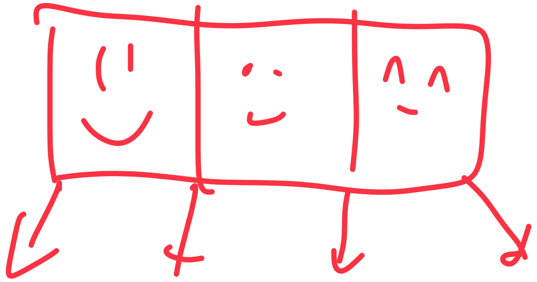
Data Structures

AVL Tree Proof (and ~~BTree~~)

Didnt get to!

CS 225
Brad Solomon & G Carl Evans

October 2, 2023



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

Review and finish AVL proof

weakness?

Discuss alternatives to BSTs

AVL Tree Analysis

Balanced BST

why BBST over BST

$O(h) = O(n)$:(

For an AVL tree of height h:

Find runs in: $O(h)$.

Insert runs in: $O(h)$.

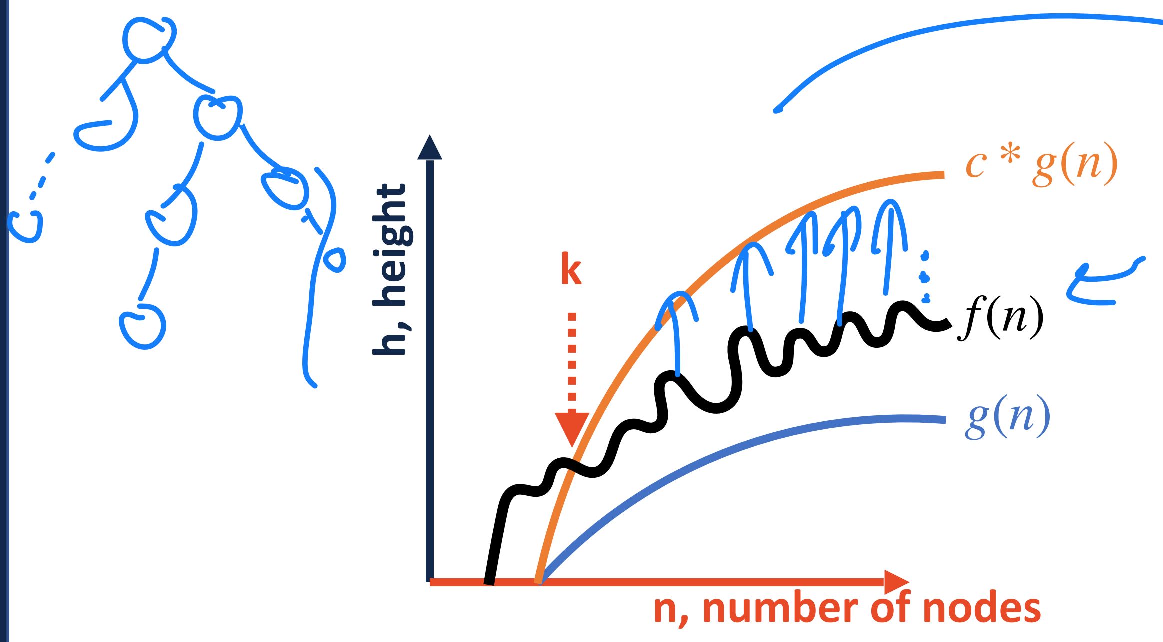
Remove runs in: $O(h)$.

