

Chapter 5, Binary search trees:

- ▶ Binary search trees; the balanced BST problem (fall-break eve; finishing **Today**)
- ▶ AVL trees (**Today** and next week Monday)
- ▶ Traditional red-black trees (next week Wednesday)
- ▶ Left-leaning red-black trees (next week Friday)
- ▶ “Wrap-up” BST (week-after Monday)

Today and Monday:

- ▶ Review BST basics and code base
- ▶ BST performance and the balanced BST problem
- ▶ Rotations; overview of solutions
- ▶ AVL tree definition
- ▶ AVL tree cases
- ▶ AVL tree performance

Coming up:

Catch up on older projects

*Do **BST rotations** project (due Mon, Oct 27)—nothing to turn in*

*Do **AVL trees** project (due Fri, Oct 31)*

*Due **Fri, Oct 24** (end-of-day)*

Read Section 5.(1 & 2)

Do Exercises 5.(2 & 6)

Take quiz (BST basics)

*Due **Mon, Oct 27** (end of day)*

Read Section 5.3

Do Exercises 5.(7 & 8)

Take quiz (AVL trees)

*Due **Mon, Nov 3** (end of day)—but spread it out*

Read Sections 5.(4-6)

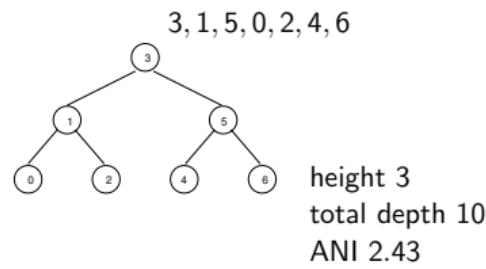
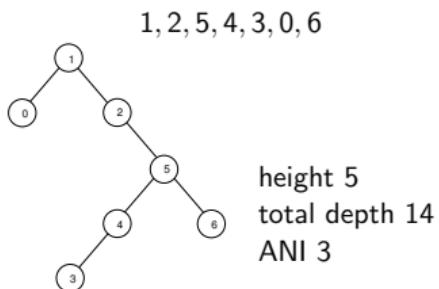
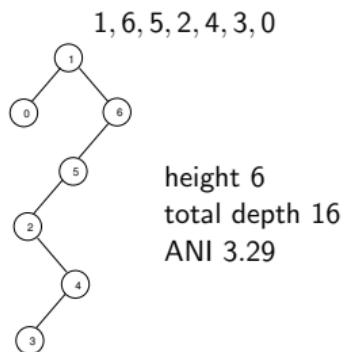
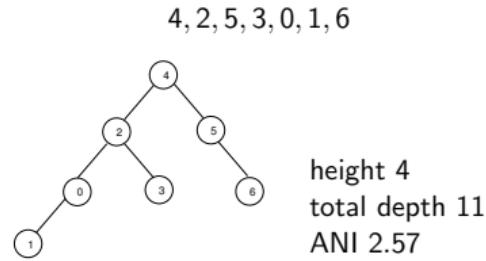
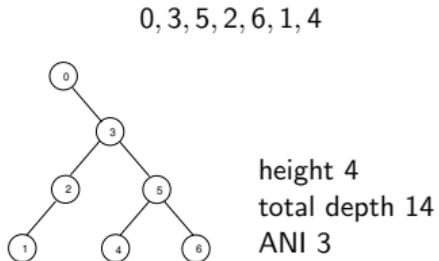
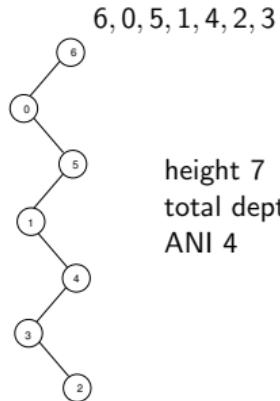
Do Exercise 5.13

