CS 225
Data Structures

October 11 – AVL Analysis
G Carl Evans
AVL Tree Analysis

We know: insert, remove and find runs in: ____________.

We will argue that: h is ____________.
AVL Tree Analysis

Definition of big-O:

...or, with pictures:

$n$, number of nodes

$h$, height
AVL Tree Analysis

- The height of the tree, $f(n)$, will always be less than $c \times g(n)$ for all values where $n > k$. 
AVL Tree Analysis
• The number of nodes in the tree, $f^{-1}(h)$, will always be greater than $c \times g^{-1}(h)$ for all values where $n > k$. 
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 \( h \):
Simplify the Recurrence

\[ N(h) = 1 + N(h - 1) + N(h - 2) \]
State a Theorem

**Theorem:** An AVL tree of height $h$ has at least ________.

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

II. Case: ______________

An AVL tree of height ____ has at least ____ nodes.
An AVL tree of height _____ has at least _____ nodes.
Prove a Theorem

By an Inductive Hypothesis (IH):

We will show that:

An AVL tree of height _____ has at least _____ nodes.
Prove a Theorem

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

...and inverting:
Summary of Balanced BST

**Red-Black Trees**
- Max height: $2 \times \lg(n)$
- Constant number of rotations on insert, remove, and find

**AVL Trees**
- Max height: $1.44 \times \lg(n)$
- Rotations:
Summary of Balanced BST

Pros:
- Running Time:
  - Improvement Over:
- Great for specific applications:
Summary of Balanced BST

Cons:
- Running Time:

- In-memory Requirement:
Red-Black Trees in C++

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

```cpp
std::map<K, V> map;
```
Red-Black Trees in C++

\[ V & \text{std::map}<K, V>::\text{operator[]} (\text{const } K & ) \]
Red-Black Trees in C++

\[
V & \text{std::map}<K, V>::\text{operator[]}\(\text{const } K \&\) \\
\text{std::map}<K, V>::\text{erase}\(\text{const } K \&\)
\]
Red-Black Trees in C++

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

Why do we care?

```
1 DFS dfs(...);
2 for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
3   std::cout << (*it) << std::endl;
4 }
```
Iterators

Why do we care?

1 DFS dfs(...);
2 for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
3   std::cout << (*it) << std::endl;
4 }

1 DFS dfs(...);
2 for ( const Point & p : dfs ) {
3   std::cout << p << std::endl;
4 }
Iterators

Why do we care?

```
1  DFS dfs(...);
2  for ( ImageTraversal::Iterator it = dfs.begin(); it != dfs.end(); ++it ) {
3      std::cout << (*it) << std::endl;
4  }
```

```
1  DFS dfs(...);
2  for ( const Point & p : dfs ) {
3      std::cout << p << std::endl;
4  }
```

```
1  ImageTraversal & traversal = /* ... */;
2  for ( const Point & p : traversal ) {
3      std::cout << p << std::endl;
4  }
```
## Every Data Structure So Far

<table>
<thead>
<tr>
<th></th>
<th>Unsorted Array</th>
<th>Sorted Array</th>
<th>Unsorted List</th>
<th>Sorted List</th>
<th>Binary Tree</th>
<th>BST</th>
<th>AVL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traverse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>