CS 38 Introduction to Algorithms **Spring 2014**

Problem Set 3

Out: April 22 *Due:* **April 29**

Reminder: you are encouraged to work in groups of two or three; however you must turn in your own write-up and note with whom you worked. You may consult the course notes and the optional text (CLRS). The full honor code guidelines can be found in the course syllabus.

Please attempt all problems. **To facilitate grading, please turn in each problem on a separate sheet of paper and put your name on each sheet. Do not staple the separate sheets.**

1. Recall that Fibonacci heaps support three operations, INSERT, EXTRACT-MIN, and DECREASE-KEY, with amortized costs $O(1)$, $O(\log n)$, and $O(1)$, respectively. Describe a sequence of $n = 2^k$ operations so that a final EXTRACT-MIN call takes at least k operations. Assume that whenever a root is added to the root-list by any operation it is added to the left-most position, and that the consolidation process associated with *extract-min* processes the root list from left to right.

2. In a list of integers, $a_1, a_2, \ldots, a_n$, an *inversion* is a pair (a_i, a_j) with $i < j$ but $a_i > a_j$. Give a divide-and-conquer algorithm for computing the number of inversions in a list of n integers, that runs in time $O(n \log n)$. Hint: your algorithm may resemble Mergesort.

3. Set $N = 2^n$ and consider the $N \times N$ matrix M with rows and columns indexed by $\{0, 1\}^n$, whose (a, b) entry equals
$$(-1)^{\sum_{i=1}^{n} a_i b_i}.$$
Give an algorithm that uses at most $O(N \log N)$ operations for multiplying M with a vector.

4. A Toeplitz matrix is a matrix whose diagonals are constant. An $n \times n$ Toeplitz matrix can thus be described by $2n - 1$ values, specifying the values on each distinct diagonal. For example, here is the 4×4 Toeplitz matrix described by the values $4, 3, 6, 7, 2, 5, 9$:
$$\begin{pmatrix} 7 & 2 & 5 & 9 \\ 6 & 7 & 2 & 5 \\ 3 & 6 & 7 & 2 \\ 4 & 3 & 6 & 7 \end{pmatrix}.$$
Give an algorithm to multiply two $n \times n$ Toeplitz matrices (each specified by a list of $2n - 1$ values) in time $O(n^2 \log n)$. Hint: reduce the problem to computing several products of polynomials.

5. In this problem you will develop the (current) asymptotically fastest deterministic algorithm for factoring integers. If an integer N has a non-trivial factor, then it can be assumed to be a

most $\sqrt{N}$, so there is a brute force algorithm that tries $\sqrt{N}$ possible factors. Your algorithm will require only $N^{1/4}$ operations.

In lecture we saw that degree-n polynomial multiplication and division with remainder could be performed in $O(n \log n)$ operations when the polynomials had coefficients in the complex numbers. In fact those algorithms work without modification over any ring, and in this problem you will be working over the ring of integers modulo N, where N is the number to be factored. In other words, all of the scalar values in this problem are integers between 0 and $N-1$ and all computations on them are performed modulo N.

(a) Remainder trees. Given a degree $n-1$ polynomial f specified by its n coefficients

$$f_0, f_1, \ldots, f_{n-1},$$

and n evaluations points $a_0, a_1, \ldots, a_{n-1}$, describe a divide-and-conquer algorithm running in time $O(n \log^2 n)$ that outputs $f(a_0), f(a_1), \ldots, f(a_{n-1})$. Hint: use these facts: (1) if f is a polynomial and a is a scalar, $f(X) \bmod (X-a)$ equals $f(a)$, and (2) if f, g and h are polynomials, $(f(X) \bmod g(X)h(X)) \bmod g(X) = f(X) \bmod g(X)$.

(b) Let M be an integer. Find an algorithm that outputs the integer $M! \bmod N$, using $O(\sqrt{M} \log^2 M)$ operations. Hint: consider the polynomial $\prod_{i=0}^{\sqrt{M}-1}(X-i)$, and apply the previous part.

(c) Given N, M (with M satisfying $M \leq \sqrt{N}$), give an algorithm that determines whether there is a nontrivial factor (i.e., other than 1) of N that is at most M, using only $O(N^{1/4} \log^2 N)$ operations. You may use Euclid's algorithm which computes the greatest common divisor (GCD) of two n-bit integers in time $O(n^2)$, and the fact that $\mathrm{GCD}(x, y) = \mathrm{GCD}(x \bmod y, y)$.

It is now standard that by using part (c) and performing a binary search on M, we can find the smallest non-trivial factor of N, divide, and repeat, until N is completely factored. The resulting algorithm for factoring N has running time $O(N^{1/4})\cdot$(lower-order terms) as promised.