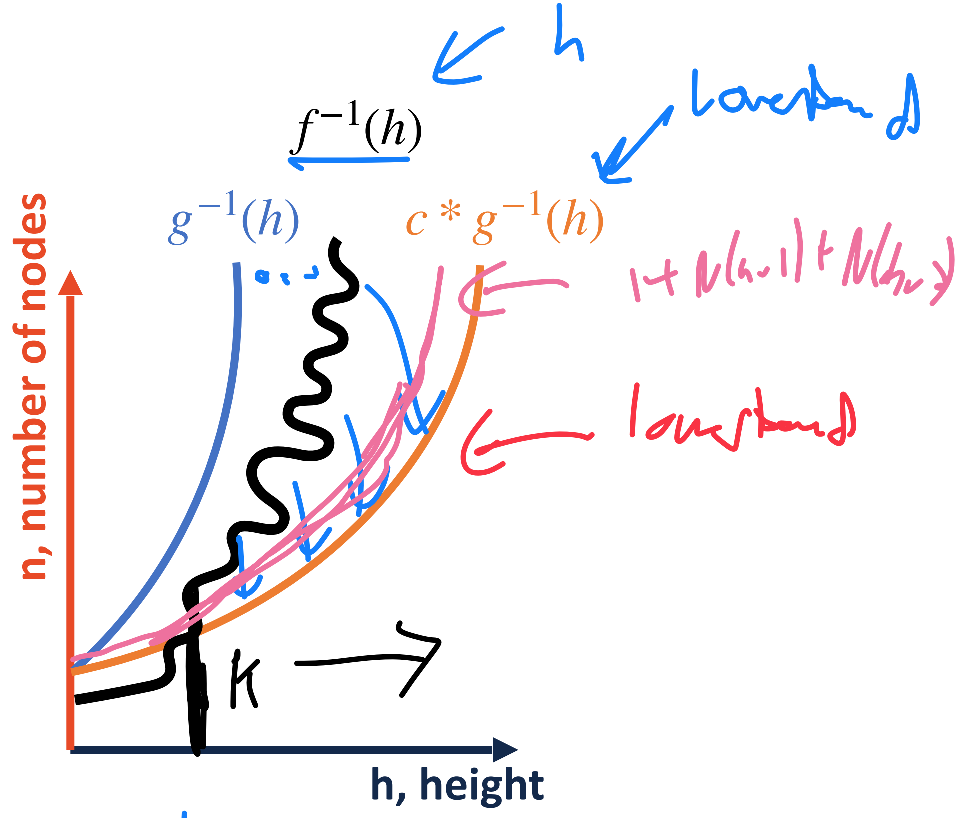
$O(\log n)$:)

$O(h) =$

Claim: The height of the AVL tree with n nodes is: $O(\log(n))$.

AVL Tree Analysis





upper bound

$f(n)$ = "Tree height given nodes"

lower bound

$f^{-1}(h)$ = "Nodes in tree given height"

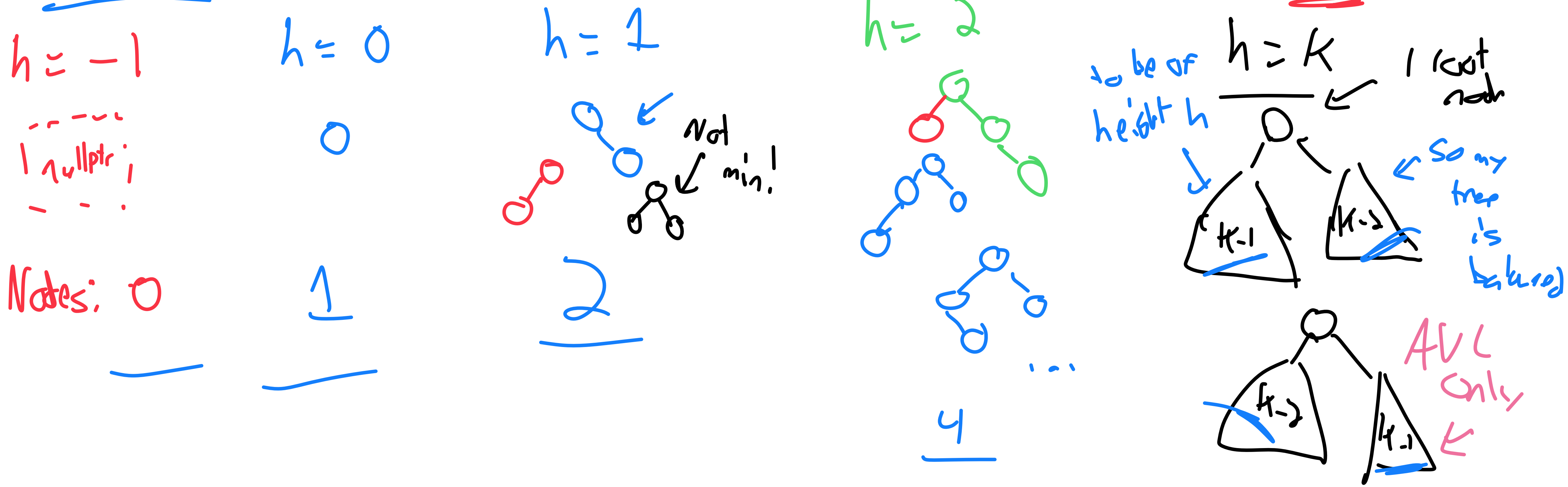
The number of nodes in the tree, $\mathbf{f^{-1}(h)}$, will always be <u>greater than</u> **c × g⁻¹(h)** for all values where **n > k**.

