Algorithms & Programming: Retake Exam

Pierre Senellart (pierre.senellart@ens.fr)

10 April 2018

The only allowed document is an A4 paper sheet, recto-verso (2 pages), with the content of your choice. This exam lasts three hours, contains four exercises, and is marked out of 20 points. Because it is a retake exam, the grade will be capped at 10.

A. Exercise: Sorting Prefixes (6 points)

Suppose that we are given a sequence $(a_1, \ldots, a_n)$ of n distinct elements that are *k-prefix unsorted*, for some integer $0 \leqslant k \leqslant \frac{n}{2}$. That is, the length-$s$ suffix of A, where $s = n - k$, is already sorted in non-decreasing order. (The elements of the length-k prefix can be arbitrary and in arbitrary order.)

For example, $A = (3, 1, 5, 2, 4, 6, 7)$ is a 3-prefix unsorted sequence of length 7.

1. (2 points) Provide an asymptotic estimate of the *worst-case* bound (in terms of n and k) on the numbers of **comparisons** that Insertion Sort makes on such k-prefix unsorted sequences.

2. (2 points) Give an asymptotic lower bound (in terms of n and k) on the number of **comparisons** needed by *any* comparison-based sorting algorithm to sort any k-prefix unsorted sequence A. Briefly justify your result. *Hint: recall that if $k \leqslant \frac{n}{2}$ then $\binom{n}{k} = \Theta(n^k)$.*

3. (2 points) Describe a comparison-based algorithm for sorting any k-prefix unsorted sequence that is efficient with respect to the number of comparisons it makes. (In other words, we care only about the number of comparisons and not the time efficiency here.) Provide an asymptotic worst-case estimate of the number of **comparisons** made by your algorithm. (You can assume that you know the value of k.) Your estimate should match the lower bound established in the previous question.

B. Paren Puzzle (5 points)

A newspaper publishes puzzles of the following form:

Parenthesize $6 + 0 \times 6$ to maximize the outcome.
Wrong answer: $6 + (0 \times 6) = 6 + 0 = 6$.
Right answer: $(6 + 0) \times 6 = 6 \times 6 = 36$.

Parenthesize $0.1 \times 0.1 + 0.1$ to maximize the outcome.
Wrong answer: $0.1 \times (0.1 + 0.1) = 0.1 \times 0.2 = 0.02$.
Right answer: $(0.1 \times 0.1) + 0.1 = 0.01 + 0.1 = 0.11$.

To save yourself from tedium, but still impress your friends, you decide to implement an algorithm to solve these puzzles. The input to your algorithm is a sequence $x_0, o_0, x_1, o_1, \ldots, x_{n-1}, o_{n-1}, x_n$ of $n+1$ real numbers $x_0, x_1, \ldots, x_n$ and n operators $o_0, o_1, \ldots, o_{n-1}$. Each operator o_i is either addition ($+$) or multiplication ($\times$). Give a polynomial-time dynamic program for finding the optimal (maximum-outcome) parenthesization of the given expression, and analyze the running time.

C. Shortest Paths with Fewest Edges (3 points)

When there is more than one shortest path from one node s to another node t in a graph, it is often convenient to choose a shortest path with the fewest edges; call this the best path from s to t. Suppose we are given a directed graph G with positive edge weights and a source vertex s in G. Describe an algorithm to compute best paths in G from s to every other vertex. Your algorithm should be a simple modification to Dijkstra's algorithm; you just have to briefly describe the modification and explain why this works.

D. Cookie Quality Control (6 points)

To ensure cookie perfection in a cookie factory, a special, highly sensitive testing apparatus has been installed that analyzes cookies by weight two-at-a-time and determines which is heavier. (It is so sensitive that no two cookies weigh exactly the same. We assume here that each use of the apparatus takes constant time.)

Spending more and more time at the bakery, you find inspirations for many advances in algorithm design. For instance, you have developed an $O(n)$ (linear time) algorithm that finds from among a set of n cookies, one that is both heavier than at least a quarter of the cookies and lighter than at least (another) quarter. You proudly name this algorithm the *quartering algorithm*, and you decide to see if you can develop more algorithms based on it. (Note that the quartering algorithm returns only the cookie and not these two quarters.)

1. (3 points) Using the quartering algorithm from above, design an algorithm that identifies the k-th heaviest cookie in a set of n cookies. Give and solve the recurrence that describes the running time of your algorithm.

2. (3 points) If one did not have the quartering algorithm, one could solve the problem of finding the k-th heaviest cookie by using a comparison-based sorting method, e.g., Merge sort, which would run in $\Theta(n \log n)$ time. This seems to be doing too much work, as it solves the problem more thoroughly than needed. (It sorts all the cookies, and at worst we only need the top-k.) Design an alternative approach to finding the k-th heaviest cookie that does not use the quartering algorithm and outperforms the $\Theta(n \log n)$ sorting bound, for sufficiently small k. Your algorithm should run in $O(n + k \log n)$ time.