Administrivia

- Reminder: Homework 6 due today. When you turn in homework, please try to remember to put in the subject line which course it's for. Sometimes it doesn't matter much, but for two courses I teach there's a Homework 6 due today!
- Next homework to be assigned Friday, due a week later.

Slide 1

Minute Essay From Last Lecture

- One person asked about how/when I use arrays in my programs. My first thought was "all the time!" but . . .
- Arrays are a simple kind of "collection" but there are others, and the others are apt to be better if you don't need to work with large numbers of elements but also don't know ahead of time how many. If you do know, and you need efficiency, arrays work well.
- Matrices are an obvious use, but if they're "sparse" (lots of entries zero) it may be more space-efficient to represent them another way.
- Another application area I know a little about indirectly is "molecular dynamics", which involves simulating large numbers of atoms, storing for each atom mass, position, velocity, etc.

Slide 2

Slide 3

Sorting — Recap/Review

- Problem: Given an array (or list) of elements for which there is a sensible “less than” operator, put them in order.
- Simple solutions include bubble sort, selection sort, insertion sort. Easy to program but not “fast” (more shortly). Textbook discusses these pretty well.
- (Examples at board of bubble sort and selection sort.)
- More-complex but “faster” algorithms exist and (at least conceptually) use recursion (!).

Slide 4

Searching — The Problem and Some Solutions

- Problem: Given an array (or list) and an element, search the array for the element.
- Simplest solution is sequential search. Easy to program and works for any array but not “fast”.
- Slightly more-complex solution is binary search. “Faster” but requires array to be in order.
- Textbook has good discussions. (Also example(s) at board?)

Slide 5

Order of Magnitude of Algorithms

- Conventional wisdom (among computer scientists) is to write programs in a way that humans can understand, and let the compiler turn them into something that will run fast.
- One exception is "order of magnitude" of algorithm, however.
- Key idea is to think about how execution time (or some other measure, such as memory requirements) scales with "problem size".
- Roughly analogous to order of magnitude of numbers — provide a way of grouping into classes in which all members of one class are sort of "the same" but members of different classes are not.
- Typically written using "big-O" notation (e.g., $O(N)$, $O(N^2)$, etc.). Formal definition possible, but informally, $O(f(N))$ means that execution time (or whatever) for problem size N scales as $f(N)$.

Slide 6

Order of Magnitude of Algorithms, Continued

- A key idea — for large enough problem sizes, algorithms with smaller orders of magnitude are faster, though this may not be true for small problem sizes.
- Another key idea — some orders of magnitude (e.g., $O(2^N)$) are sufficiently "big" that solving problems of any non-trivial size is simply not feasible, so "wait until computers get faster" is probably not a good strategy. "Hm!"?
- Can help rule out algorithms that would not be practical/feasible for large problems.

  A famous(?) example — "traveling salesperson problem", for which all known algorithms require considering, for N cities, all possible permutations, making them $O(N!)$. Not reasonable! (Worth noting that there apparently *are* practical approximations. Still!)

Slide 7

Sidebar: `gnuplot`

- A tool I like for both quick interactive plots and nice-looking ones to use in papers is `gnuplot`. Available on most UNIX-like systems and (I think!) optionally for other operating systems. Home page at `gnuplot.sourceforce.net`. Can do 2D and 3D plots, the former with Cartesian or polar coordinates.
- To start it, `gnuplot`. Brings up a command-line interface. Online help available with `help`. More next time, but for now we could plot some simple polynomial functions and observe how much faster ones with an x^2 grow than linear functions, and how ones with an x^3 grow even faster, and so forth.

Slide 8

Order of Magnitude of Algorithms, Continued

- As an example, look at bubble sort and selection sort.
- For both, "problem size" is the number of elemnents to sort, and a rough measure of how execution time scales with problem size is based on how many comparisons are needed, in the worst case.
- Again for both, total number of comparisons is $N(N-1)/2$, making them "$O(N^2)$".
- As another example, look at sequential search and binary search. The first is $O(N)$, but the second is ... What? ($O(\log N)$)

Slide 9

Minute Essay

- None — quiz.