Algorithms – CS-27200/37000 Homework – January 16, 2004

Instructor: László Babai Ry-164 e-mail: laci@cs.uchicago.edu

READING Review all previous handouts. Study PSEUDOCODE conventions in handouts. By Monday: review graph theory (Discrete Math).

IMPORTANT. If you have not done so yet, please send e-mail to the instructor with your name, major, year, type of credit sought (letter grade, P/F, etc.), list of proof-oriented math courses previously taken; include whether or not you took CMSC-17400. In the subject write 27000 info or 37000 info, as approrpiate.

HOMEWORK. Please **print your name on each sheet.** Print "U" next to your name if you seek 27000 credit and "G" if you seek 37000 credit. Undergraduates receive the stated number of points as *bonus points* for "G only" problems. – Please try to make your solutions readable. Unless expressly stated otherwise, all solutions are due at the **beginning of the next class.**

Recall from class:

DEFINITION. A function $f(n)$ is **polynomially bounded** if

$$(\exists C)(f(n) = O(n^C)).$$

We say that C is an *admissible exponent.*

We say that an algorithm is **polynomial time** if the cost of the algorithm is polynomially bounded as a function of the *length of the input.* Note that this is a *worst-case* concept.

Recall the following characterization of polynomially bounded functions.

THEOREM. Suppose $f(n) \geq 1$ for all sufficiently large n. Then the function $f(n)$ is **polynomially bounded** if and only if $\ln f(n) = O(\ln n)$.

5.1 (G only) (3 points)] Can we omit the condition "$f(n) \geq 1$" in the Theorem? Prove your answer.

5.2 (U, G) (2 points) True or false: $\log_2(n) = \Theta(\ln n)$. Prove your answer.

5.3 (U, G) (4 points each) Decide whether or not each of the following functions is polynomially bounded. Prove your answers. If your answer is "yes," state all admissible exponents. (a) $n^2 \log n$; (b) $8^{\log n}$; (c) $2^{\sqrt{n}}$; (d) (G only) $(\lceil \log n \rceil)!$.

5.4 (U, G) (10 points) An algorithm takes positive integers x as input (in binary) and requires $\lfloor 2^{\sqrt{\log x}} \rfloor$ time. Is this a polynomial time algorithm? State and prove your answer. Give a proof of the required asymptotic relation. *Caveat.* Remember that the complexity of an algorithm is measured as a function of the *length* of the input. In case of integer inputs, this means the bit-length (total number of bits).

5.5 (U,G) (12 points) A "string over the English alphabet" means a sequence of letters of the English alphabet. A *substring* is obtained by deleting some of the letters. (It is permitted to delete all or to delete none.) Note that substrings *do not need to be contiguous.* Examples: ATTIC and HEAT are substrings of MATHEMATICS, but EMMA and HATE are not.)

Given two strings of respective lengths m and n over the English alphabet, find a longest common substring in $O(mn)$ steps. Example: $(m = 11, n = 15)$: input: MATHEMATICS, COMPUTERSCIENCE. Output: MTEIC.

Descibe your algorithm in *pseudocode.* (The strings are given as arrays of characters.) Name the method used.

If you introduce auxiliary variables, define them in English. A method of calculation is no substitute for a *simple and natural definition.* That definition is the key to the solution and account for at least half the credit. *Elegance counts.*

Hint. This is yet another member of a family of very elegant algorithms discussed in class and in handouts.

5.6 (U,G) (for your entertainment only, 0 points, do not hand in) Out of 12 given coins we know that one is counterfeit but we don't know whether it is heavier or lighter. Given a balance, using three measurements determine which coin is counterfeit and whether it is heavier or lighter than the others. (All the normal coins have the same weight. Each measurement on the balance can result in three outcomes: left-heavy (L), right-heavy (R), or equal (E).)

5.7 (U,G) (a) (5 points) Prove that 3 measurements do not suffice if we have 14 coins in the preceding problem. – Your solution should be very elegant, just a couple of lines. Use a method studied in class for proving lower bounds on complexity. Name the method.

(b) (G only, 10 points) Prove that 3 measurements do not suffice if we have 13 coins. *Elegance counts.* (Note: if you solve (b) then you solved (a) as well, but not in the most elegant way. You will not automatically receive the 5 points for (a), only if you separately give the most elegant solution to (a).)

5.8 (G only) **UNDERGRADS:** READ and UNDERSTAND the problem and the algorithm. EXPERIMENT with the algorithm. (Do not hand in unless you solve the problem for bonus points.)

In this problem, "graph" means undirected graph. Two edges are said to be *independent* if they do not share a vertex. A *matching* in a graph is a set of independent edges. (In other words, a matching in G is a spanning subgraph of G in which every vertex has degree ≤ 1.) A *maximum matching* is a matching of maximum size (max number of independent edges). The *greedy algorithm* for finding a maximum matching proceeds as follows:

Greedy_Matching(G)

The variable M maintains a growing list of independent edges.

```
0 Initialize: M := empty list
1 for e ∈ E(G) do
2     if e is independent of all edges in M then
3         add e to M
4     end(if)
5 end(for)
6 return M
```

(a) (6 points) Prove: this algorithm does not always return a maximum matching. Show that for every k there exists a graph with maximum matching size $2k$ where the algorithm returns a matching of size k only. For 3 bonus points, make your graphs connected.

(b) (6 points) Prove that the algorithm always returns a matching of size at least half of the maximum.

(c) (2 points) Estimate the number of steps taken by the algorithm in terms of the number of vertices (n) and the number of edges (m). Express your answer using the big-oh notation (ignore a constant factor). If we define $n + m$ to be the input size, is this a "polynomial time algorithms," i. e., is the number of steps polynomially bounded as a function of the input size?

Note that the result of the greedy algorithm depends not only on the graph but on the order in which its edges are accessed.