(VERSION WITH ANSWERS)
CORE EXAMINATION
Department of Computer Science
New York University
February 6, 2009

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

You will be assigned a seat in the examination room.

Use the proper booklet or answer sheet for each question. Each booklet is marked with the Area and Question number, in the form PL1, PL2, OS1, OS2, ALG1, ALG2, and ALG3. DO NOT put your name on the exam booklet or answer sheet. 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!

ALGORITHMS PART

Question 1 – please use the Exam Booklet labeled ALG1

Graphs

Suppose $G(V, E)$ is an undirected graph given in the adjacency list representation. Let n be the number of nodes and m be the number of edges in the graph. Recall that a **tree** is a connected graph that has no cycles. (Note that such a tree is sometimes called a "free tree".)

(a) (1 Point) How many edges does a tree with n nodes have?

Answer: $n-1$.

(b) (3 Points) In a tree, a node with degree one is called a **leaf**. What is the maximum and minimum number of leaves an n node tree can have? In both cases, you must give an example.

Answer: Maximum is $n-1$ for a star graph. Minimum is 2 for a line graph.

(c) (6 Points) Give an efficient algorithm to detect if $G(V, E)$ is a tree. What is the running time of your algorithm?

Answer: Pick one node s of the graph and do breadth-first search from s. BFS partitions the set of nodes reachable from s into layers $L_0 = \{s\}, L_1, \ldots, L_k$. The graph is connected iff all nodes of the graph are included in these layers. There is no cycle in the graph iff there is no edge whose both endpoints are in the same layer. Overall running time is $O(m+n)$.

Question 2 – please use the Exam Booklet labeled ALG2

Running Time Analysis

In this question, all the (undeclared) programming variables are integers, and program blocks are indicated by indentation.

Give the order of the running time for each of the following procedures. Briefly justify your answers.

(a) (3 Points) Let $T_1(n)$ be the running time of the following procedure:

```
Mystery(n)
    s ← 1
    for i ← 1 to n
        s ← s + 1
        for j ← 1 to 1000
            for k ← n to n − j
                s ← s + i + j + k
```

Answer: $T_1(n) = O(n)$ because the two inner loops (on variables j and k) only has a constant number of iterations.

(b) (3 Points) Let $T_2(n)$ be the running time of the following procedure, where $A[1, \ldots, n]$ is a global array of integers.

```
Puzzle(i, m)
    ▷ Input Assertion: (1 ≤ i ≤ i + m ≤ n) and (0 ≤ m)
    if (m = 0) return A[i]
    if (A[i] > A[i + m])
        return A[i+m]+ Puzzle(i, ⌊m/3⌋)
    else
        return A[i] + Puzzle(i + ⌊m/3⌋ , ⌊2m/3⌋)
```

Answer: The argument m indicates the size of the subarray $A[i, \ldots, i + m]$ that is potentially used. In each recursive call, the size of this subarray is reduced to at most 2/3 of the input size. Thus $T_2(n) = O(1) + T(2n/3) = O(\log n)$.

(c) (4 Points) Let $T_3(n)$ be the running time of the following procedure,

```
What(n)
    x ← 0
    for i ← 1 to ⌊√n⌋
        for j ← 1 to ⌊√n⌋
            for k ← 1 to ⌊√n⌋ − j + 1
                x ← x + 1
```

Answer: Each of the three loops have $O(\sqrt{n})$ iterations. Hence $T_3(n) = O((\sqrt{n})^3) = O(n\sqrt{n})$.

Question 3 – please use the Exam Booklet labeled ALG3

Finding Duplicates

Let A be an array of size n with entries in the set $\{1, 2, \ldots, n-1\}$. Observe that this implies at least one duplicated entry.

(a) [1 point] Give a simple linear-time algorithm to find a duplicated value. Your algorithm should use no more than $O(n)$ additional storage. Hint: One way to solve this problem is by adapting an algorithm that computes a count of each of the numbers $\{1, \ldots, n-1\}$ in the data set.

Answer: Count the number of entries with value i using an array B of size $n-1$ initialized to zero. If $B[i] = 2$, then i is duplicated. Storing B requires $O(n)$ additional space.

(b) (1 Point) Assume now that no more than additional constant space can be afforded but that the array A can be modified. Give a simple algorithm with average-case running time $O(n \log n)$ to find a duplicated value.

Answer: Use quicksort. Then scan the sorted array, looking for two consecutive entries that are duplicated.

(c) (2 Points) In this question alone, it is assumed that exactly one entry is duplicated. Give a simple linear-time algorithm to determine the duplicated value, without creating an additional array and without modifying the original array, using the sum of the entries.

Answer: Sum all the entries and compare to $S = n(n+1)/2$.

(d) (2 Points) For any array index i, let $A^2[i]$ denote $A[A[i]]$ and more generally $A^k[i] = A[A^{k-1}[k]]$ for $k \geq 2$.

Show that there exist positive integers i and k, $0 < i + k \leq n$, such that $A^i[n] = A^{i+k}[n]$.

Answer: $A^k[n]$ takes its values in $\{1, 2, \ldots, n-1\}$, thus, there is at least one repetition in the sequence $A^1[n], A^2[n], \ldots, A^n[n]$.

(e) (2 Points) For i and k as in the previous question, we can see that

$$A^i[n] = A^{i+k}[n], \quad A^{i+1}[n] = A^{i+k+1}[n], \quad \cdots, \quad A^{i+q}[n] = A^{i+k+q}[n].$$

Let q be the smallest integer such that $i + q$ is a multiple of k. Let $u = i + q$, show that $u \leq n$ and $A^{2u}[n] = A^u[n]$. Hint: observe that $A^{u+pk}[n] = A^u[n]$ for any non-negative integer p.

Answer: The hint is straightforward by iteration from $A^u[n] = A^{u+k}[n]$. One of $i, i+1, \ldots, i+k-1$ is 0 modulo k. So, choose q ($0 \leq q < k$) such that $i+q$ is 0 modulo k. Also $i+k \leq n$ by the previous part, thus $u = i+q \leq n$.

(f) (2 Points) Show that the following is the pseudocode of an algorithm returning a duplicated value in linear time, without creating an additional array and without modifying the original array. HINT: You need can assume the results from the previous sections, even if you did not prove them.

```
1  x ← A[n]; y ← A[A[n]]
2  while (x ≠ y)
3      do x ← A[x]; y ← A[A[y]]
4  x ← n
5  while (x ≠ y)
6      do x ← A[x]; y ← A[y]
7  return x
```

Answer: By part (d), the first loop takes no more than $u \leq n$ steps. The second loop finds the first v such $A^{u+v}[n] = A^v[n]$, and by part (e), this takes no more than n steps.