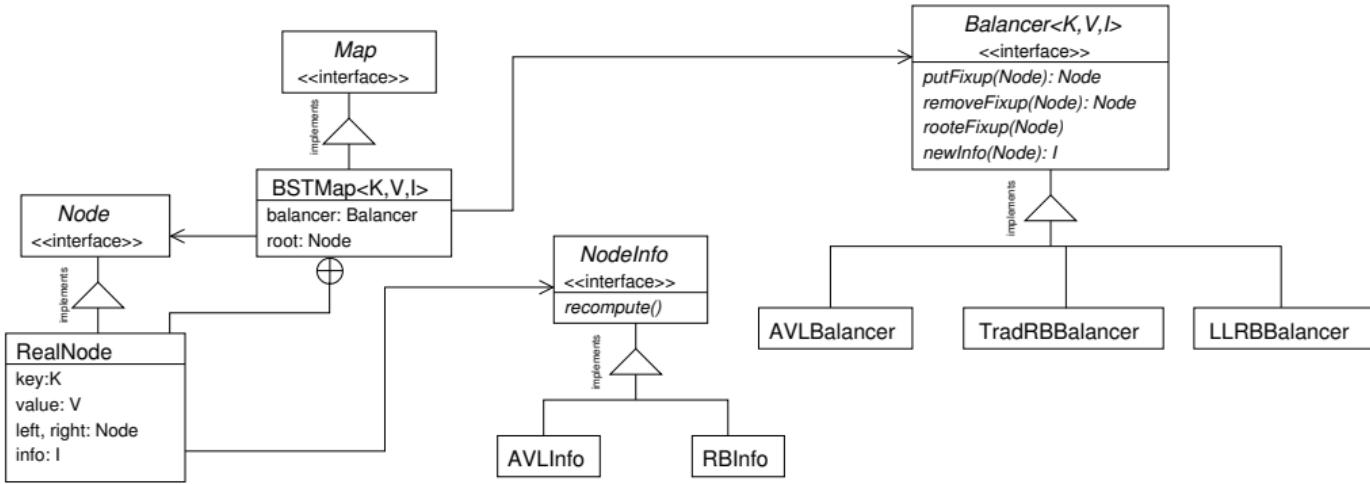
Take quiz (red-black trees)

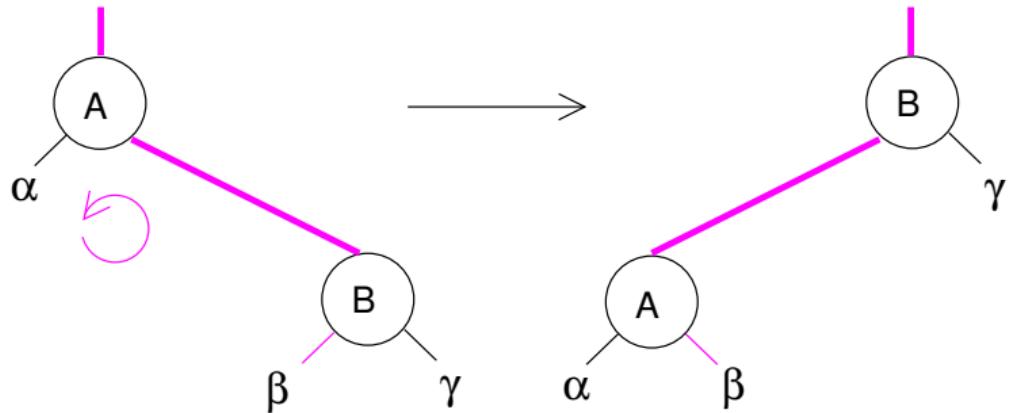
A **binary search tree** (BST) over some ordered key type is either

