Illinois Institute of Technology
Department of Computer Science

Second Examination

CS 430 Introduction to Algorithms
Spring, 2016

Wednesday, March 9, 2016
10am–11:15am & 11:25am–12:40pm, 111 Life Sciences

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. **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 a 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 LIS in $a_1, a_2, \ldots, a_j$, let A_j be index of the smallest possible largest element in that increasing subsequence, and let B_j be index of the second-largest element in that increasing subsequence. Express L_j recursively. You may assume a dummy element $a_0 = -\infty$.)

   (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 three arrays as memos: $L[j]$ for the length of the LIS among $a_1, a_2, \ldots, a_j$, $A[j]$ for the index of the largest element of the LIS among $a_1, a_2, \ldots, a_j$, and B_j for the index of the second largest element in the LIS.)

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

   (e) Explain how to use the arrays A_j and B_j to recover the elements of the LIS found by your code in (a) or (c).

1. **Dynamic Programming, continued.**

2. **Greedy Algorithms**

You are planning purchases of equipment for a business. Bureaucratic problems limit you to one purchase per month, but for each month you delay a purchase, the price of the piece of equipment increases at some rate greater than 1. You need to purchase n pieces of equipment, $E_1, E_2, \ldots, E_n$ where E_i costs \$100 if purchased in the first month, \$100 $\times$ r_i if purchased in the second month, \$100 $\times$ r_i^2 if purchased in the third month, and so on. The pieces of equipment can be purchased in any order.

(a) Give an example for which the greedy strategy "lowest rate first" fails to be the cheapest.

(b) Prove that the greedy strategy "highest rate first" always gives the cheapest way to schedule purchases.

(*Hint*: If not, when does the strategy make its first mistake?)

(c) What is the running time of an algorithm based on the "highest rate first" strategy?

2. **Greedy Algorithms, 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. **Disjoint Sets**

   Give a sequence of m MAKE-SET, UNION, and FIND-SET operations, n of which are MAKE-SET operations, that takes time $\Omega(m \log n)$ time when we use union by rank only (without path compression).