CSE 332: Data Structures & Parallelism

Lecture 10: AVL Trees (part 2)

Ruth Anderson

Autumn 2025

Administrative

- EX03 – On Recurrences, Due Fri Oct 17
- EX04 – On AVL Trees: programming, Due Fri Oct 24
- Mid Quarter Survey, Due Sat Oct 18
- "Meet the Staff" activity
  - Sometime during the first 4-5 weeks of class, visit a CSE 332 office hour (in person or on zoom)
- Lecture MegaThread in Ed Discussion

Today

- Dictionaries
  - AVL Trees
    - AVL Rotations
    - Height bound on AVL Trees

The AVL Balance Condition:

Left and right subtrees of *every node*
have *heights* **differing by at most 1**

Define: **balance**(x) = height(x.left) – height(x.right)

AVL property: **$-1 \le$ balance(x) ≤ 1, for every node x**

- Ensures small depth
  - Will prove this by showing that an AVL tree of height h must have a lot of (*roughly* 2^h) nodes
- Easy to maintain
  - Using single and double rotations

Note: height of a null tree is -1, height of tree with a single node is 0

AVL tree insert

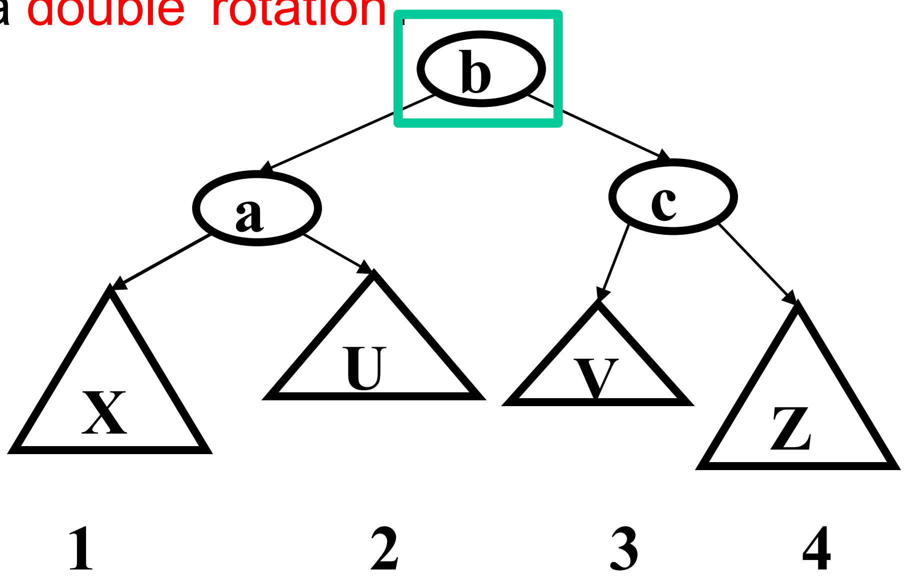
Let *b* be the node where an imbalance occurs.

Four cases to consider. The insertion is in the

1. left subtree of the left child of *b.*
2. right subtree of the left child of *b.*
3. left subtree of the right child of *b.*
4. right subtree of the right child of *b.*

Idea: Cases 1 & 4 are solved by a single rotation.

Cases 2 & 3 are solved by a double rotation.



How Long Does Rebalancing Take?

- Assume we store in each node the height of its subtree.
- How do we find an unbalanced node?

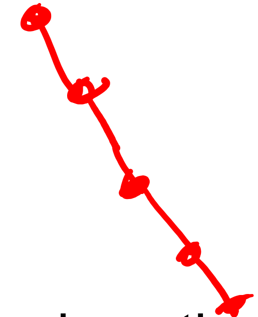
- How many rotations might we have to do?

How Long Does Rebalancing Take? (answer)

- Assume we store in each node the height of its subtree.
- How do we find an unbalanced node?
  - Just go back up the tree from where we inserted.

- How many rotations might we have to do?
  - Just a single or double rotation on the lowest unbalanced node.
  - A rotation will cause the subtree rooted where the rotation happens to have the same height it had before insertion.

Where Were We?



- We used rotations to restore the AVL property after insertion.
- If h is the height of an AVL tree:
  - It takes $O(h)$ time to find an imbalance (if any) and fix it.
  - So the worst case running time of insert? $\Theta(h)$.
- Is h always $O(\log n)$? YES! These are all $\Theta(\log n)$. Let's prove it!