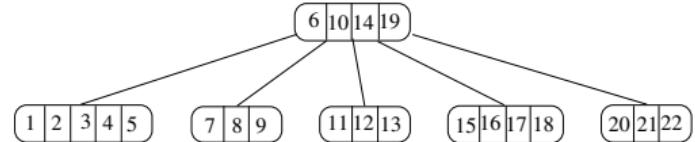
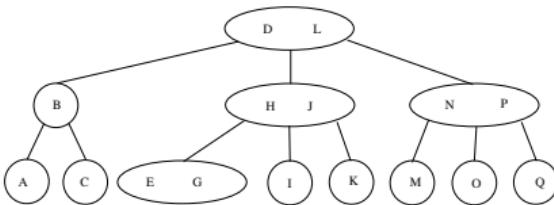
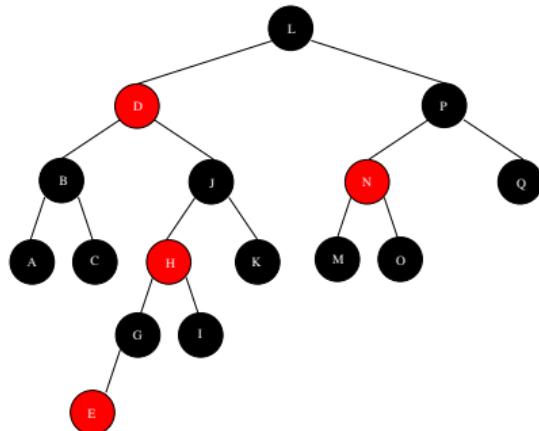
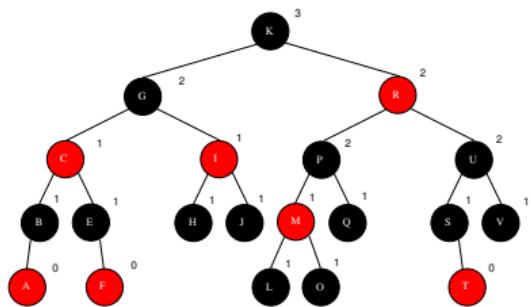
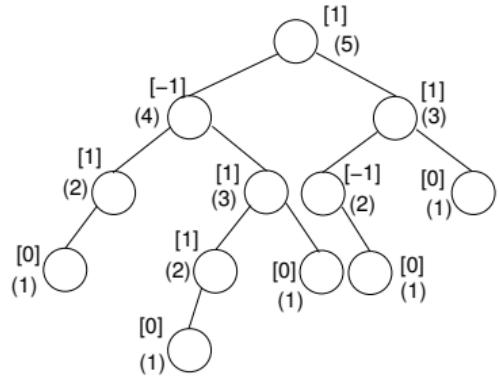
- ▶ empty, or
- ▶ a node augmented with a key k together with two children, designated *left* and *right*, such that
 - ▶ *left* is a binary search tree such that all of the keys in that tree are less than or equal to k , and
 - ▶ *right* is a binary search tree such that all of the keys in that tree are greater than or equal to k .

		Unsorted	Sorted
Array	Find	$\Theta(n)$	$\Theta(\lg n)$
	Insert	$\Theta(1)$ expected, $\Theta(n)$ worst	$\Theta(n)$
	Delete	$\Theta(n)$	$\Theta(n)$
Linked structure	Find	$\Theta(n)$	$\Theta(n)$
	Insert	$\Theta(1)$	$\Theta(1)$
	Delete	$\Theta(1)$	$\Theta(1)$









The *height* of a node (or (sub)tree) is the maximum number of nodes on any path from that node to any leaf, inclusive.

