Illinois Institute of Technology
Department of Computer Science

Second Examination—Early Administration

CS 430 Introduction to Algorithms
Fall, 2010

6pm–7:15pm, Tuesday, November 2, 2010
Rice Campus

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.

In order for your examination to be graded, you *must* sign your name in the box, agreeing to the stated condition.

I agree, on penalty of failure of CS 430, not to disclose to anyone any details of the contents of this examination until 12:40pm, Wednesday, November 3, 2010.

Question	Points	Score	Grader
1	20		
2	20		
3	20		
4	20		
5	20		
Total	100		

1. **Augmenting Red-Black Trees**

   (a) Define *width*(x), the *width* of a node x in a red-black tree T, as follows: The width of a leaf is 0. The width of a non-leaf node x is the width of its wider child, if the children of x have unequal widths; it is one plus the width of the child if the children of x have equal widths. Can a red-black tree be augmented with the *width*(x) values without affecting the $O(\log n)$ performance of the insertion and deletion algorithms? Prove your answer.

   (b) Define *depth*(x), the depth of a node x in a red-black tree T, as the number of number of nodes in the tree that are ancestors of x. Can a red-black tree be augmented with the *depth*(x) values without affecting the $O(\log n)$ performance of the insertion and deletion algorithms? Prove your answer.

2. **Dynamic Programming**

A roofer has a ladder with n rungs, labeled 1 through n. To climb onto the roof, the roofer must reach the top (nth) rung of the ladder, starting from the ground (0th rung). The roofer's legs are long enough to go up either 1 or 2 rungs at a time. Stepping on the ith rung has a cost of c_i dollars; assume $c_0 = 0$.

(a) Let $C(k)$ be the minimum cost of going from the ground to step k. Give a recurrence relation for $C(k)$, including initial conditions, based on the Principle of Optimality.

(b) Analyze a recursive, *unmemoized* algorithm based on your recurrence in part (a); you need not write the algorithm.

(c) Give a memoized, iterative algorithm based on (a), and analyze it.

(d) What additional memoization is needed to determine the *sequence of rungs* used by the roofer in the cheapest way to the top?

3. **Greedy Heuristics**

Professor Reingold wants to drive from Chicago to Seattle along I-90, stopping as infrequently as possible to buy gasoline. His GPS continuously shows him the distance to the next gas station and his dashboard continuously displays the number of miles he can drive on his remaining fuel, so that as he approaches a gas station he can tell if he has enough gas to make it to the following station.

(a) Give a greedy algorithm for him to use in choosing where to stop for gas.

(b) Prove that your greedy algorithm has him stopping the fewest number of times possible.

4. **Amortized Analysis**

We saw in class (October 13) and in CLRS (Chapter 17) that a k-bit binary counter can be incremented in $O(1)$ amortized time (bit changes). Now we want to add a RESET operation that sets the counter to 1. We implement this operation by keeping track of m, the index of the most significant (left-most) 1-bit in the counter. For simplicity we assume that the counter has an unbounded number of bits. The INCREMENT operation becomes

```
1: INCREMENT(A, m)
2: i = 0
3: while A[i] == 1 do
4:     A[i] = 0
5:     i = i + 1
6: end while
7: A[i] = 1
8: if i > m then
9:     m = i
10: end if
```

and the RESET operation is

```
1: RESET(A, m)
2: for i = 0 to m do
3:     A[i] = 0
4: end for
5: A[0] = 1
6: m = 0
```

(a) What is the worst-case running time of RESET?

(b) Assume that the counter starts at 1 with $m = 1$. Use the potential function

$$\Phi(A) = m + \text{number of 1-bits in } A$$

to prove that the amortized time per operation of any intermixed sequence of RESET and INCREMENT operations is $O(1)$.

4. **Amortized Analysis, continued.**

5. **Heaps**

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.

(*Hint*: This is based on exercise 19.4-2 on page 526 of CLRS; it was suggested as a good study problem in the lecture of October 25 and covered in a more general fashion in the TA's Friday recitation session of October 29.)