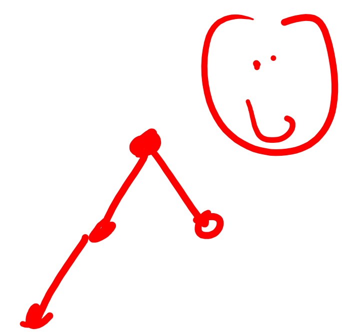
- Notice: for a given height h:
  - I would prefer a tree that stores as many nodes as possible
  - My adversary would like it to store as **few nodes as possible**.

Bounding the Height

- Suppose you have a tree of height h, meeting the AVL condition:

AVL condition: For every node, the height of its left subtree and right subtree differ by at most 1.

- What is the **minimum number of nodes** in the tree?



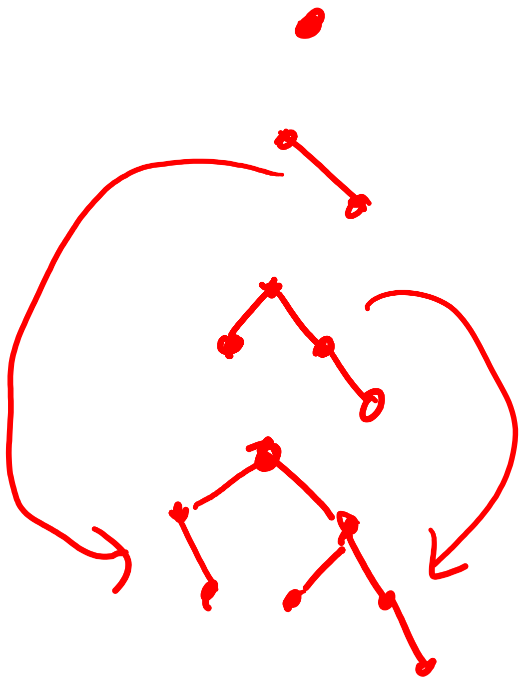
- If $h = 0$, then 1 node
- If $h = 1$, then 2 nodes.

- In general?

Let $S(h)$ be the minimum # of nodes in an AVL tree of height h, then:

