Data Structures
AVL Analysis

CS 225
Brad Solomon & G Carl Evans

September 27, 2023

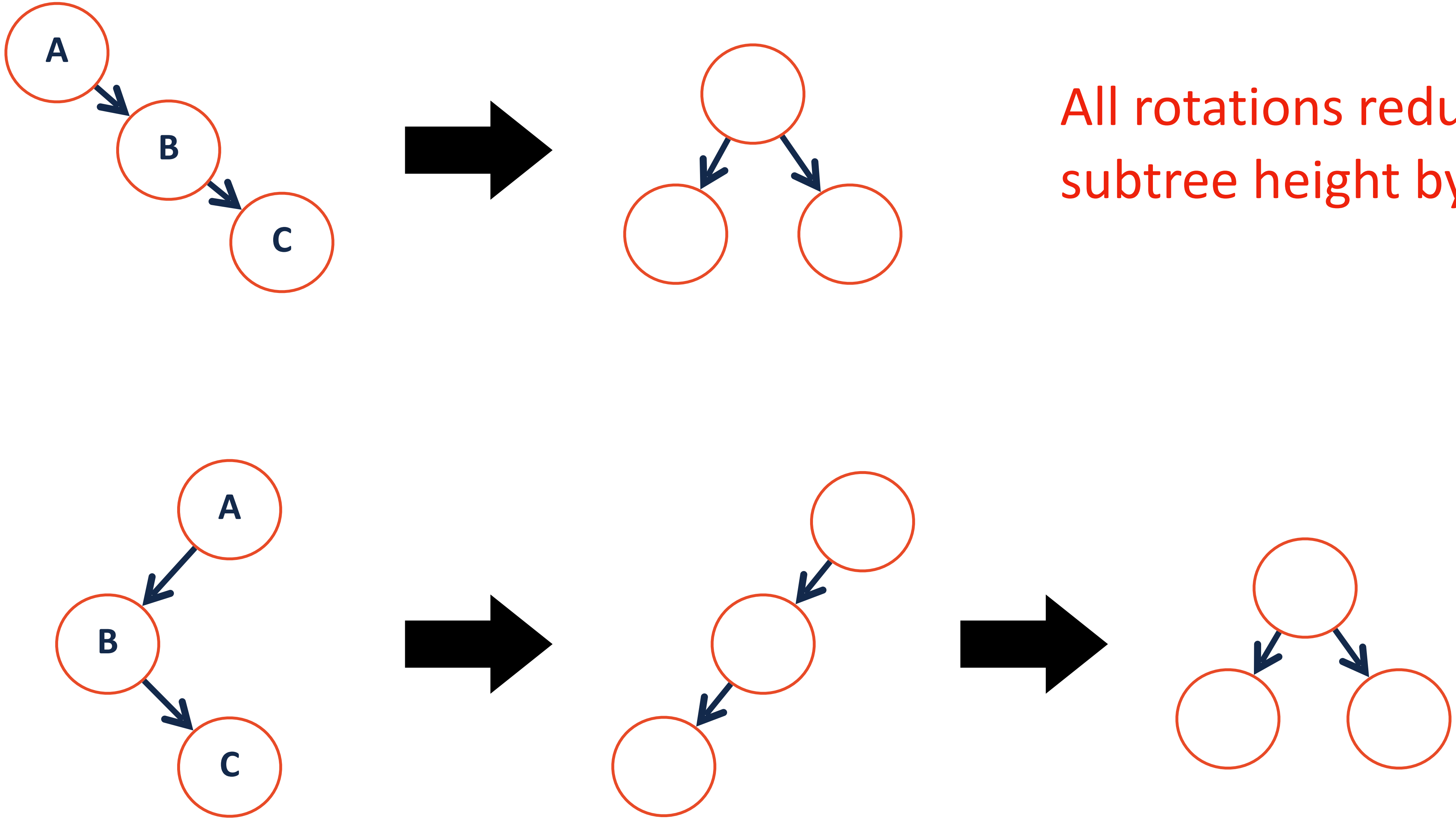
Department of Computer Science

Learning Objectives

Review AVL trees

Prove that the AVL Tree speeds up all operations

AVL Tree Rotations



All rotations are O(1)

All rotations reduce subtree height by one

AVL Tree Analysis

For an AVL tree of height h:

Find runs in: ___________.

