Announcements

1. MP Mosaics EC due Today!
2. Exam 2 Grades In Process
3. Extra Credit Survey due Wednesday
    a. 10 more people to hit threshold!

Extra Credit Survey!

Join Code: **225**

Warm-Up Question: What is the minimum number of nodes needed to make an AVL tree with height 3?

AVL Tree Analysis

Learning Objectives

1. Prove the upper bound on an AVL Tree's Height
2. Enumerate the runtime of Insert, Find, Remove, and Traverse for every Data Structure Seen so far

AVL Tree Analysis

We know: insert, remove and find runs in: $O(h)$.

We will show that: h is $O(\log n)$.

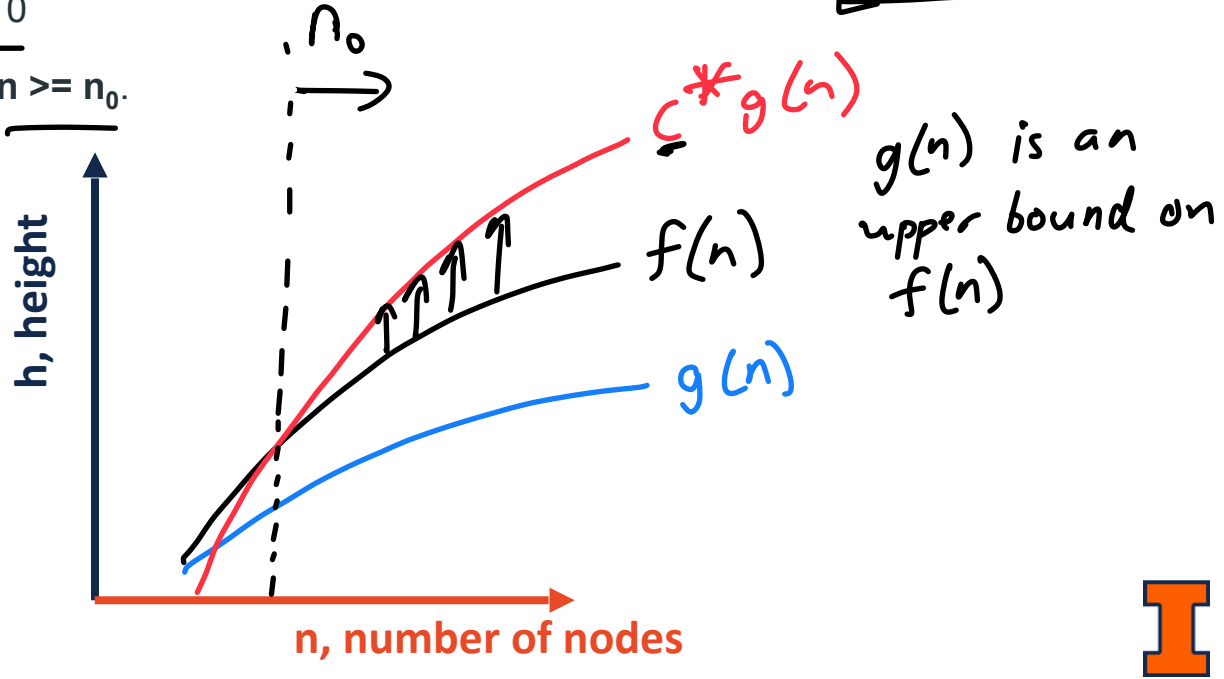
Goal: Find the lowest upper bound, big O, on the maximum height of an AVL tree given the number of nodes.

$f(n) \rightarrow h$

Big O - Lowest Upper Bound

Definition of big-O: Given two functions **f(n)** and **g(n)**, we say that **f(n)** is **O(g(n))** if there exist constants **c > 0** and **n_0 >= 0**

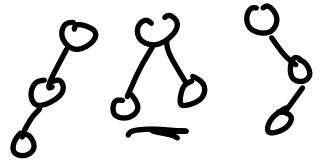
such that **f(n) <= c*g(n)** for all **n >= n_0**.



…or, with pictures:

Q: What is max h of AVL tree given n

n	h
0	-1
1	0
2	1
3	1



New Goal

Goal: Find the lowest upper bound, big O, on the maximum height of an AVL tree given the number of nodes.

