CMPS 256 — ADVANCED ALGORITHMS AND DATA STRUCTURES
Fall 2012 – 2013 Semester
Final Exam
Wednesday January 9, 8:00 a.m.
2 hours

**Instructions.**

This final is scored out of 100.
This final is open book and open notes: you can use the textbook, notes you have taken in class, your homework solutions, and all material that I have posted to the course Moodle site.
Show your work, as partial credit will be given.
You may use any algorithm that was covered in class: give the name of the algorithm and a page number in the textbook where the algorithm is presented. If taken from your lecture notes, just say "lecture notes."

**Please start each answer on a new page.**

**Exam policy: if you are unsure about the meaning of a question, raise your hand and I will come to you. I will discuss only what is on the exam sheet. I will NOT discuss anything that you have written down.**

Best of luck!

**Problem 1. (20 points)** State whether the following are true or false (5 points each)

*a.* $n^3 = \Omega(n^3)$

**Sample solution.** True

*b.* $\lg(n^n) = \Theta(n \lg n)$

**Sample solution.** True

*c.* $\lg(n^n) + n^2 = \Theta(n \lg n)$

**Sample solution.** False

*d.* $n^4 + 4n^3 + 5n^2 + 7 \sim 2n^4$

**Sample solution.** False

**Problem 2. (20 points)** Consider the following recurrence

$$T(n) = 3T(n/3) + n.$$

*a. (10 points)* Draw the recursion tree for this recurrence.

*b. (10 points)* Solve the recurrence by summing up the recursion tree. GIve a tight bound, i.e., a solution of the form $T(n) = \Theta(f(n))$. Half credit for proving the upper bound only, or proving the lower bound only.

**Sample solution.** Level $i$ of the recursion tree has $3^i$ nodes, each with input $n/3^i$. Each level has cost $n$. The height is $\log_3 n$. Hence cost of tree is $n \log_3 n$, which is $\Theta(n \lg n)$.

**Problem 3. (30 points)** You are given an array $A$ of size $n$. An element $A[i]$ is *out-of-order* iff $A[i] < max(A[0..i-1])$ where $max(A[0..i-1])$ is the maximum element in $A[0], \ldots, A[i-1]$. In other words, an element is out-of-order if it is smaller than the maximum of all the elements below it.

You are given that $A$ contains $k$ out-of-order elements, with $k \leq n/\lg n$. You are not given the actual value of $k$.

Give pseudocode for an algorithm to sort $A$ in time $O(n)$.

Grading is as follows:

- 10 points for the code itself, provided that it is actually correct and runs in worst-case time $O(n)$
- 10 points for an informal argument which proves that your code is correct, i.e., actually sorts $A$
- 10 points for a running time analysis which proves that your code runs in worst-case time $O(n)$

**Sample solution.** Pseudocode:

Traverse $A$, maintaining the maximum $m$ of the elements that are not out-of-order. That is, if the next element is larger, then set $m$ to it, otherwise leave $m$ unchanged.

Use $m$ to determine whether the next element is out-of-order ($< m$) or not ($\geq m$).

While traversing, copy all elements that are not out-of-order to another array $B$, and copy all elements that are out-of-order to a third array $C$.

By its construction, $B$ is ordered.

$C$ is not necessarily ordered, and has size $k$. Sort $C$ using mergesort in time $O(k \lg k)$.

Merge $B$ and $C$ to give the result of sorting $A$.

Running time:

Merge and traversal take time $O(n)$. So total running time is $O(n + k \lg k)$. Now $k \leq n/\lg n$, so $k \lg k \leq (n/\lg n) \lg(n/\lg n) = (n/\lg n)(\lg n - \lg\lg n) = n - n \lg\lg n/\lg n = O(n)$. Hence total running time is $O(n + k \lg k) = O(n)$.

**Problem 4. (30 points)** Let $G = (V, E)$ be a directed graph. A *source node* in $G$ is a node with no incoming edges. Write pseudocode for an algorithm that determines if $G$ contains a source node. Your algorithm should have **average case** running time $O(V + E)$.

Grading is as follows:

- 10 points for the code itself, provided that it is actually correct and runs in average-case time $O(V + E)$
- 10 points for an informal argument which proves that your code is correct, i.e., correctly determines if $G$ contains a source node
- 10 points for a running time analysis which proves that your code runs in average-case time $O(V + E)$

Reduced credit (21 points) for an algorithm with **worst case** running time $O(E \lg V)$.

**Sample solution.** Use a hash table $H$. Use separate chaining with array of size $V/5$, or linear probing with array of size $2V$, which gives average case constant time operations. First insert all nodes of $G$ into $H$. This takes time $O(V)$ on average. Now traverse all the adjacency lists, and remove from $H$ any node found on some adjacency list, which therefore has an incoming edge, and so is not a source. This takes time $O(E)$ on average. After traversal, any nodes left in $H$ are source nodes. So $G$ contains a source node iff $H$ is not empty. Can check emptiness of $H$ in $O(V)$ worst case time. Total running time is $O(V + E)$ average case.