AVL Running Times

<table>
<thead>
<tr>
<th>AVL Tree</th>
<th>find</th>
<th>insert</th>
<th>remove</th>
</tr>
</thead>
</table>

Motivation:
Big-O is defined as:

Let $f(n)$ describe the height of an AVL tree in terms of the number of nodes in the tree ($n$). Visually, we can represent the big-O relation:

$f(n) \leq c \times g(n)$: Provides an upper bound:

The height of the tree, $f(n)$, will always be less than $c \times g(n)$ for all values where $n > k$.

$f'(h) \geq c \times g'(h)$: Provides a lower bound:

The number of nodes in the tree, $f'(h)$, will always be greater than $c \times g'(h)$ for all values where $n > k$.

Plan of Action:
Goal: Find a function that defines the lower bound on $n$ given $h$.

Given the goal, we begin by defining a function that describes the smallest number of nodes in an AVL of height $h$:

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

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

II. Case: __________________

III. Case: __________________

Inductive hypothesis (IH):

Proving our IH:

V. Using a proof by induction, we have shown that:
...and by inverting our finding:

Summary of Balanced BSTs:

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Using a Red-Black Tree in C++

C++ provides us a balanced BST as part of the standard library:
```cpp
std::map<K, V> map;
```

The map implements a dictionary ADT. Primary means of access is through the overloaded `operator[]`:
```cpp
V & std::map<K, V>::operator[]( const K & )
```

Removing an element:
```cpp
void std::map<K, V>::erase( const K & );
```

Range-based searching:
```cpp
iterator std::map<K, V>::lower_bound( const K & );
iterator std::map<K, V>::upper_bound( const K & );
```

Iterators and MP Traversal

With a traversal you can use the for-each syntax
```
1
2 DFS dfs(...);
3 for ( const Point & p : dfs ) {
4    std::cout << p << std::endl;
5 }
```

The exact code you might use will have a generic `ImageTraversal`:
```
1
2 ImageTraversal & traversal = /* ... */;
3 for ( const Point & p : traversal ) {
4    std::cout << p << std::endl;
5 }
```

Running Time of 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>
</tr>
</thead>
<tbody>
<tr>
<td>Find</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traverse</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

CS 225 – Things To Be Doing:

1. mp_traversals extra credit submission ongoing – due today!
2. Daily POTDs are ongoing!