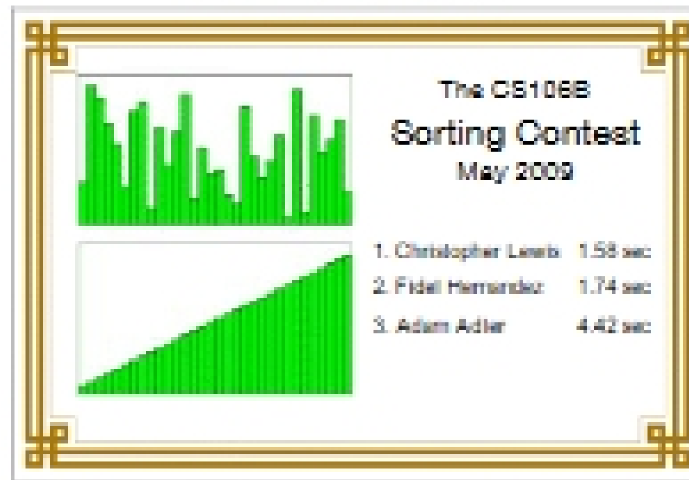
Eric Roberts
CS 106B

Handout #45
May 15, 2009

Binary Search Trees

Contest Results



Map Lookup Revisited

- The hash table strategy we introduced a week ago is by far the most common implementation of maps in practice.
- Despite its many advantages—most notably its extraordinary efficiency—the hash table implementation is not necessarily the best strategy for all applications.
- The hash table strategy has the following limitations:
  - Hash tables depend on being able to compute a hash function on some key. Expanding the hash-function idea so that it applies to types other than strings can be subtle.
  - The iterator for a map based on hash tables does not generate its values in any kind of order. If the key type has a natural order (which is called ***lexicographic order*** in the case of strings), the iterator cannot take advantage of that fact.

Looking Up Keys in an Array

- Instead of using a map, suppose that you entered all the state abbreviations (along with their names) in an array:

- How many operations (as a function of the size *N*) are needed to find a state abbreviation, such as "GA"?
- The lookup operation in a sorted array runs in *O*(log *N*) time.

Adding a Key to an Array

- But what about adding a new key? Suppose, for example, that Puerto Rico (PR) becomes a state:

- All the cells at the end of the array need to move to make room.
- The add operation in a sorted array runs in *O*(*N*) time.

Fixing the Problem

- When we worked with the editor buffer two weeks ago, we solved the insertion problem by using a linked list instead of an array.
- Unfortunately, turning a sorted array into a linked list makes it impossible to apply binary search because there is no way to find the middle element.
- But what if you could point to the middle element in a linked list? That idea seems unlikely, but it is the key to finding a data structure that offers *O*(log *N*) performance for both the lookup and add operations.

Binary Search Trees

- The structure that ends up solving this problem is called a ***binary search tree***. Each node in such a tree has exactly two subtrees: a left subtree that contains all the nodes that come before this one and a right subtree that contains all the nodes that come after it.
- The classical example is the following binary search tree of the seven dwarves:

The bst.h Interface

The bst.h Interface

The bst.h Interface

The bst.h Interface

The bst.h Interface

The bstpriv.h Data Structure

The cmpfn.h Interface

The bstimpl.cpp Implementation

The bstimpl.cpp Implementation

The bstimpl.cpp Implementation

The bstimpl.cpp Implementation

The bstimpl.cpp Implementation