Plan of Action

→ Proof By Induction if we can make a recurrence relation [N(h) ≈ N(h-1)]

Since our goal is to find the lower bound on **n** given **h**, we can begin by defining a function given **h** which describes the smallest number of nodes in an AVL tree of height **h**:

$N(h)$ = minimum number of nodes in an AVL tree of height h



Simplify the Recurrence

Base Case $(N(-1) = 0)$

root   Subtree   Smaller subtree

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

$> N(h-1) + N(h-2)$ Don't want two terms

$> 2N(h-2)$

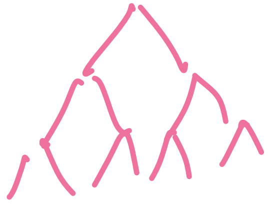
$$\boxed{N(h) > 2N(h-2)}$$

Not every step but every other step

$N(h) > N(h) - 1$ (simplify)

$N(h-1) > N(h-2)$ (replace $h-1$ w/ $h-2$)

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



2^h

1) Know my characteristic equations $\Rightarrow$ know answer

2) Unroll $N(h) > 2N(h-2) = 2(2N(h-4))\ldots$ $h-2-2 \Rightarrow h-4$

$2 \cdot 2 \cdot 2$

3) Intuit

$2^{h/2}$

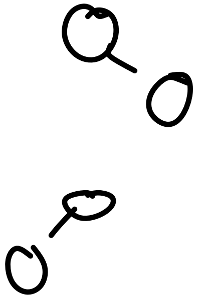
State a Theorem

Theorem: An AVL tree of height h has at least $2^{h/2}$ nodes.

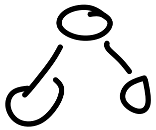
Proof by Induction:

I. Consider an AVL tree and let **h** denote its height.

II. Base Case: h = 1



2 nodes



3 nodes

2

compare real # of nodes w/ equation

→ eq should be lowerbound

plug in h=1

$2^{1/2}$

sqrt(2)

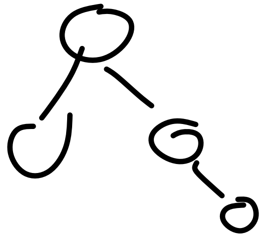
1.41

An AVL tree of height 1 has at least 1.41 nodes.

Prove a Theorem

$1 + \underbrace{N(h-1)}_{} + \underbrace{N(h-2)}_{}$ 2 base cases

III. Base Case: $h=2$



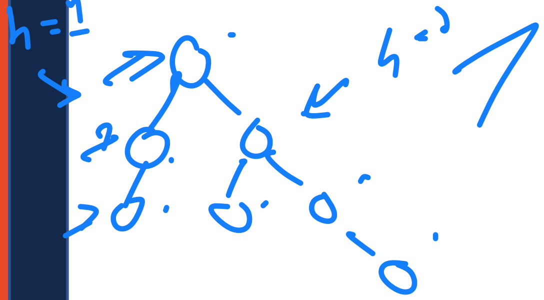
min is always 4

Eq says $2^{h/2}$

$\downarrow$

$2^{2/2} = 2$

$h=3$



$> 2^{3/2}$

eq holds twice

↳ B/c my real recurrence has two different values ✓

An AVL tree of height __2__ has at least __2__ nodes.

Prove a Theorem

IV. Induction Case: ________________

Assume for all heights $\boxed{i < h, N(i) \geq 2^{i/2}}$. Prove that $N(h) \geq 2^{h/2}$

$N(h-1) + N(h-2) > 2N(h-2)$ ← Is true!

Not true!