Insert runs in: ___________.

Remove runs in: ___________.

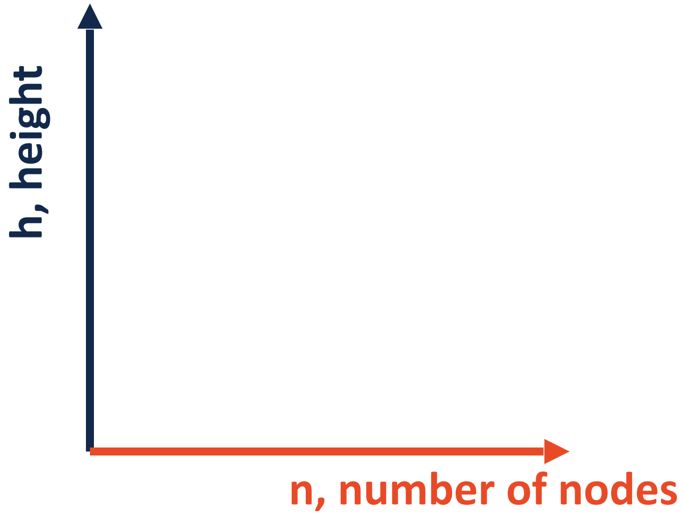
Claim: The height of the AVL tree with n nodes is: **___________.**

AVL Tree Analysis

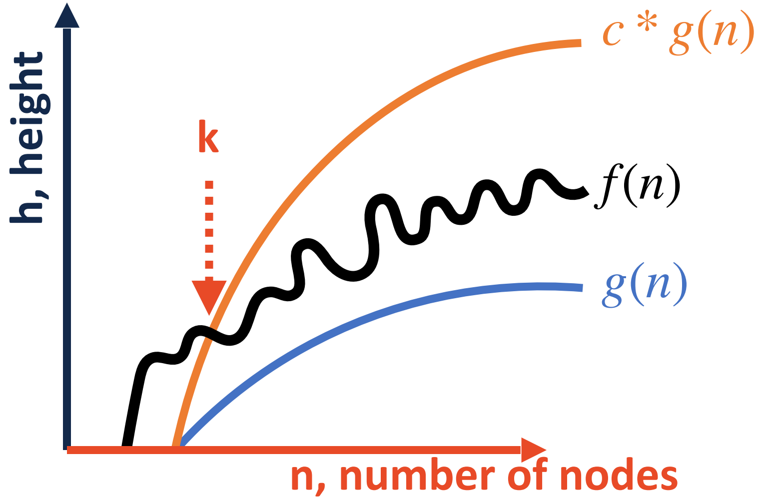
Definition of big-O:

$$f(n) \text{ is } O(g(n)) \text{ iff } \exists c, k \text{ s.t. } f(n) \le cg(n) \ \forall n > k$$

...or, with pictures:

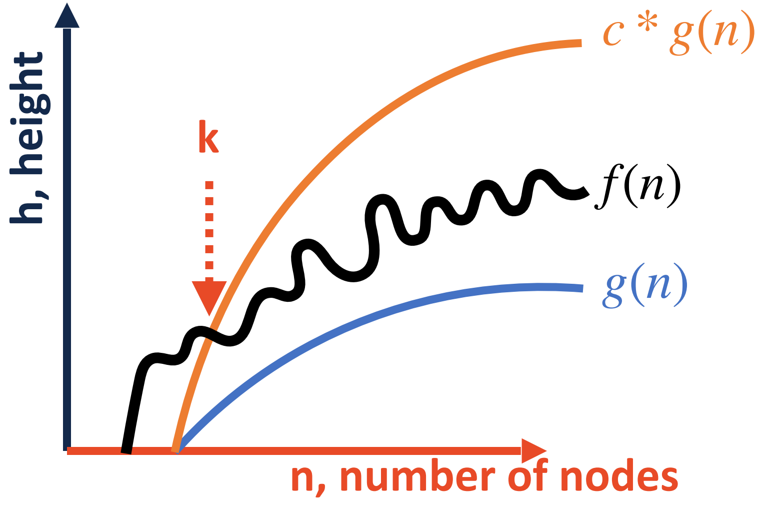


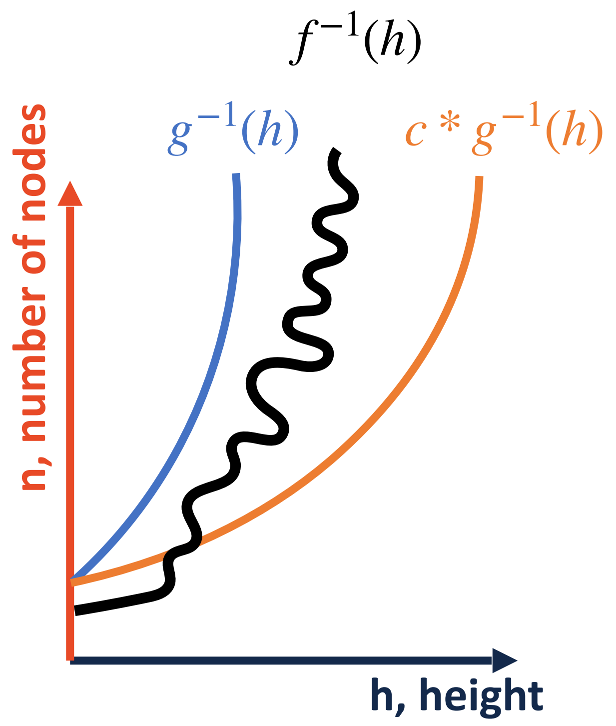
AVL Tree Analysis



The height of the tree, **f(n)**, will always be <u>less than</u> **c × g(n)** for all values where **n > k**.

AVL Tree Analysis





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

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

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

Plan of Action

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

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

$$N(h) \geq N(h) - 1$$

State a Theorem

Theorem: An AVL tree of height h has at least ____________.

Proof by Induction:

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

II. Base Case: _________________

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

Prove a Theorem

III. Base Case: ________________

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

Prove a Theorem

IV. Induction Case: ________________

If for all heights $i < h, N(i) \geq 2^{i/2}$

then we must show for height h that $N(h) \geq 2^{h/2}$

Prove a Theorem

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

…and inverting:

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))**:

# of nodes (n) ≥ N(h) > $2^{h/2}$

$n > 2^{h/2}$

$\lg(n) > h/2$

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

$h < 2 \times \lg(n)$, *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

AVL Trees

- Max height: 1.44 * lg(n)
- Rotations:

Summary of Balanced BST

AVL Trees

- Max height: 1.44 * lg(n)
- Rotations:

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

Red-Black Trees

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

Summary of Balanced BST

Pros:

- Running Time:
  - Improvement Over:
- Great for specific applications:

Summary of Balanced BST

Cons:

- Running Time:

- In-memory Requirement:

Range-based Searches

Q: Consider points in 1D: $\mathbf{p = \{p_1, p_2, \ldots, p_n\}}$.

…what points fall in [11, 42]?

Tree construction:

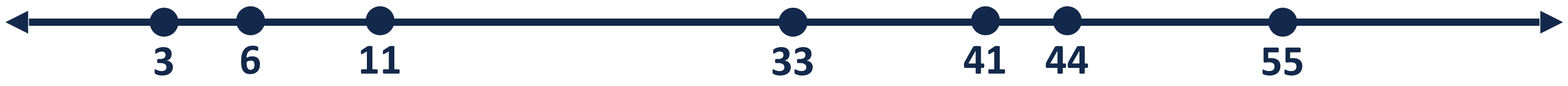
Range-based Searches

Balanced BSTs are useful structures for range-based and nearest-neighbor searches.

Q: Consider points in 1D: $\mathbf{p} = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}\}$.

…what points fall in [11, 42]?