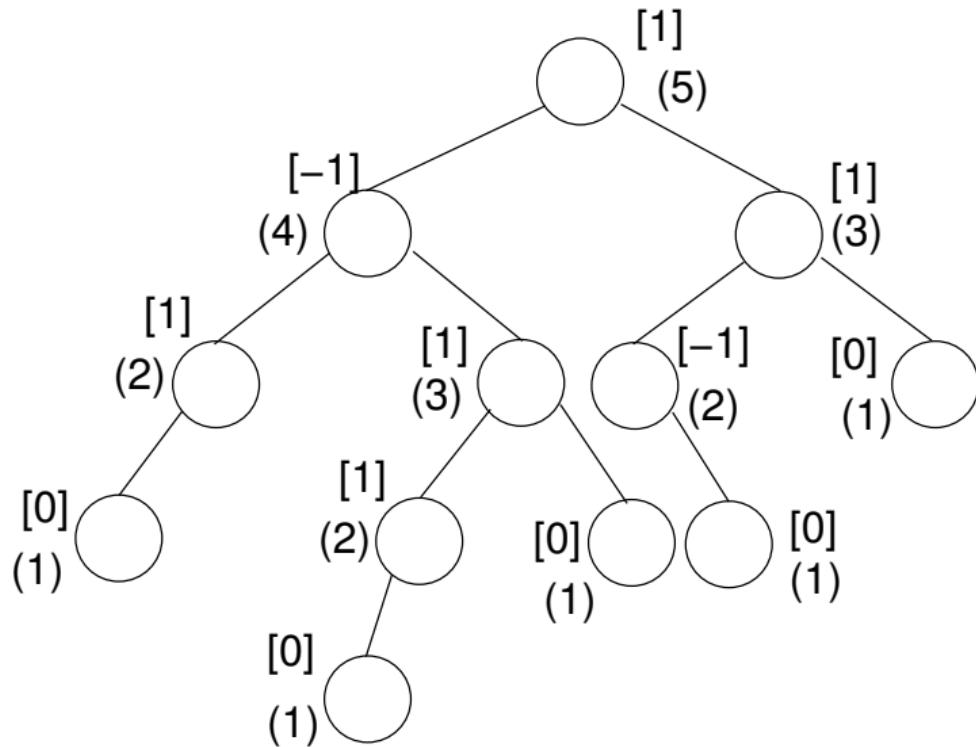
$$\text{height}(c) = \begin{cases} 0 & \text{if } c \text{ is null} \\ \max(\text{height}(c.\ell), \text{height}(c.r)) + 1 & \text{otherwise} \end{cases}$$

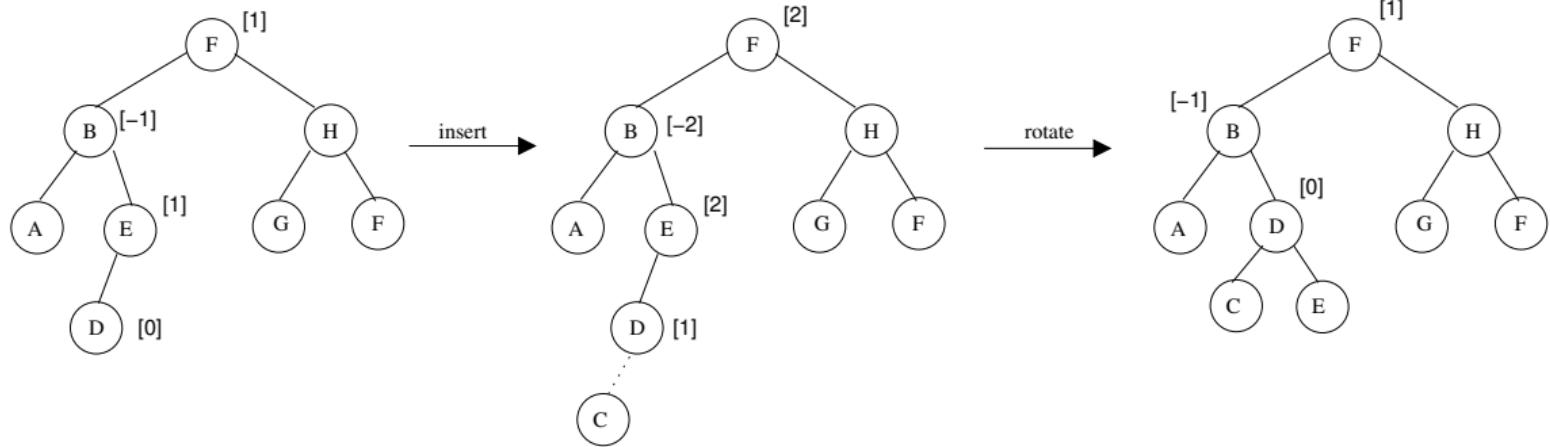
The *balance* of a node is the difference between the heights of its left and right children. In an AVL tree, each node's subtrees' heights must differ by at most 1:

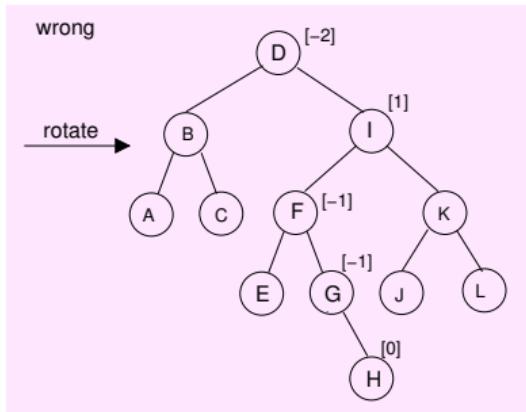
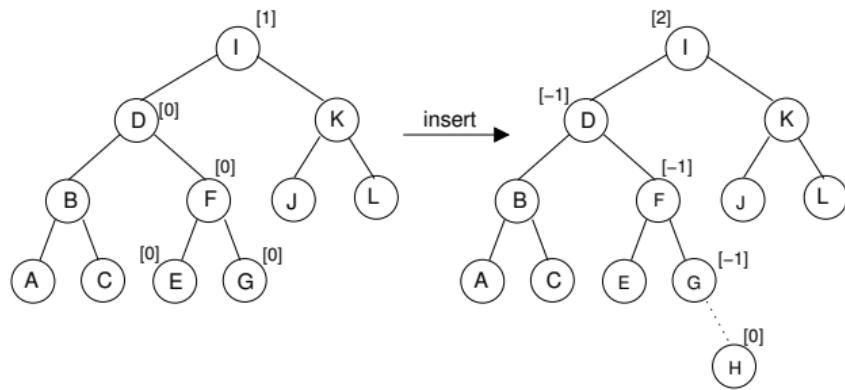
$$\forall x \in \text{nodes}, |\text{height}(x.\text{left}) - \text{height}(x.\text{right})| \leq 1$$

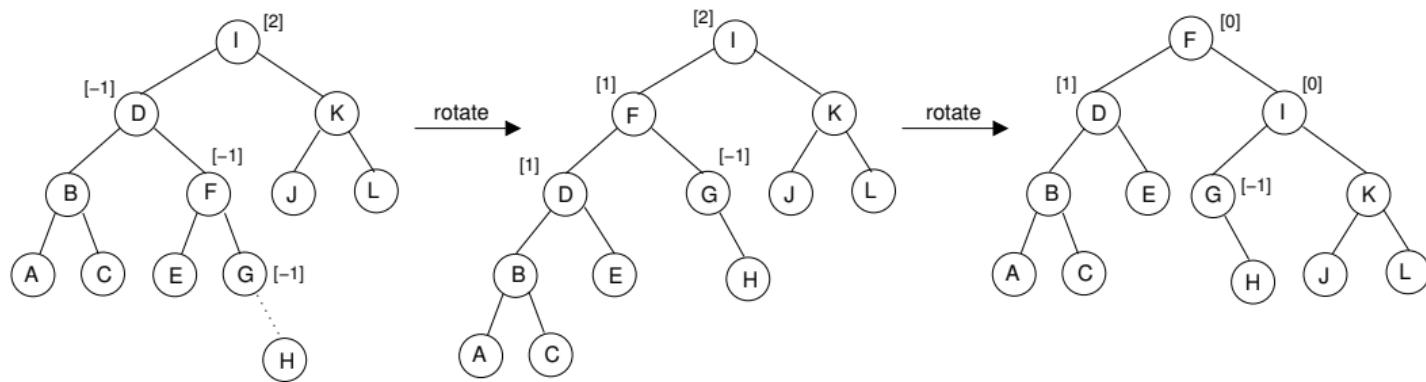
A node that has balance 1 or -1 has a *bias*. A node that (temporarily) has balance 2 or -2 is in *violation*.