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

???

plug in $N(h-2)$ $(i = h-2)$

$> 2N(h-2)$

$> 2 \cdot 2^{(h-2)/2}$

$= 2 \cdot 2^{h/2 - 1 + 1}$

$2^{h/2}$:)

Goal

Thus

$N(h) \geq 2^{h/2}$

Prove a Theorem

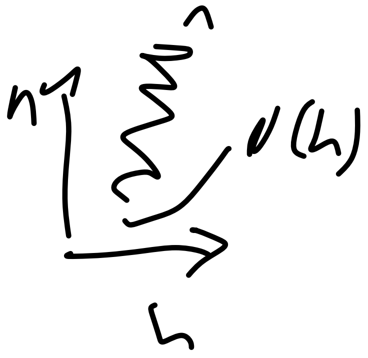
? $n > X$ strange claim $n \geq X$

V. Using a proof by induction, we have shown that:

min! $N(h) > 2^{h/2}$ (min # of nodes of tree of height h)

$n > N(h)$ n the number of nodes in any AVL tree

← lower bound not tight



...and inverting: Goal: Relate n & h

$n \geq 2^{h/2}$

$\log n \geq h/2 \rightarrow h \leq 2\log n$

$h = O(\log n)$

AVL proof!

AVL Runtime Proof

An upper-bound on the height of an AVL tree is **O(lg(n))**:

$N(h)$:= Minimum # of nodes in an AVL tree of height h

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

$> 1 + 2^{(h-1)/2} + 2^{(h-2)/2}$

$> 2 \times 2^{(h-2)/2} = 2^{(h-2)/2+1} = 2^{h/2}$

Theorem #1:

Every AVL tree of height h has at least $2^{h/2}$ nodes.

AVL Runtime Proof

An upper-bound on the height of an AVL tree is **O(lg(n))**:

proved before

$$\# \text{ of nodes } (n) \geq N(h) > 2^{h/2}$$

invert

$$n > 2^{h/2}$$

$$\lg(n) > h/2$$

$$2 \times \lg(n) > h$$

$$h < 2 \times \lg(n) \qquad , \textit{for } h \geq 1$$

Proved: The maximum number of nodes in an AVL tree of height h is less than 2 × lg(n).

Summary of Balanced BST

$\geq \log(n)$

AVL Trees

- Max height: 1.44 * lg(n) → All ops are O(log n)
- Rotations:

  → Zero rotations on find
  → One rotation on insert
  → O(**h**) == O(**lg(n)**) rotations on remove

Red-Black Trees → complicated!

- Max height: 2 * lg(n)
- Constant number of rotations on insert (max 2), remove (max 3).

Summary of Balanced BST

Pros:

- Running Time: O(log n)

  - Improvement Over: Linked List (if not insert/remove front) – O(n)

    Array → Sorted array* O(log n) find

    BST (unbalanced) O(n)

- Great for specific applications:

  ↳ Nearest neighbor (range find)

Summary of Balanced BST

Cons:

- Running Time:

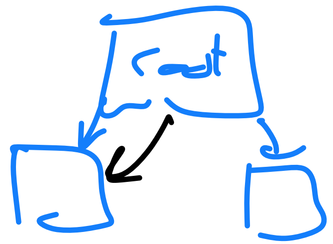
Pretty good! → $O(\log n)$

$O(n)$

$O(1)$

- In-memory Requirement:

↳ Assume that following pointers are efficient



$O(1)$

Considering hardware limitations

L1 RAM — faster
disk — ~
Online — super slow

Can we always fit our data in main memory?

↳ No!

Where else can we keep our data?

RAM – .0007 ms (100 ns)

Disk memory ~200x slower

Cloud – 40 ms latency

If we use these they are not O(1)

Does this match our assumption that all memory lookups are O(1)?

↳ No!