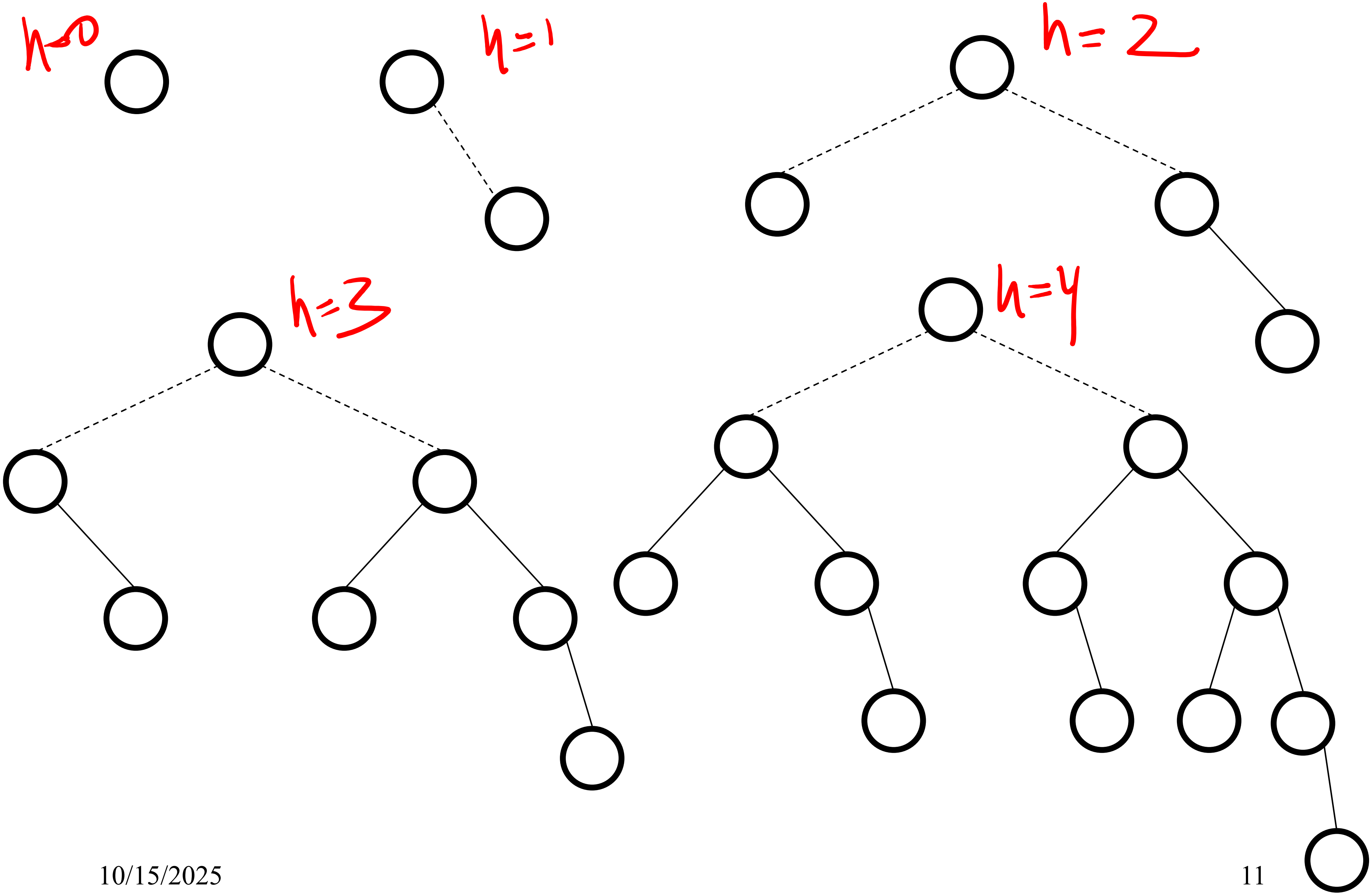
example of

h	***Minimal AVL Tree***	***S(h)***
0		1
1		2
2		4
3		7

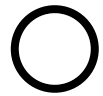


+1

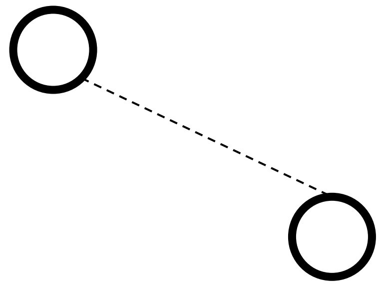
Minimal AVL Trees



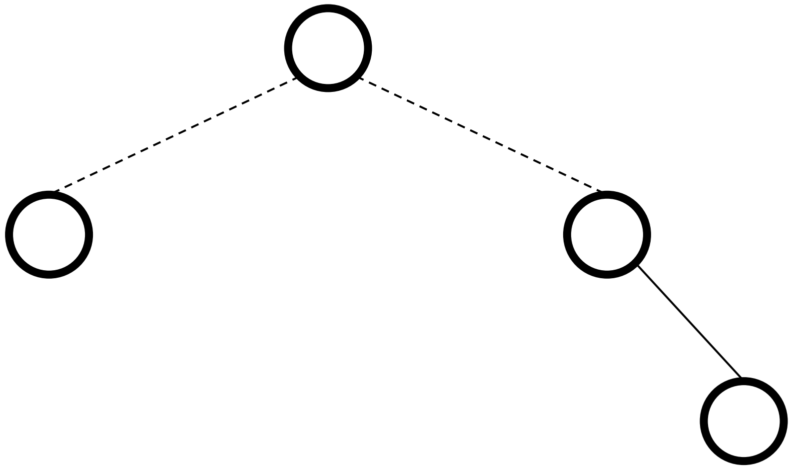
Minimal AVL Tree (height = 0)



