CORE EXAMINATION
Department of Computer Science
New York University
February 4, 2005

This is the common examination for the M.S. program in CS. It covers core computer science topics: Languages and Compilers, Operating Systems, and Algorithms. The exam has two parts. The first part lasts three hours and covers the first two topics. The second part, given this afternoon, lasts one and one-half hours, and covers algorithms.

Use the proper booklet or answer sheet for each question. Each booklet is marked with the Area and Question number, in the form PL&C1, PL&C2, PLC&C3, OS1, OS2, ALGS1 (this question has an answer sheet and not a booklet), ALGS2, ALGS3. Use the appropriate booklet for each question. DO NOT put your name on the exam booklet. Instead, your exam number must be on every booklet.

You will be graded according to your exam number, shown on the envelope containing the booklets. Remember your exam number: when grades are given out, they will be published according to this number, not by name.

Make sure your name and signature are on the envelope. This is the only place where your name appears. Please include all the booklets inside the envelope. You can keep the exam.

Good luck!

Question 1

Answer this question on the answer sheet for question 1, rather than in a booklet.

Part 1.

For each of the following pairs of functions, write a letter *a*–*d* in the corresponding box, as follows:

(a) if $f = O(g)$ and $g \neq O(f)$,

(b) if $f = O(g)$ and $g = O(f)$,

(c) if $f \neq O(g)$ and $g = O(f)$,

(d) if $f \neq O(g)$ and $g \neq O(f)$.

☐ $f(n) = n^2$, $g(n) = n^2 + 10n$

☐ $f(n) = 2^n$, $g(n) = 3^n$

☐ $f(n) = n^2$, $g(n) = n^3$

☐ $f(n) = \log_2 n$, $g(n) = \log_3 n$

☐ $f(n) = n^3$, $g(n) = n^2(\log_2 n)^{10}$

☐ $f(n) = 10^7 n + 10^8$, $g(n) = n/10^9$

☐ $f(n) = n^{1.5}(\log_2 n)$, $g(n) = n^{1.6}$

☐ $f(n) = n^{\log_2 3}$, $g(n) = 3^{\log_2 n}$

☐ $f(n) = \sum_{i=1}^{n} i$, $g(n) = n^{1.5}$

☐ $f(n) = 10^{\sqrt{\log_2 n}}$, $n^{0.001}$

Solution to Question 1, part 1. b, a, a, b, c, b, a, b, c, a.

Question 1, part 2.

In the following, you are presented with descriptions of recursive algorithms. For each, indicate the rate of growth of the running time by writing one of the letters *a*–*g* in the corresponding box to specify one of the following functions:

(a) $\log n$

(b) n

(c) $n \log n$

(d) n^2

(e) $n^2 \log n$

(f) n^3

(g) 2^n

☐ On inputs of size n, the algorithm spends a constant amount of time, plus the time needed to recursively solve one subproblem of size $n/2$. Assume n is a power of two.

☐ On inputs of size n, the algorithm spends time $\Theta(n)$, plus the time needed to recursively solve one subproblem of size $n-1$.

☐ On inputs of size n, the algorithm spends time $\Theta(n)$, plus the time needed to recursively solve two subproblems, each of size $n/2$. Assume n is a power of two.

☐ On inputs of size n, the algorithm spends a constant amount of time, plus the time needed to recursively solve two subproblems, each of size $n-1$.

☐ On inputs of size n, the algorithm spends time $\Theta(n^2)$, plus the time needed to recursively solve four subproblems, each of size $n/2$. Assume n is a power of two.

Solution to Question 1, part 2. a, d, c, g, e.

Question 2

A vertex v in directed acyclic graph (DAG) $G = (V, E)$ is said to be the root if for every vertex w in the graph (other than v), there is at least one path in G from v to w. Present an efficient algorithm to determine if a DAG G has a root. What is the operation count of your algorithm (up to constant factors) as a function of the number of elements in the graph? Justify your answer.

Please note: this question has three parts.

a) An algorithm is requested.

b) The running time (up to constant factors) is required.

c) A brief justification of your answer for part b is required.

