CS151.02 2010S *Functional Problem Solving*

Class 49: Merge Sort

Held: Monday, May 3, 2010

Summary: We continue our exploration of sorting by considering the applicability of divide-and-conquer to the problem of sorting. We look at one particular divide-and-conquer algorithm, *merge sort*. We explore how the running time for that algorithm varies based on the number of values we are sorting.

Related Pages:

* EBoard.
* Lab: Merge Sort.
* Reading: Merge Sort.

Notes:

* For Tuesday and Wednesday, please review your classmates' work (URL distributed electronically).
* I'll reserve time at the start of class for questions on the examination.
* EC for Thursday's Convocation.
* EC for Sunday's Belly Dance performance (1:30 in Flanagan).

Overview:

* More efficient sorting techniques.
* Divide and conquer, revisited.
* Merge sort.
* Analyzing merge sort.

Key Ideas of Merge Sort

* Divide and conquer is often a useful design strategy.
* For sorting, natural division is "first half" / "second half".
* What do we do after sorting the two halves? Merge 'em.

An Alternate Implementation

* We can do something very much like merge sort while avoiding the splitting step.
* We start with a list of sorted lists, and repeatedly merge neighboring pairs.
* This technique is slightly easier to implement.
* This technique is much easier to analyze.

The Costs of Merge Sort

- What's the running time? We do approximately $N*\log_2 N$ comparisons.
- The analysis:
  - *N* steps to merge 2 sorted lists of length *N/2*
  - *N* steps to merge 4 sorted lists of length *N/4* into those two sorted lists.
  - *N* steps to merge 8 sorted lists of length *N/8* into those four sorted lists.
  - And so on and so forth.
- Can we do better? Not if our sorting is based on comparing values to each other.

Lab

- Do the lab.
- Pause to reflect.