Class 47: Introduction to Sorting

Held: Wednesday, April 28, 2010

Summary: We explore the problem of *sorting*. When you sort a list, vector, or other collection, you put the elements in order. The order of the elements usually corresponds to the type of the elements. We might sort strings alphabetically, grades numerically, colors by brightness, and so on and so forth.

Related Pages:

- EBoard.

Notes:

- Reading for Friday: Sorting.
- I will not be in class on Friday; Prof. Weinman and a guest will be.
- EC for Osgood convocation tomorrow.
- EC for Friday's CS Extra Table (Science 3821). I need a headcount!

Overview:

- The problem of sorting.
- Writing sorting algorithms.
- Examples: Insertion, selection, etc.
- Formalizing the problem.

The Problem of Sorting

- As we saw recently, one problem that seems to crop up a lot in programmming (and elsewhere) is that of *sorting*.
- The problem: Given a list, array, vector, sequence, or file of comparable elements, put the elements in order.
  - *In order* typically means that each element is no bigger than the next element. (You can also sort in decreasing order, in which case each element is no smaller than the next element.)
- We'll look at techniques for sorting vectors and lists.

Designing Sorting Algorithms

- I suggest that you think about the development of sorting algorithms in Scheme similarly to the way you think about writing many algorithms.
- Start by thinking about the way you might do it by hand.
  - We may find a few different ways to sort by hand.
  - We'll probably leave the Scheme-ification to the end.

- Generalize what you're doing.
  - What is the "philosophy" of your techinque?
  - What are the key steps.
- Come up with some initial test cases.
- Consider whether there are any deficiences to your technique.
- Decide on your parameters (e.g., are you sorting a list or a vector).
- Translate your algorithm into Scheme.
- Test test test.

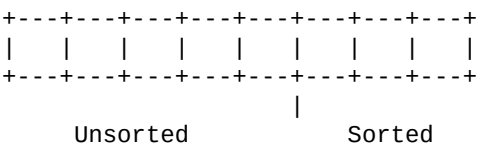
Sample Sorting Algorithms

Insertion Sort

- One simple sorting technique is *insertion sort*.
- Insertion sort operates by segmenting the list into unsorted and sorted portions, and repeatedly removing the first element from the unsorted portion and inserting it into the correct place in the sorted portion.
- This may be likened to the way typical card players sort their hands.
- How might we code this recursively?
- Does our code differ for lists and arrays?

Selection Sort

- *Selection sort* is among the simpler and more natural methods for sorting vectors.
- In this sorting algorithm, you segment the vector into two subparts, a sorted part and an unsorted part. You repeatedly find the largest of the unsorted elements, and swap it into the beginning of the sorted part. This swapping continues until there are no unsorted elements.
- Here's a potentially-helpful picture:



- Note that we can also write selection sort iteratively.

A More Formal Description

- Before moving on to algorithms for solving the sorting problem, let's take a look at the way we might document one (or all) of the procedures
  - Purpose?
  - Parameters?
  - Produces?
  - Preconditions?

    - Postconditions?
- Here are some postconditions I typically think about:
    - You also need to ensure that all elements in the original list are in the sorted list.
    - You also need to ensure that no other elements are in the list.
