Illinois Institute of Technology
Department of Computer Science

Second Examination

CS 430 Introduction to Algorithms
Spring, 2012

11:25am–12:40pm, Wednesday, March 14, 2012
104 Stuart Building

Print your name and student ID, *neatly* in the space provided below; print your name at the upper right corner of *every* page. Please print legibly.

Name:
Student ID:

This is an *open book* exam. You are permitted to use the textbook, any class handouts, anything posted on the web page, any of your own assignments, and anything in your own handwriting. Foreign students may use a dictionary. *Nothing else is permitted*: No calculators, laptops, cell phones, Ipods, Ipads, etc.!

Do all four problems in this booklet. *All problems are equally weighted, so do not spend too much time on any one question.*

Show your work! You will not get partial credit if the grader cannot figure out how you arrived at your answer.

Question	Points	Score	Grader
1	25		
2	25		
3	25		
4	25		
Total	100		

1. **Augmenting Red-Black Trees**

   Define $EPL(x)$, the *external path length* of the subtree rooted at a node x in a red-black tree T as in the lecture notes for February 1,

   $$EPL(x) = \sum_{l \text{ is a leaf of subtree } x} depth_x(l),$$

   where $depth_x(l)$ is the depth of leaf l in the subtree rooted at x. Can a red-black tree be augmented with the $EPL(x)$ values without affecting the $O(\log n)$ performance of the insertion and deletion algorithms? Prove your answer.

2. **Dynamic Programming**

We are given a sequence of n numbers, $a_1, a_2, \ldots, a_n$ and want to find the *longest increasing subsequence* (LIS); that is, we want to find indices $i_1 < i_2 < \cdots < i_m$ such that $a_{i_j} < a_{i_{j+1}}$ and m is as large as possible. For example, given the sequence 5, 2, 8, 6, 3, 6, 9, 7 we have an increasing subsequence 2, 3, 6, 9 and there is no longer increasing subsequence.

(a) Give an recursive dynamic programming recurrence for the LIS of a sequence $a_1, a_2, \ldots, a_n$.
(*Hint*: Let L_j be the length of the longest increasing subsequence in $a_1, a_2, \ldots, a_j$ and let A_j be index of the largest element in that increasing subsequence. Express L_j recursively.)

(b) Analyze, as a function of n, the time required to evaluate directly your recurrence in (a).

(c) Give memoized code for (a).
(*Hint*: Use an array $L[j]$ for the length of the LIS among $a_1, a_2, \ldots, a_j$, and an array $A[j]$ for the index of the largest element of the LIS among $a_1, a_2, \ldots, a_j$.)

(d) Analyze, as a function of n, the time required by your code in (c).

2. **Dynamic Programming, continued.**

3. **Amortized Analysis**

We are going to use the stack operations of the Chapter 17 of CLRS to store objects $o_1, o_2, \ldots$ as they arrive with PUSH operations. A POP operation removes the top stack element, just as in CLRS. However unlike CLRS, a MULTIPOP(k) operation not only removes the top k elements, but must also report if there are any duplicate elements among the k removed; because we know nothing about the objects being stored, the only way to determine duplicates is to compare all $\binom{k}{2} = k(k-1)/2$ pairs of elements being removed from the stack. A MULTIPOP(k) operation thus requires time $\Theta(k^2)$.

Using the potential function $\Phi(D) = |D|^2$, the square of the number of elements on the stack, show that any sequence n of PUSH, POP, and MULTIPOP operations takes time $\Theta(n^2)$.

4. **Fibonacci Heaps**

The last problem in the sample second exam was as follows:

> We decide that we don't want to bother with the mark bits in Fibonacci heaps, so we change the cascading-cut rule to cut a node from its parent as soon as it loses its *first* child. Is still true that $D(n) = O(\log n)$? Prove your answer.

The posted solution said yes, $D(n)$ is still $O(\log n)$ and went on to prove an analogue of Corollary 19.5 on page 526 for the modified Fibonacci heaps showing that the maximum degree $D(n)$ of an node in an n-node modified Fibonacci heap is $\lfloor \lg n \rfloor$.

But the problem ignored an important question: Does the amortized time analysis still hold? Explain why or why not for each of the heap operations analyzed in chapter 19 of CLRS (insertion, union, consolidation, extracting the minimum, decreasing a key, and deleting a node).