Old Output

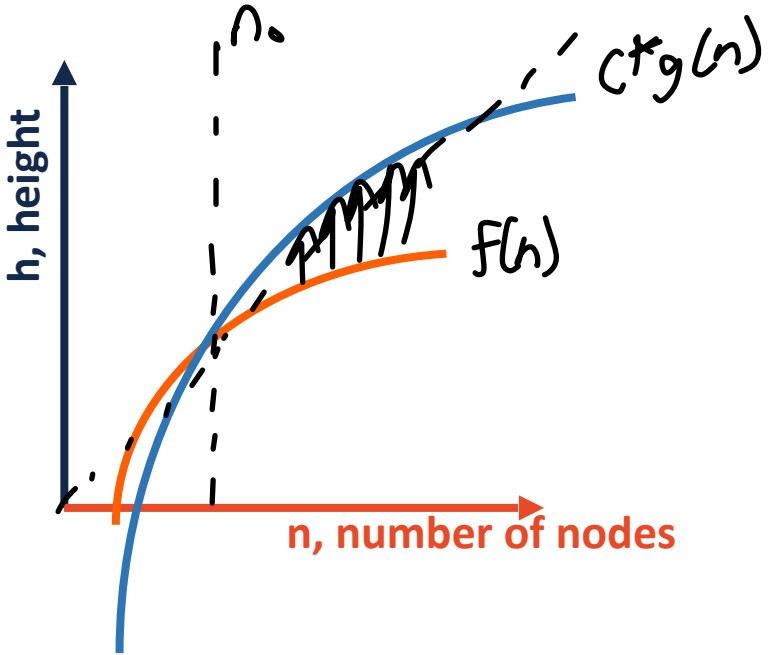
Old Inputs

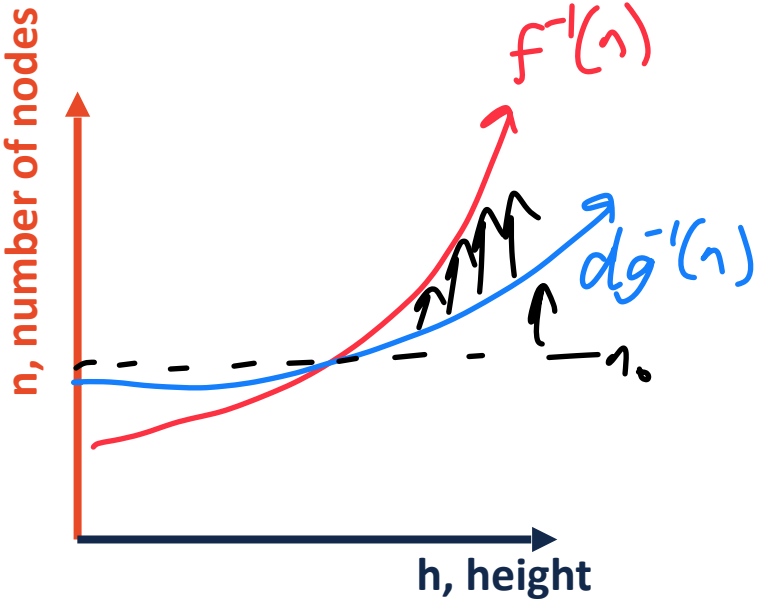
New Goal: Find the highest lower bound on the minimum number of nodes in an AVL tree given the height, N(h)

