Institut for Matematik og Datalogi
Syddansk Universitet

January 31, 2008
JFB

DM508 – Algorithms and Complexity – F08 Lecture 2

Lecture, January 28

We began with an introduction to the course. Randomized Quicksort, from sections 7.3 to 7.4 in the textbook, were presented. Randomized Quicksort is also presented in the course notes, those by Motwani and Raghavan. Counting sort from section 8.2 was also be presented. We began on information theoretic lower bounds, defining decision trees, and observing that showing that a problem has at least k outputs implies that an algorithm doing t-way tests must do at least $\lceil \log_t(k) \rceil$ tests in the worst case. This is in section 8.1 of the textbook and section 2.4 of the first part of the course notes.

Lecture, January 30

Lower bounds from section 2.4 of the first part of the notes will be discussed (part of this is also in section 8.1 of the textbook). We will also begin on sections 3.1, 3.2 and 3.3 of those notes.

Lecture, February 4

We will cover section 3.5 of the DM508 notes, plus median finding from chapter 9 (sections 9.2 and 9.3) in the textbook. We may also begin on NP-completeness, from chapter 34 in the textbook and the section by Papadimitriou and Steiglitz from the course notes.

Problems to be discussed on February 8

Do problems:

1. Do problems 3.2 (use Stirling's approximation - formula 3.17 from the textbook, and compare your result to the upper bound for merging, rather than to the lower bound mentioned) and 3.10 from pages 140 and 141 of the notes.
2. 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.

3. From Baase's textbook: Suppose $L1$ and $L2$ are arrays, each with n keys sorted in ascending order.

   a. Devise an $O((\lg n)^2)$ algorithm (or better) to find the nth smallest of the $2n$ keys. (For simplicity, you may assume the keys are distinct.)

   b. Give a lower bound for this problem.

4. Design an algorithm for finding the second largest item in an array, which is similar to the standard algorithm for finding the largest. Keep track of the largest and second largest at each step. How many comparisons does your algorithm do in the worst case?

5. Design an efficient algorithm to find the third largest item in an array.

6. Consider the problem of Sorting by Reversals. You are given a permutation of the numbers from 1 to n in an array, A. The operation you have is to choose two indices, i and j, and to reverse the elements in the subarray from $A[i]$ to $A[j]$, inclusive. The objective is to end with a sorted array. For example, given $A = [8, 6, 4, 2, 7, 5, 3, 1]$, the first operation could be $(3, 6)$, resulting in $A = [8, 6, 5, 7, 2, 4, 3, 1]$. Then doing the operations $(2, 4), (3, 4), (5, 7), (5, 6), (1, 8)$ would finish sorting the array.

   a. Give an algorithm which sorts the array in $O(n)$ operations.

   b. Prove that any algorithm needs at least $\Omega(n)$ operations in the worst case.

Assignment due Friday, February 15, 8:15

Note that this is part of your exam project, so it must be approved in order for you to take the exam in March, and you may not work with or get help from others not in your group. You may work in groups of two or three. You may write your solutions in English or Danish, but write very neatly if you do it by hand.

Consider a company with a customer service department consisting of an automatic system and two representatives. The company has customers in three cities, Detroit, Chicago, and San Francisco. A request for service requires one of the service representatives to be in that city to service it. Each city has an apartment for the service representative, who will stay there until called to another city. Assume the flight between Chicago and Detroit costs only a fraction $1/d$ of what the flight between Chicago and San Francisco costs, and that it is impossible to fly from Detroit to San Francisco without changing planes in Chicago and paying the sum of the ticket costs from Detroit to Chicago and from Chicago to San Francisco. (Assume that all costs are symmetric, so that it costs exactly as much fly the opposite direction between two cities.) The company wishes to minimize the amount it spends on plane tickets. Assume in what follows that there is initially one service representative in Detroit and one in San Francisco. In the following, assume that if the next request for service is in a city where there is already a service representative, then no service representative flies anywhere, and this costs nothing.

1. Show that in the worst case, any algorithm has cost at least $n \cdot f$ on a sequence of requests of length n, where f is the cost of the cheaper flights.

2. Consider the Greedy algorithm which always requires the closer representative (note that the closer pair of cities has the lower cost airfare) to travel (when travel is necessary). Also consider the algorithm Dummy which, on requests in Chicago always sends the representative which is in San Francisco (when travel is necessary), but otherwise sends the closer representative. Show that for any constant c, there exists a sequences where Dummy pays a factor more than c times what Greedy pays and another sequence where Greedy pays a factor more than c what Dummy pays.

3. Design an algorithm Random, where its expected cost on any sequence is never more than twice what Greedy's is (on that same sequence). (Hint: Let the closest service representative fly unless the request is in Chicago. If it is in Chicago, choose randomly between the two with some fixed probabilities depending on d.)

4. Let S be a sequence requests, containing at most one request in San Francisco, which is at the end if it exists. An example of such a sequence might be [Chicago, Detroit, Detroit, Chicago, Chicago, Chicago, Detroit, Chicago, San Francisco]. Suppose the sequence requires Greedy to pay for n flights (in this example $n = 5$). What is Random's expected cost as a function of n, f, and d?

5. Show that Random's expected cost on any sequence is never more than twice what Greedy's is. Hint: For any request sequence, consider subsequences ending with requests in San Francisco, so that you can assume there are no requests in San Francisco in the middle of the request sequence. Analyze these subsequences separately. Divide the analysis into two cases, one where Greedy has at most $2d$ flights and one where it has more. For Greedy, give a lower bound on the flight costs for the service representatives. For Random, give an upper bound on the expected flight costs for

each of the two service representatives. You may use that $\sum_{i=0}^{\infty} i \cdot r^i = \frac{r}{(r-1)^2}$ for $|r| < 1$.