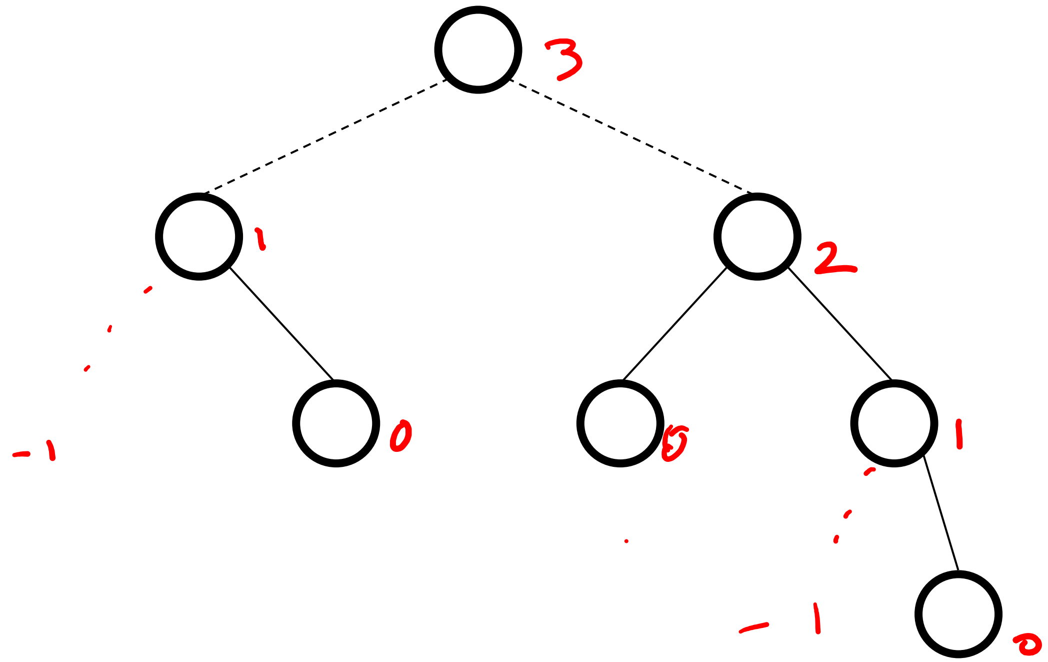
Minimal AVL Tree (height = 1)



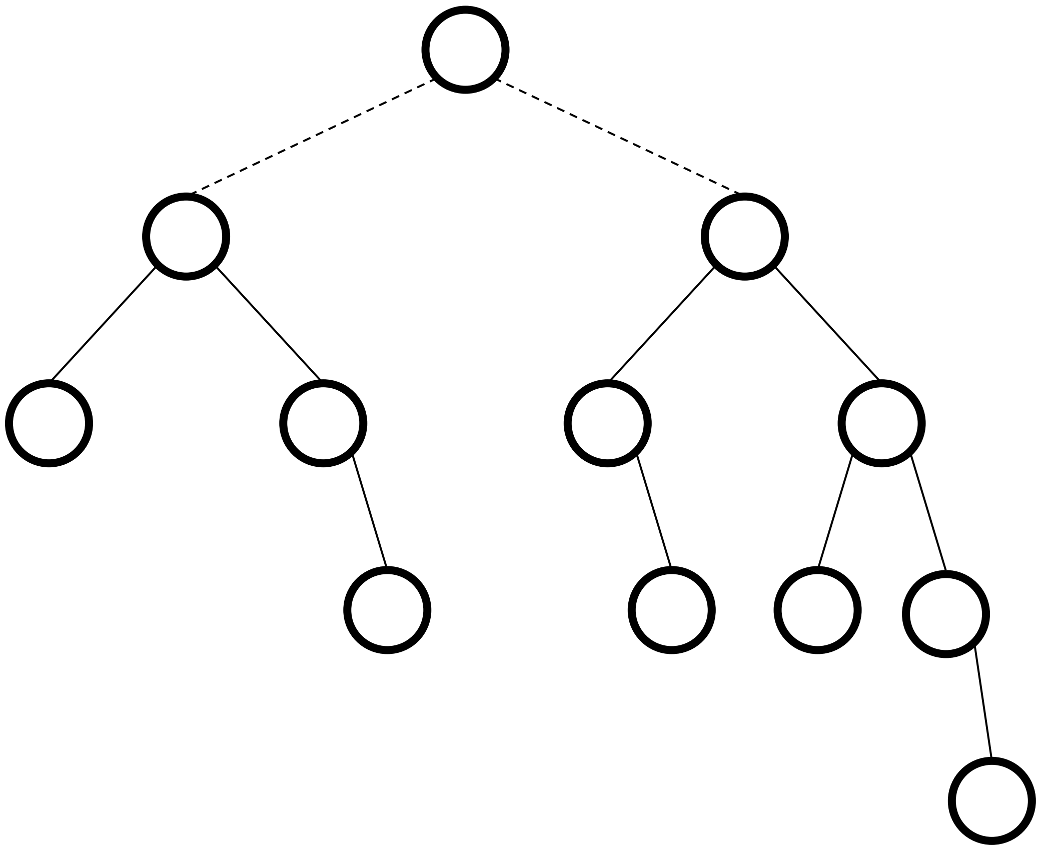
Minimal AVL Tree (height = 2)



Minimal AVL Tree (height = 3)



Minimal AVL Tree (height = 4)



Height of an AVL Tree?

