

CSC151.02 2014S, Class 47: Introduction to Sorting

Overview

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

Preliminaries

Admin

- A draft version of exam 3 is ready.
- Yay to work/life balance!

Upcoming Work

- Reading for Friday: Sorting.
- Homework for Thursday: Projects.
- No lab writeup.

Extra Credit

- Any pride week activity.
- Town hall Wednesday 7:30 p.m. (work/life balance)
- Math Extra, Thursday. Sphere packing.
- Iowater project April 26 - Tag drains. Mail [iowater] for details.
- Dance shows, 7:30 p.m. Thursday, Friday, Saturday, 2:30 pm Sunday
- Jazz combo Bucksbaum 152 Thursday 7pm.
- 24 hour improv Saturday (e.c. for 1 hour)

Questions

The problem of sorting

- Given an arbitrary list/vector of values, put them in order.
- Lets us do binary search.
- Sometimes gives us orders for doing things.

Writing sorting algorithms

- A few issues:
 - Write instructions for sorting
 - Verify correctness
 - Explore efficiency
- Sorting is something each of us does all the time; it's just a matter of formalizing what we do.
- Challenge: In a few minutes, I'll bring ten students to the front of the room and try to sort them using your instructions.
- In groups of about three-four, write instructions for putting those students in order. Assume that we can compare in various ways.

Strategy one:[Quicksort]

- Divide into small and large
 - Recursively do that to each half

Strategy two: [Selection sort]

- Repeatedly
 - Find the smallest remaining number and move it to the beginning

Strategy three: [Bubble sort]

- Repeatedly
 - Swap neighboring elements

Strategy four: [Insertion sort]

- Repeatedly
 - Put an unsorted value into the correct place in a list of sorted values

All are correct.

Quicksort is much faster than the other three. Quicksort is also really easy to get wrong.

Examples: Insertion, selection, etc

Formalizing the problem

Suppose we had to write the documentation for a sorting procedure for lists

;;; Procedure: ;;; sorter ;;; Parameters: ;;; ah, a potentially empty list ;;; get-key, a unary procedure which extracts the key used for sorting ;;; may-precede?, a predicate which determines whether one key ;;; should come before another ;;; Purpose: ;;; Rearrange the elements of the list in a particular order ;;; Produces: ;;; sortah, a list ;;; Preconditions: ;;; may-precede? must be applicable to any two keys ;;; may-precede? is a complete order - transitive ;;; all the elements of ah should be the same type ;;; get-key must be applicable to all elements of ah ;;; Postconditions: ;;; sortah is ordered by get-key/may-precede ;;; For all reasonable i, ;;; (may-precede? (get-key (list-ref sortah i)) (get-key (list-ref sortah (+ i 1)))) ;;; sortah is a permutation of ah

;;; Procedure: ;;; vector-sorter ;;; Parameters: ;;; um, a vector ;;; Purpose: ;;; Produces: ;;; Preconditions: ;;; Postconditions:

Samuel A. Rebelsky, rebelsky@grinnell.edu

Copyright (c) 2007-2014 Janet Davis, Samuel A. Rebelsky, and Jerod Weinman. (Selected materials are copyright by John David Stone or Henry Walker and are used with permission.)



This work is licensed under a Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.