New Output

New Input

Invert and Find Highest Lower Bound

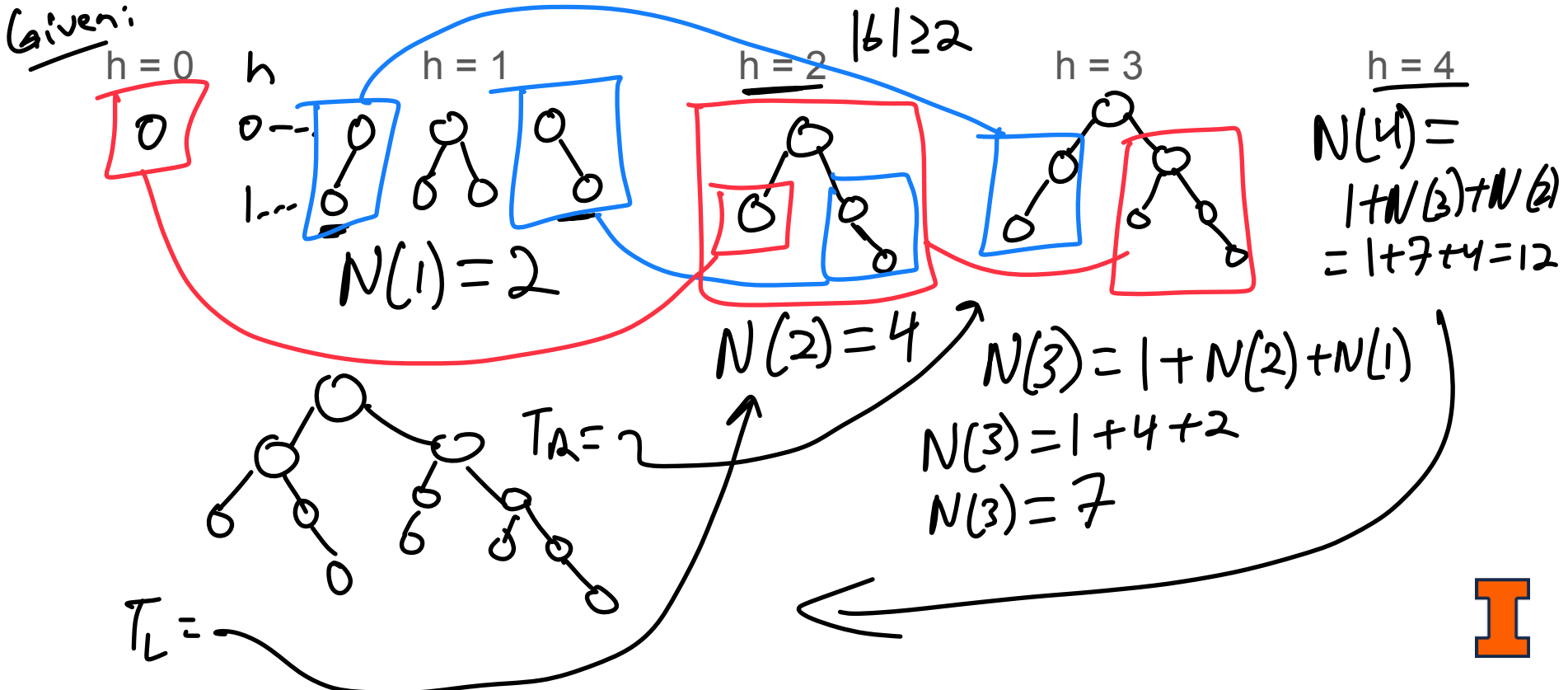




N(h) - # of nodes at minimum in an AVL of height h

Small Examples

1:00



Recurrence

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

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

3 $N(h) > 2 * N(h-2)$

4 $N(h) > 2 * 2 * N(h-4)$

5 $N(h) > 2^{h/2}$

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

$n_h > 2^{h/2}$

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

$N(h-2) > 2 * N(h-4)$

n_h - # of nodes in any AVL w/ height h

$\log(n_h) > \log(2^{h/2})$

$\log(n_h) > h/2$

$2 \log(n_h) > h$

$h = O(\log n)$

Conclusion

$h = O(\log(n))$

Summary

AVL Trees

Max height: 1.44 * log(n)

Rotations: Always rotate when an imbalance is found

Find is faster, because its shorter

Red-Black Trees

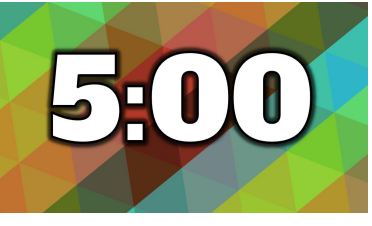
Max height: 2 * log(n)

Constant number of rotations on insert, remove, and find

Insert/remove faster

	Unsorted Array	Sorted Array	Unsorted List	Sorted List	Binary Tree	BST	AVL
Find	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$	$O(h) = O(n)$	$O(h) = O(\log n)$
Insert	$O(1)/O(n)$*	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(h) = O(n)$	$O(h) = O(\log n)$
Remove	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(h) = O(n)$	$O(h) = O(\log n)$
Traverse	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Corrections/Clarifications post lecture



	Unsorted Array	Sorted Array	Unsorted List	Sorted List	Binary Tree	BST	AVL
Find	O(n)	O(log n)	O(n)	O(n)	O(n)	O(h)/O(n)	O(h)/O(log n)
Insert	O(n)/O*(1)	O(n)	O(1)	O(n)	O(1)	O(h)/O(n)	O(h)/O(log n)
Remove	O(n)	O(n)	O(n)	O(n)	O(n)	O(h)/O(n)	O(h)/O(log n)
Traverse	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)	O(n)