Institut for Matematik og Datalogi
Syddansk Universitet

September 4, 2003
JFB

DM19 – Algorithms and Complexity – E03 – Lecture 2

Announcement

More copies of the notes for DM19 have been ordered for the bookstore. We will be using some of the notes before they arrive, so I have copies of the first parts of the first set of notes. You may get a copy from me, in my office.

Lecture, September 1

We began with an introduction to the course. Randomized Quicksort, from sections 7.3 and 7.4 in the textbook, was presented. We also covered probability from Appendix C, sections C.2, C.3, and C.4. The slides I used for this are included in those which are now available on the DM19's Web page. We also covered radix sort from section 8.3.

Lecture, September 8

We will start with counting sort from section 8.2. Then lower bounds from section 2.4 of the first set of notes will be discussed. (Part of this is also in section 8.1 of the textbook.) We will continue with adversary arguments from section 3.1 of the same set of notes.

Lecture, September 15

Paul Medvedev will lecture on string matching from chapter 32.

Problems to be discussed in week 38

- Do the following problems from the textbook: 8.1-1, 8.1-3 (just do the first part), 8.2-2. 8.2-4.

- Do problems 3.2 (use Stirling's approximation - formula 3.17 from the textbook, and compare your result to the upper bound in section 7.1 of the textbook, rather than to the lower bound mentioned) and 3.10 from page 140 and 141 of the notes.

- From the following pages of that book by Baase:

  > Consider the problem of determining if a bit string of length n contains two consecutive zeros. The basic operation is to examine a position in the string to see if it is a 0 or a 1. For each $n = 2, 3, 4, 5$ either give an adversary strategy to force any algorithm to examine every bit, or give and algorithm that solves the problem by examining fewer than n bits.

  and

  > a. You are given n keys and an integer k such that $1 \leq k \leq n$. Give an efficient algorithm to find *any one* of the k smallest keys. (For example, if $k = 3$, the algorithm may provide the first-, second- or third-smallest key. It need not know the exact rank of the key it outputs.) How many key comparisons does your algorithm do? (Hint: Don't look for something complicated. One insight gives a short, simple algorithm.)
  > b. Give a lower bound, as a function of n and k, on the number of comparisons needed to solve this problems.