CSCI 2270
Data Structures and Algorithms
Review

Elizabeth White
elizabeth.white@colorado.edu
Office hours: ECCS 112/128
Wed 9:30am-11:30pm
Thurs 10am-11am

Administrivia

HW5 is due tonight.

Colloquium writeup is due to me by email tonight.

Final exam is MONDAY, 1:30 to 4:00, in here.

Open book, open note, no calculators, no computers

15 questions, short answer, code

Administrivia

Review sheet's been posted.

Michael Aaron's running a review session after class and another one tomorrow.

Interview grading survey is up.

Check your grades on Moodle; we're trying to make sure all of them are there

If you see a problem as of today, email me and cc your TA.

Arrays

Most ubiquitous data structure you'll ever use

We started with fixed-size arrays that could fill up

Unsorted: O(1) to add, O(n) to find, O(n) to remove

Then we learned to write resize (and copy) to make room

And we learned that we can search a sorted array way faster…

Binary search is O(log n); removal is O(n), addition is O(n)

Pointers

Pointers store addresses of variables in memory

Any pointer variable in a class (like ArrayBag's items or BigNum's digits) raises the risk of a shallow copy.

Wow, I have the exact same dog!

Linked Lists

Still a linear data structure

Held together entirely by pointers

Compared to arrays, it's easier to add/remove nodes and avoid shifting in a list

But it's slower to find element n in the list, and the pointer dereferences slow it down a little.

Most critical data structure to understand for CSCI 2400

Trees

Act like linked list with branches outward

Held together entirely by pointers

We prefer trees that are balanced (not list-like)

Balanced binary search tree of n items is searchable in O(log n) time

Most critical data structure to understand for Programming Languages (and natural language processing).

Sorting

Quadratic: bubble sort, selection sort, insertion sort, O(n^2)

Recursive sorts are preferable, O(n log n)

Mergesort:

- Split array into tiny sorted arrays (easy)
- Merge sorted arrays back together (time consuming)
- log n splits, each merging all n items: O(n log n)
- Needs an extra array's worth of space

Sorting

Heapsort:

Add each element to a heap (in array form, not tree form) and swap children with parents to make proper heap

When done, largest element is at the root

Move element to back of array and re-heapify

Timing: O(n log n)

Add n elements to heap, repairing with log(n) swaps per element

Remove n elements from heap, repairing with log(n) swaps per element

Sorting

Quicksort: based on partition algorithm

Choose a pivot

run a counter from the back of the array backwards until it finds an element < pivot

Run a counter from the front of the array forwards until it finds an element > pivot

Swap elements

Resume counters

When counters pass each other, swap the last element > pivot with the pivot

Timing: O(n log n) in best case, O(n^2) if bad pivots chosen

Stacks and queues

Special array/list structures

Stack: last in, first out

Queue: first in, first out

Deque: add and remove from either end

Graphs

Like trees with no branching rules

Used to keep track of relationships between entities

We studied how to look through all the elements in a graph by depth first or breadth first search

Search depends on using a stack (DFS) or a queue (BFS)

Be able to traverse a small graph by either rule

Hash tables

Get close to linear time in search by using hash function

Collisions are inevitable, so it's not perfectly linear

Depends on α, how full the table is

Open addressing: inspect $\frac{1}{2}\left(1+\frac{1}{1-\propto}\right)$ slots if item's there

Double hash: inspect $\frac{-ln(1-\propto)}{\propto}$ slots if item's there

Chained hashing: inspect $1+\frac{\propto}{2}$ slots if item's there