(A balance less than -2 or greater than 2 shouldn't happen even temporarily.)

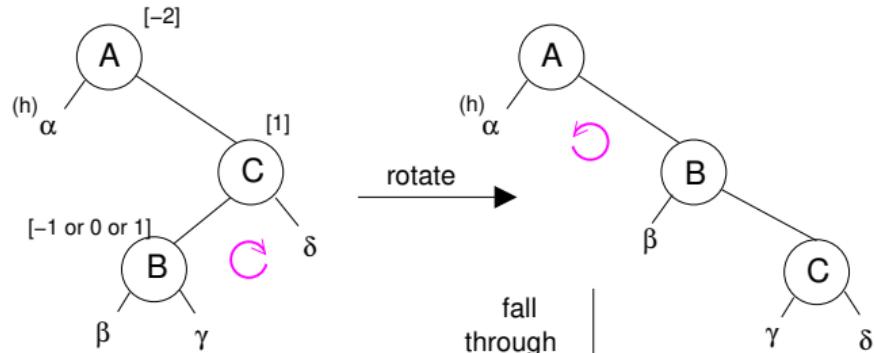




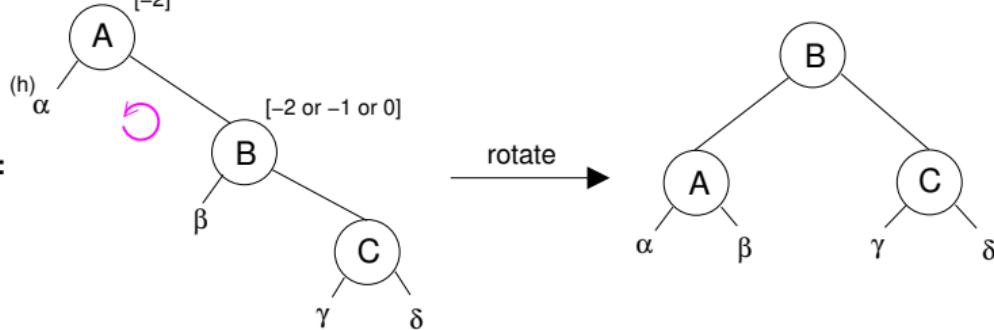




Right-Left:



Right-Right:



Invariant 30 (Postconditions of `RealNode.put()` with `AVLBalancer`.)

Let x be the root of a subtree on which `put()` is called and y be the node returned, that is, the root of the resulting subtree. The subtree rooted at y has no violations and the height of the subtree rooted at y is equal to or one greater than the original height of the subtree rooted at x .

Proof. *Suppose `put()` is called on node x in a BST using AVL balancing which has no violations. There are three cases: x is null, x is a `RealNode` containing the key being searched for, or x is a `RealNode` with a different key. We use structural induction with the first two cases as base cases.*

Base case 1. Suppose x is null, which has height 0. Then the node y returned is a new *RealNode* with null as both children, height 1, and balance 0. The subtree rooted at y has no violations and height one greater than the original height of x .

Base case 2. Suppose x is a *RealNode* whose key is equal to the key used for this *put()*. Then the value at node x is overwritten but node x itself is returned (so $y = x$ in this case) with the tree structure unchanged.

Inductive case. Suppose x is a *RealNode* and, without loss of generality, the key used for this *put()* is greater than the key at x , and so *put()* is called on the right child of x . Let h_0 be the height of the tree rooted at x before this call to *put()* on the right child, and let z the root of the subtree that results from this call to *put()* on the right child. Our inductive hypothesis is that the subtree rooted at z has no violations and that its height is equal to or one greater than the height of the original right child of x .

