Illinois Institute of Technology
Department of Computer Science

First Examination

CS 430 Introduction to Algorithms
Fall, 2014

Monday, September 29, 2014
10am–11:15pm, 121 Life Sciences
11:25am–12:40pm, 113 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 five 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	20		
2	20		
3	20		
4	20		
5	20		
Total	100		

1. **Time Bounds**

   Suppose you are given five algorithms with the following exact running times in terms of the input size n:

   (a) $T(n) = T(n-1) + 100$

   (b) $T(n) = T(n-1) + n$

   (c) $T(n) = T(n-1) + T(n-2)$

   (d) $T(n) = T(n-1) + 1/n$

   How much *faster* do these algorithms become when you halve the size of the input? Justify your answers!

2. **Searching a Sorted Two-Dimensional Array**

   You are given a two-dimensional $n \times m$ array of elements A_{ij} in which every row and every column is in increasing order (that is, $A_{ij} < A_{ik}$ if $j < k$ and $A_{ij} < A_{kj}$ if $i < k$). You want to search A for a value x that may or may not be in the array. Show how to do the search in worst-case $n + m - 1$ element comparisons.

   (*Hint*: Consider a zig-zagging "linear" search starting at the upper right corner of the array.)

3. **Quicksort Comparisons**

As in section 7.4.2 in CLRS3 (pages 181–184), let $z_1, z_2, \ldots, z_n$ be the array elements in increasing order *after sorting* by QUICKSORT.

(a) What is the largest number of element comparisons during QUICKSORT that could have involved z_1?

(b) What is the smallest number of element comparisons during QUICKSORT that could have involved z_1?

(c) What is the probability that QUICKSORT made the comparison $z_1 : z_2$ (that is, the probability that QUICKSORT compared the two smallest elements)?

(d) What is the probability that QUICKSORT made the comparison $z_1 : z_3$ (that is, the probability that QUICKSORT compared the smallest element with the third smallest element)?

(e) What is the expected number of comparisons in QUICKSORT that involve z_1?

4. **Heaps**

A *min-max heap* is a data structure that permits the retrieval of *both* the minimum *and* the maximum elements in a set in $O(1)$ time; the min-max heap can be constructed in $O(n)$ time, while INSERT, DELETEMAX and DELETEMIN take $O(\log n)$ time. In a min-max heap, the key of a node at *odd* level is no larger than the keys of any of its descendants, whereas the key of a node at *even* level is at least as large as the keys of any of its descendants; consider the root as level 1. Thus, the minimum element in the min-max heap appears at the root, whereas the maximum element appears as one of its two children.

Assuming that the min-max heap is stored in an array A in the same manner as an ordinary heap, give an $O(\log n)$ implementation of insertion into a min-max heap.

(*Hint*: The basic idea behind insertion is the same as in an ordinary heap.)

5. **Red-Black Trees**

In Lectures 6 and 7 (September 15–17) we discussed the extreme values of the external path length in binary trees.

(a) Show that the nodes of any $(n + 1)$-leaf binary tree with *minimal* external path length can be colored red and black to make it a proper red-black tree.

(b) Is that statement true for any binary tree with *maximal* external path length?