Solution to Question 2 Solution 1. Sequence through the edges to compute the indegree of each vertex. This approach requires initializing each count to zero and incrementing the count for vertex z for each edge from y to z. Then the counts are post-processed to see if there is just one that remains at zero; if so the vertex with count zero is the root of the DAG.

Solution 2. Do not work that hard. Just process the each edge (y, z) by marking vertex z. Then check to see if exactly one vertex w remains unmarked; again, w is the root in this case.

Solution 3. Use topolological sort to get a possible root z. Then run a DFS from z to see if all of the vertices are reached. If so, z is the root.

b) In each case, the run time is $\Theta(|V| + |E|)$.

c) Justifications: Each of the algorithms processes each edge just once, (or twice, for Solution 3), and has an initialization and/or post-processing phase that is either linear in the number of vertices or (as already mentioned for the post-processing for Solution 3) linear in $|V| + |E|$.

Question 3

Let G be a directed graph on the vertices $\{1, 2, 3, \ldots, n\}$, and let $Dis[i, j]$ contain the length of the edge (i, j) if it exists, and have the value ∞ if no such edge is in G. Suppose that every cycle in G has edges with lengths that have a positive sum.

The standard All-Pairs-Shortest-Paths problem is to compute, for all vertex pairs i, j the length of the shortest path from i to j. The usual algorithm used to solve this problem is the Floyd-Warshall algorithm. For this problem you are to compute, for all vertex pairs i, j, the lengths of the shortest and second shortest paths from i to j. (Note: If there are two different paths from i to j with the same shortest value, then the shortest and second shortest path lengths are both this best value. We also note that two paths are different if they are not the identically same sequence of edges.)

For convenience, you can use the function $\min\{*\}$, which returns the smallest value in its sequence of arguments so that, for example, $\min\{2,7,1,8\} = 1$.

Likewise, you can use *second* $\min\{*\}$, which return the second smallest value in its sequence of arguments so that, for example, *second* $\min\{2,7,1,8\} = 2$, and *second* $\min\{2,7,1,8,1\} = 1$.

Solution to Question 3 We want to compute two $n \times n$ matrices A, B such that for all i, j, $A(i,j)$ and $B(i,j)$ are the lengths of the shortest path and the second shortest path from i to j.

It follows that $A(i,j) \leq B(i,j) \leq \infty$. Also $B(i,j) = \infty$ iff there is at most one path from i to j.

For $k = 0, 1, \ldots, n$, define the matrix A^k where $A^k(i,j)$ is the length of the shortest path whose intermediate nodes are restricted to come from the set $\{1, \ldots, k\}$. Thus $A(i,j) = A^n(i,j)$. The matrix B^k is defined similarly.

The algorithm can now be given. The main part of the algorithm is a triply-nested loop. In the kth iteration of the outermost loop, the matrix A is a hybrid between A^{k-1} and A^k, and similarly for B. We also write $\min_2\{...\}$ for the operation that picks the second smallest value in a multiset. We may assume $Dis(i,i) = 0$.

```
INITIALIZATION:
    For $i = 1$ to $n$
        For $j = 1$ to $n$
            $A(i,j) = Dis(i,j)$; $B(i,j) = \infty$
MAIN LOOP:
    For $k = 1$ to $n$
        For $i = 1$ to $n$
            For $j = 1$ to $n$
                $A(i,j) = \min\{A(i,j), A(i,k) + A(k,j)\}$
                $B(i,j) = \min_2\{A(i,j), B(i,j),$
                    $A(i,k) + A(k,j), B(i,k) + A(k,j), A(i,k) + B(k,j)\}$
```

CORRECTNESS. The correctness of A is standard from Floyd-Warshall. The correctness of B is similarly justified: the second shortest path either passes through k or it does not. If not, it must be $A(i,j)$ or $B(i,j)$. If it does, it must be one of the 3 sums in the $\min_2$ expression. It is easy to give examples showing that we need all five terms in the $\min_2$ expression.