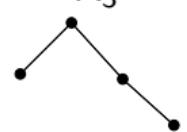
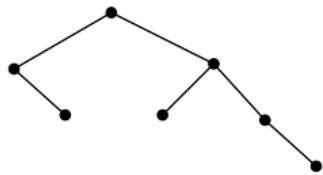
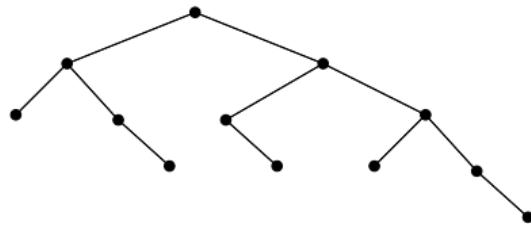
Let h_1 be the height of the tree rooted at x after the call to `put()` on the right child but before the call to `putFixup()` with x .

Since since at most the height of its right subtree has increased by one, either $h_1 = h_0$ or $h_1 = h_0 + 1$. By supposition, the balance of x before the call to `put()` was no less than -1 , since we supposed the tree had no violations. Since at most the height of its right subtree has increased by one, the balance of x is now no less than -2 . We now have two subcases: Either the balance of x is greater than -2 or it is equal to -2 .

Suppose the balance of x is greater than -2 . Then the call to `putFixup()` with x returns x unchanged, which is also returned as the result of `put()` (again $y = x$), a tree with no violations and height h_1 .

On the other hand, suppose the balance of x is equal to -2 . Then y is a node other than x returned by `putFixup()`. Let h_2 be the height of the subtree rooted at y when `putFixup()` returns. By inspection of the right-right and right-left subcases given above, the subtree rooted at y has no violations and either $h_2 = h_1$ or $h_2 = h_1 - 1$. In either of those cases $h_2 = h_0$ or $h_2 = h_0 + 1$.

□

A_1  A_2  A_3  A_4  A_5  A_h  A_{h-2} A_{h-1}

$$B_h = \begin{cases} 1 & \text{if } h = 1 \\ 2 & \text{if } h = 2 \\ B_{h-2} + B_{h-1} + 1 & \text{otherwise} \end{cases} \quad B_h + 1 = \begin{cases} 2 & \text{if } h = 1 \\ 3 & \text{if } h = 2 \\ (B_{h-2} + 1) + (B_{h-1} + 1) & \text{otherwise} \end{cases}$$

h	1	2	3	4	5	6
$B_h + 1$	2	3	5	8	13	21
B_h	1	2	4	7	12	20

$$B_h + 1 > \frac{\phi^{h+2}}{\sqrt{5}} - 1$$

$$B_h + 2 > \frac{\phi^{h+2}}{\sqrt{5}}$$

$$\sqrt{5}(B_h + 2) > \phi^{h+2}$$

$$\phi^{h+2} < \sqrt{5}(B_h + 2)$$

$$\log_{\phi}(\phi^{h+2}) < \log_{\phi}(\sqrt{5}(B_h + 2))$$

$$h + 2 < \log_{\phi}(\sqrt{5}(B_h + 2))$$

$$h < \log_{\phi}(\sqrt{5}(B_h + 2)) - 2$$

$$= \log_{\phi}(B_h + 2) + \log_{\phi}\sqrt{5} - 2$$

$$= \frac{1}{\lg \phi} \lg(B_h + 2) + \log_{\phi}\sqrt{5} - 2$$

Coming up:

Catch up on older projects

*Do **BST rotations** project (due Mon, Oct 27)—nothing to turn in*

*Do **AVL trees** project (due Fri, Oct 31)*

*Due **Fri, Oct 24** (end-of-day)*

Read Section 5.(1 & 2)

Do Exercises 5.(2 & 6)

Take quiz (BST basics)

*Due **Mon, Oct 27** (end of day)*

Read Section 5.3

Do Exercises 5.(7 & 8)

Take quiz (AVL trees)

*Due **Mon, Nov 3** (end of day)—but spread it out*

Read Sections 5.(4-6)

Do Exercise 5.13

Take quiz (red-black trees)