B-Tree Motivation

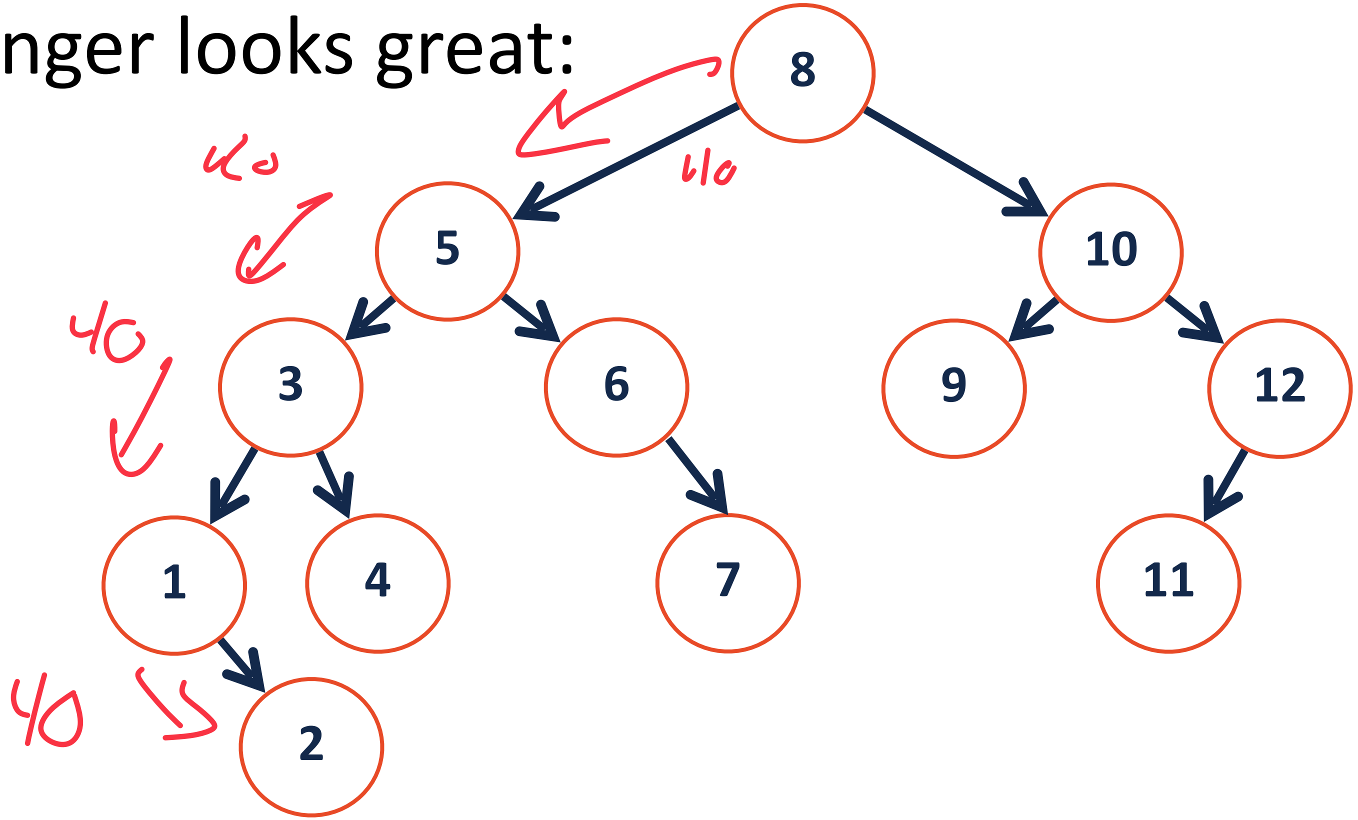
In Big-O we have assumed uniform time for all operations, but this isn't always true.

.0001 ms

However, seeking data from the cloud may take 40ms+.

…an O(lg(n)) AVL tree no longer looks great:

Big problem!



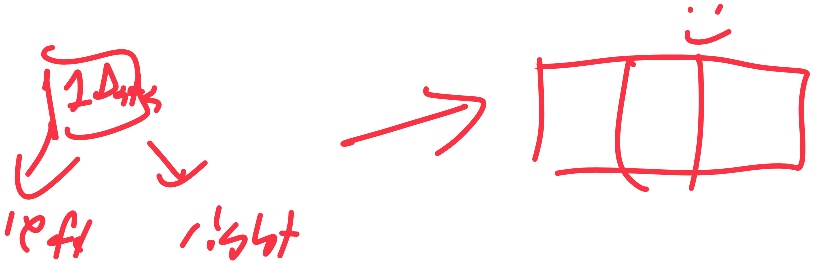
BTree Design Motivations

w/ a new tree

When large seek times become an issue, we address this by:

→ 10 ms

1) Pack node w/ more data



2) Other suggestions?

Tree as Vector? → Friday? Next week?