Using the AVL balance property, we can determine the minimum number of nodes in an AVL tree of height *h*

Let $\mathbf{S}(h)$ be the minimum # of nodes in an AVL tree of height *h*, then:

$$\mathbf{S}(h) = \mathbf{S}(h-1) + \mathbf{S}(h-2) + 1$$

where $\mathbf{S}(-1) = 0$ and $\mathbf{S}(0) = 1$

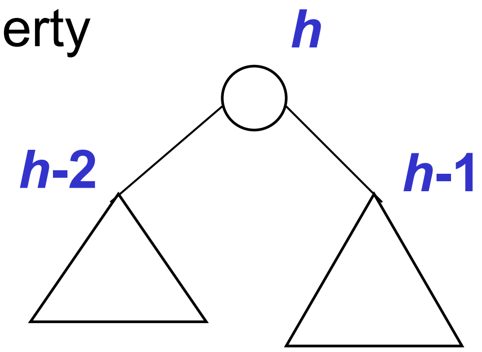
Solution of Recurrence: $\mathbf{S}(h) \approx 1.62^h$

The shallowness bound

Let $S(h)$ = the minimum number of nodes in an AVL tree of height h

– If we can prove that $S(h)$ grows exponentially in h, then a tree with n nodes has a logarithmic height

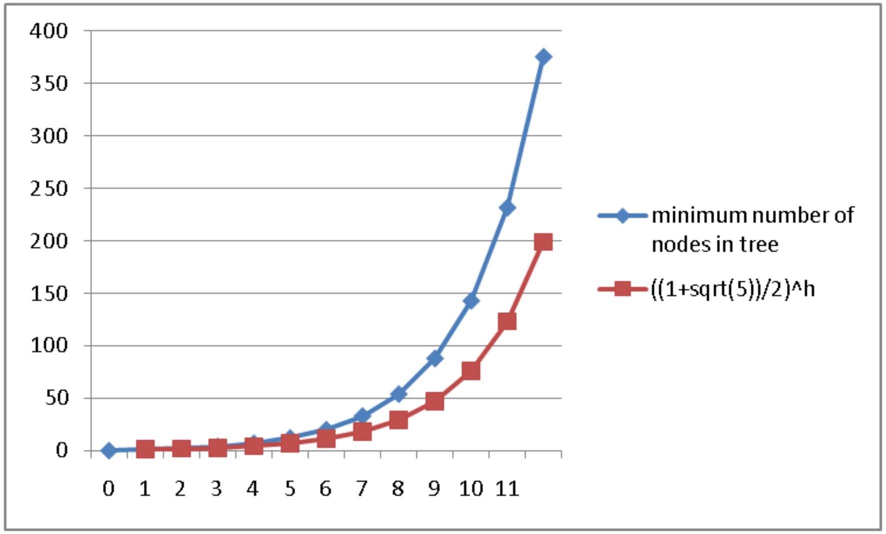
- Step 1: Define $S(h)$ inductively using AVL property
  – $S(-1)=0$, $S(0)=1$, $S(1)=2$
  – *For* $h \geq 1$, $S(h) = 1+S(h-1)+S(h-2)$

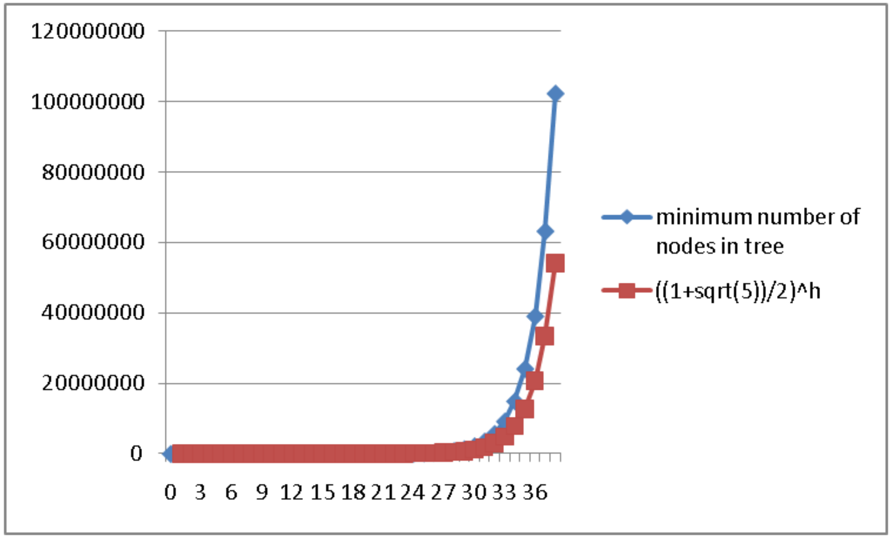


- Step 2: Show this recurrence grows really fast
  – Similar to Fibonacci numbers
  – Can prove for all h, $S(h) > \phi^h - 1$ where ϕ is the golden ratio, $(1+\sqrt{5})/2$, about 1.62
  – Growing faster than 1.6^h is "plenty exponential"

