CSOR W4231 (sec. 001, 002, H02): Analysis of Algorithms Oct. 2

Homework 3: Due on Oct 17 by 12:01am

Instructors: *Alex Andoni, Cliff Stein*

Instructions. Please follow the homework policy and submission instructions in the Course Information handout. A few highlights to remember:

- follow the collaboration and academic honesty policies;
- write your name, uni, and collaborators on the top page;
- submission is via GradeScope (you are encouraged to use LaTeX, using the provided template);
- if you don't know how to solve a particular part of a problem, just write "*empty*", for which you will get 20% of the points of that part (in contrast, note that non-sensical text may get 0%).

Note that, for each bullet item of a problem, you can use the previous bullets as "black box", even if you didn't manage to solve them. Similarly, you can use anything from the lectures as black-box.

Problem 1 (25pts)

Consider a sorting problem in which you do not know the numbers exactly. Instead, for each number, you know an interval on the real line to which it belongs. That is, you are given n closed intervals of the form $[a_i, b_i]$, where $a_i \leq b_i$. The goal is to *fuzzy-sort* these intervals, i.e., to produce a permutation $\langle i_1, i_2, \ldots, i_n \rangle$ of the intervals such that for $j = 1, 2, \ldots, n$, there exist $c_j \in [a_{i_j}, b_{i_j}]$ satisfying $c_1 \leq c_2 \leq \cdots \leq c_n$.

a) Design a randomized algorithm for fuzzy-sorting n intervals, by modifying the Quick-Sort algorithm. Specifically, your algorithm should follow the general structure of an algorithm that quicksorts the left endpoints (the a_i values), but it should take advantage of overlapping intervals to improve the running time. (As the intervals overlap more and more, the problem of fuzzy-sorting the intervals becomes progressively easier. Your algorithm should take advantage of such overlaps, to the extent that it exists.)

b) Argue that your algorithm runs in expected time $\Theta(n \lg n)$ in general, but runs in expected time $\Theta(n)$ when all of the intervals overlap (i.e., when there exists a value x such that $x \in [a_i, b_i]$ for all i). Your algorithm should not be checking for the latter case explicitly; rather, its performance should naturally improve as the amount of overlap increases (i.e., it should still be $O(n)$ even if all but a few intervals overlap).

Problem 2 (25pts)

Suppose we have n strings of characters (from the 26-letter English alphabet). The lengths of the strings are $\ell_1, \ell_2, \ldots \ell_n \geq 1$. For $m = \ell_1 + \ell_2 + \ldots \ell_n$, show how to sort the strings lexicographically in time $O(m)$. (For reference, here's an example of 6 strings sorted lexicographically: "a" < "aa" < "abracadabra" < "b" < "bcd" < "c".)

Problem 3 (25pts)

Consider the problem of searching an item x in an array $A[1 \ldots n]$ with distict entries. We will prove that this problem requires $\Omega(\log n)$ runtime for comparison-based algorithms.

a) Fix any *comparison-based* algorithm for outputing the position of x in the array A (if it exists, and "no" otherwise). Prove that any such algorithm must make at least $\Omega(\log n)$ comparisons.

b) Now consider a *comparison-based* algorithm that just outputs "yes" if x appears in A and "no" otherwise. Prove the same lower bound of $\Omega(\log n)$ comparisons.

Problem 4 (25pts)

A certain string-processing programming language allows you to break a string into two pieces. Because this operation copies the string, it costs n time units to break a string of n characters into two pieces.

Suppose that you want to break a string into many pieces (at specified positions). The order in which the breaks occur can affect the total amount of time used. For example, suppose that you want to break a 20-character string after characters 2, 8, and 10 (numbering the characters in ascending order from the left-hand end, starting from 1). If you program the breaks to occur in left-to-right order, then the first break costs 20 time units, the second break costs 18 time units (breaking the string from characters 3 to 20 at character 8), and the third break costs 12 time units, totaling 50 time units. If you program the breaks to occur in right-to-left order, however, then the first break costs 20 time units, the second break costs 10 time units, and the third break costs 8 time units, totaling 38 time units. In yet another order, you could break first at 8 (costing 20), then break the left piece at 2 (costing another 8), and finally the right piece at 10 (costing 12), for a total cost of 40.

Design an algorithm that, given the indeces of characters after which to break, determines the least-cost way to sequence those breaks. More formally, given an array $L[1 \ldots m]$ containing the break points for a string of n characters, compute the lowest cost for a sequence of breaks, along with a sequence of breaks that achieves this cost.