CS1003/1004: Intro to CS, Spring 2004

Lecture #7: Algorithms III

Janak J Parekh
janak@cs.columbia.edu

Administrivia

- HW#2 due this week
  - I'll cover running times today
- HW#1 being returned between last week and this week
  - We'll coordinate returns better in the future
- Midterm in two weeks
  - Format of the midterm
  - I'll post a list of topics next week
  - Extra review session?

Agenda

- Finish algorithms discussion (for now)

Here's another way to look at repetition

- fib(n) = fib(n-1) + fib(n-2), right?
- We can actually encode that in a computer
  - *Recursion:* Define a solution in terms of a smaller version of itself
  - Must have *stopping* (base) case(s)
  - What's the base case for the above recursion?
- How about doing *x^y* using recursion?

Other recursive examples

- Power (x^y)
- Binary search
- Palindrome checking
- Most iterative structures can be done recursively, and vice-versa

Algorithm efficiency

- Often, there's multiple ways to implement an algorithm
- How to characterize if one's better or not?
- Two primary considerations:
  - How *fast* does an algorithm run?
  - How much *memory* does an algorithm take?
- Let's focus on the first one for now

Our multiple Fibonacci algorithms

- Do they run at the same speed?
- Let's try fib(10)… then 20… then 40
- Hmm, why do they differ?
- And can we classify this difference

How fast does an algorithm run?

- Let's first think of it in the context of *steps*
- How long might a linear search take through a list of N elements?
- Canonical way to characterize this is to use "big-Oh" notation
  - Key insight: we're interested in orders of magnitude, not constants
  - Strangely, book uses big-Theta notation, which is less used except when doing more formalized analysis

Big-Oh notation

- Basic intuition:
  - Find the number of steps in terms of *n* or other variables
  - Drop any constants or additive lower-order terms
  - Put a O( ) around the result
- Let's look at the previous algorithms we discussed today and see what their big-Oh complexity is…

Other algorithms?

1. An algorithm to compute n! – recursively
2. Sort the contents of an array
   - I don't like insertion sort – let's do bubble sort
- We'll continue to do more "interesting" algorithms as the semester proceeds

Next time

- Continue algorithms