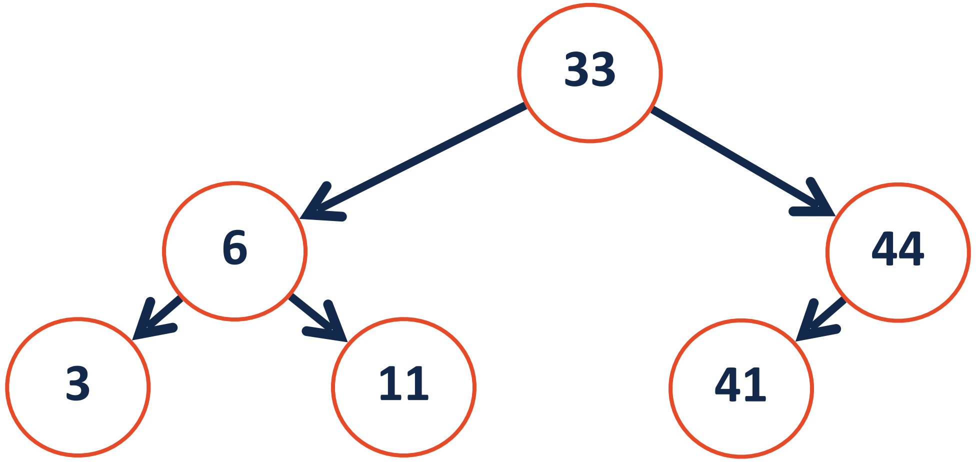
Ex:



Range-based Searches

Q: Consider points in 1D: $\mathbf{p} = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}\}$.

…what points fall in [11, 42]?



Red-Black Trees in C++

C++ provides us a balanced BST as part of the standard library:

```
std::map<K, V> map;

V & std::map<K, V>::operator[]( const K & )



std::map<K, V>::erase( const K & )
```

Red-Black Trees in C++

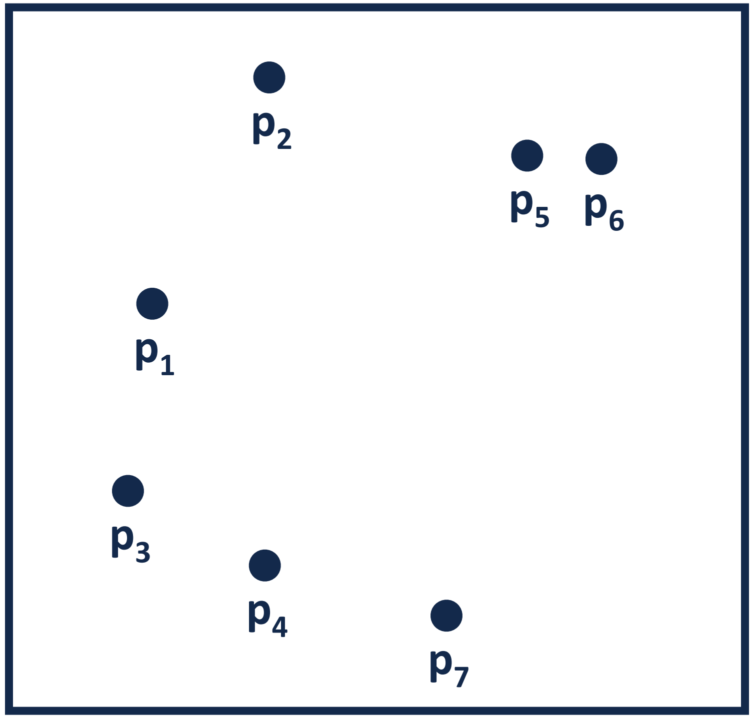
```
iterator std::map<K, V>::lower_bound( const K & );
iterator std::map<K, V>::upper_bound( const K & );
```

Range-based Searches

Consider points in 2D: $\mathbf{p} = \{\mathbf{p_1, p_2, \ldots, p_n}\}$.

Q: What points are in the rectangle:
[$(\mathbf{x_1, y_1}), (\mathbf{x_2, y_2})$]?

Q: What is the nearest point to $(\mathbf{x_1, y_1})$?



Range-based Searches

Consider points in 2D: $\mathbf{p} = \{\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}\}$.

Tree construction: