree Visual: Single Rotation Fails



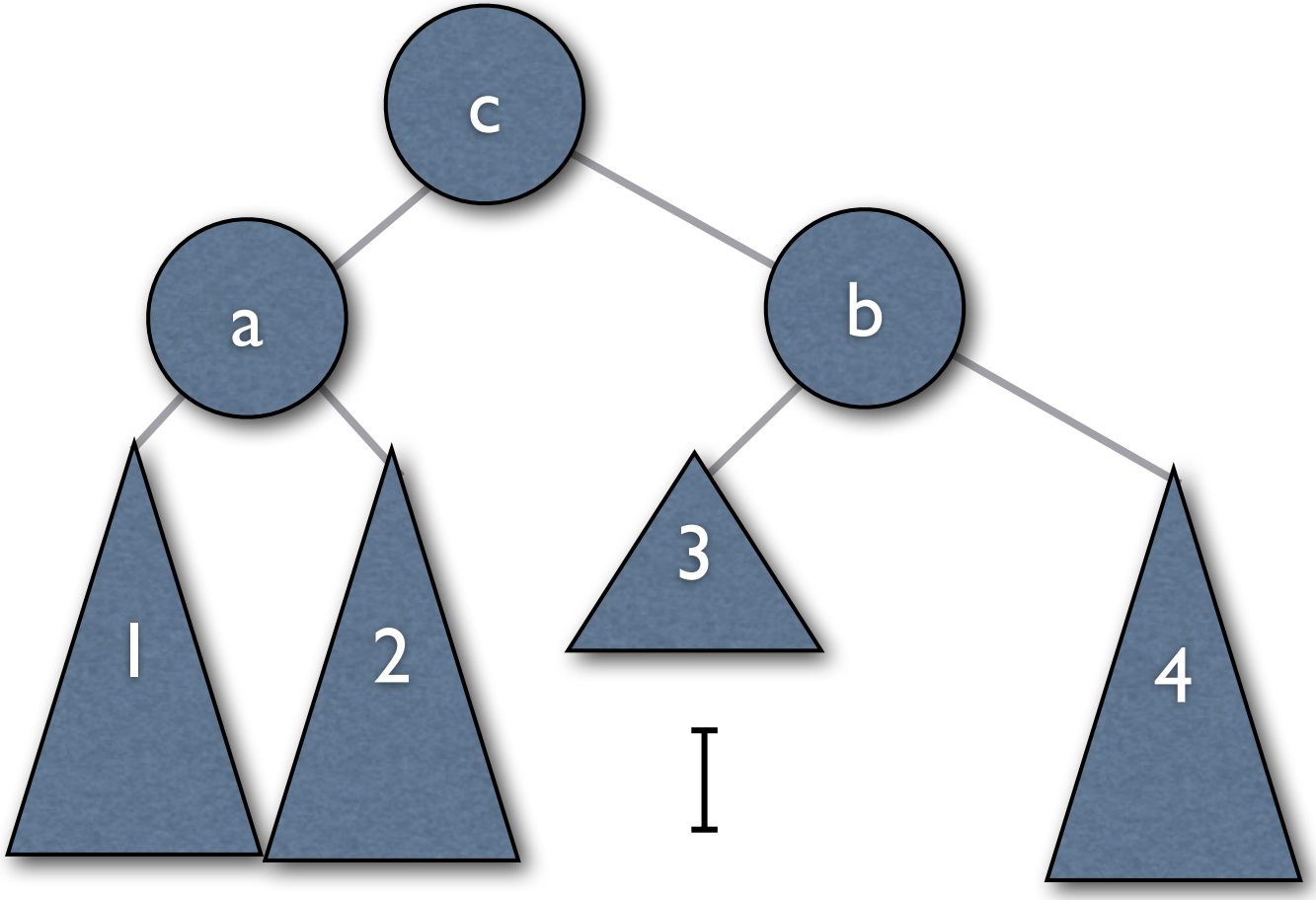
# AVL Tree Visual: Double Rotation



# AVL Tree Visual: Double Rotation



# AVL Tree Visual: Double Rotation



# Rotation Algorithms

- Single Rotation L:
  tmp = root.left
  root.left = tmp.right
  tmp.right = root

- Double Rotation LR:
  singleRotate R root.left
  singleRotate root

# Rotation running time

- Constant number of link rearrangements
- Double rotation needs twice as many, but still constant
  - So AVL rotations do not change O(d) running time of all BST operations*
- * remove() can require up to O(d) rotations; use lazy deletion

# Depth analysis

- Worst case: minimum number of nodes in an AVL tree of height *h*: *N(h)*
- *N(1) = 1, N(2) = 2*
- For greater heights, the total number of nodes includes:
  - the root node
  - the # of nodes in a subtree of size *h-1*
  - the # of nodes in a subtree of size *h-2*

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

# Depth analysis

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

- Recursively subbing in the formula above, we get

$$\begin{aligned} N(h) &= 1 + (1 + N(h-2) + N(h-3)) + \\ &\quad (1 + N(h-3) + N(h-4)) \end{aligned}$$

- Combine all the newly generated 1's, then recurse

$$\begin{aligned} N(h) &= 1 + 2 \\ &\quad + (1 + N(h-3) + N(h-4)) \\ &\quad + (1 + N(h-4) + N(h-5)) \\ &\quad + (1 + N(h-4) + N(h-5)) \\ &\quad + (1 + N(h-5) + N(h-6)) \end{aligned}$$

# Depth analysis

- Each time we recurse, we generate a new constant, which is the count of the number of evaluations we did.
- We can recurse h/2 times before at least one N(h-k)=N(0)
- Therefore, we can lower bound

$$N(h) > \sum_{i=0}^{h/2} 2^i > 2^{h/2}$$

$$N(h) > \sum_{i=0}^{h/2} 2^i > 2^{h/2}$$

- Solve for h,
$$\begin{aligned} \log(N) &> h/2 \\ 2\log(N) &> h \\ h &= O(\log N) \end{aligned}$$
- Recap:
  - We analyzed minimum number of nodes necessary to cause height *h*
  - We lower bounded that minimum; a formula that is even worse than the worst case
  - We showed that worst case means the height is still O(log N)

# Looking Forward

- AVL Trees aggressively guarantee log running time
  - Every operation is now log running time
  - May be overkill

# Reading

- This class: 4.4
- Next class: 4.5