CS 225

#20: AVL Analysis

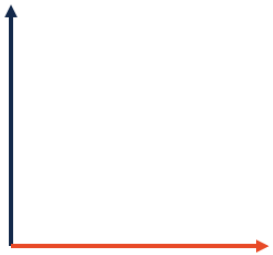
October 11, 2021 · *G Carl Evans*

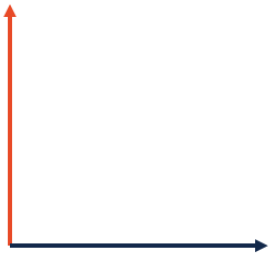
AVL Running Times

	AVL Tree
find	
insert	
remove	

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 **$n > k$**.

$f^{-1}(h) \geq c \times g^{-1}(h)$: Provides a lower bound:

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:
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:

Advantages	Disadvantages

Using a Red-Black Tree in C++

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

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

The map implements a dictionary ADT. Primary means of access is through the overloaded `operator[]`:

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

This function can be used for both insert and find!

Removing an element:

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

Range-based searching:

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

The exact code you might use will have a generic `ImageTraversal`:

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

Running Time of Every Data Structure So Far:

	Unsorted Array	Sorted Array	Unsorted List	Sorted List
Find				
Insert				
Remove				
Traverse				

	Binary Tree	BST	AVL
Find			
Insert			
Remove			
Traverse			

CS 225 – Things To Be Doing:

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