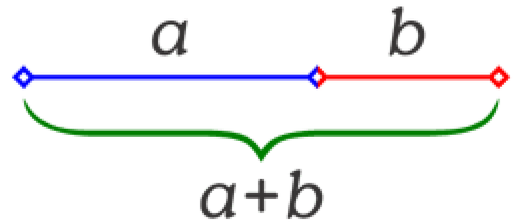
Aside: Before we prove it

- Good intuition from plots comparing:
  - *S*(*h*) computed directly from the definition
  - $((1+\sqrt{5})/2)^{h}$
- *S*(*h*) is always bigger, up to trees with huge numbers of nodes
  - Graphs aren't proofs, so let's prove it





Aside: The Golden Ratio



$a+b$ is to a as a is to b

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.62$$

This is a special number

- Since the Renaissance, many artists and architects have proportioned their work (e.g., length:height) to approximate the ***golden ratio***: If **`(a+b)/a = a/b`**, then **`a = ϕb`**

- We will need one special arithmetic fact about ϕ :

$$\begin{aligned}
\phi^2 &= ((1+5^{1/2})/2)^2 \\
&= (1 + 2*5^{1/2} + 5)/4 \\
&= (6 + 2*5^{1/2})/4 \\
&= (3 + 5^{1/2})/2 \\
&= 1 + (1 + 5^{1/2})/2 \\
&= 1 + \phi
\end{aligned}$$

The proof

$S(-1)=0$, $S(0)=1$, $S(1)=2$
For $h \geq 1$, $S(h) = 1+S(h-1)+S(h-2)$

Theorem: For all $h \geq 0$, $S(h) > \phi^{h} - 1$

Proof: By induction on h

Base cases:

$S(0) = 1 > \phi^{0} - 1 = 0$ $\qquad$ $S(1) = 2 > \phi^{1} - 1 \approx 0.62$

Inductive case ($k > 1$):

Show $S(k+1) > \phi^{k+1} - 1$ assuming $S(k) > \phi^{k} - 1$ and $S(k-1) > \phi^{k-1} - 1$

$\mathbf{S(k+1)}$	$= 1 + S(k) + S(k-1)$	by definition of S
	$> 1 + \phi^{k} - 1 + \phi^{k-1} - 1$	by induction
	$= \phi^{k} + \phi^{k-1} - 1$	by arithmetic (1-1=0)
	$= \phi^{k-1} (\phi + 1) - 1$	by arithmetic (factor ϕ^{k-1})
	$= \phi^{k-1} \phi^{2} - 1$	by special property of ϕ
	$= \boldsymbol{\phi^{k+1} - 1}$	by arithmetic (add exponents)

Now efficiency

- Worst-case complexity of **`find`**: __________
  - Tree is balanced

- Worst-case complexity of **`insert`**: __________
  - Tree starts balanced
  - A rotation is $O(1)$ and there's an $O(\log n)$ path to root
  - (Same complexity even without one-rotation-is-enough fact)
  - Tree ends balanced

- **`delete?`** (see 3 ed. Weiss) requires more rotations: _________
- Lazy deletion? ________________

Now efficiency

- Worst-case complexity of **`find`**: $O(\texttt{log}\ n)$
  - Tree is balanced

- Worst-case complexity of **`insert`**: $O(\texttt{log}\ n)$
  - Tree starts balanced
  - A rotation is $O(1)$ and there's an $O(\texttt{log}\ n)$ path to root
  - (Same complexity even without one-rotation-is-enough fact)
  - Tree ends balanced

- **`delete?`** (see 3 ed. Weiss) requires more rotations: $O(\texttt{log}\ n)$

Pros and Cons of AVL Trees

Arguments for AVL trees:

1. All operations logarithmic worst-case because trees are *always* balanced
2. Height balancing adds no more than a constant factor to the speed of **`insert`** and **`delete`**

Arguments against AVL trees:

1. Difficult to program & debug
2. More space for height field
3. Asymptotically faster but rebalancing takes a little time