Illinois Institute of Technology
Department of Computer Science

Second Examination

CS 430 Introduction to Algorithms
Spring, 2018

Wednesday, March 7, 2018
10am–11:15am & 11:25am–12:40pm
111 Robert A. Pritzker Science Center

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. **Augmented Red/Black Trees**

   Recall the 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 January 31 (Lecture 7),

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

   where $\text{DEPTH}_x(l)$ is the depth of leaf l in the subtree rooted at x. Let $\text{SIZE}(x)$, the number of (internal) nodes in the subtree rooted at x. Consider a red-black tree augmented with a combination of both EPL(x) and $\text{SIZE}(x)$. Can such a red-black tree be maintained without affecting the $O(\log n)$ performance of the insertion and deletion algorithms? Prove your answer.

2. **Dynamic Programming**

Various positive integers are arranged in an $N \times N$ triangular array T such as

					(k=0)					
($n = 0$)					②₀		(k=1)			
($n = 1$)				5		④₁		(k=2)		
($n = 2$)			3		4		⑦₂		(k=3)	
($n = 3$)		1		6		⑨₃		6		(k=4)
($n = 4$)	3		4		⑩₄		5		2	(k=5)
($n = 5$)	2		4		8		⑯₅		3	1

We want to find the largest sum in a descent from the apex ($n = 0$, $k = 0$) to the base ($n = 5$, in this case) through a sequence of adjacent numbers, one per each level, as shown by the circled values.

(a) Write a dynamic programming recurrence for the maximum sum.

(b) If you implement your recurrence *without memoization*, what is its running time (in terms of N)?

(c) If you memoize your implementation, what is its running time (in terms of N)?

(d) What additional memoization is needed to determine the *sequence of steps* used in to obtain the largest sum?

3. **Greedy Scheduling Variant**

Professor Reingold decided to sort the jobs in the activity-selection problem of section 16.1 in CLRS3 in *decreasing order by starting time* (instead of increasing order by finishing time).

(a) Give an example of a set of at least three jobs for which this greedy approach gives the best solution.

(b) Give an example of a set of at least three jobs for which this greedy approach does *not* give the best solution, or prove that no such example exists (that is, that this greedy heuristic always gives an optimum schedule).

4. **Amortized Analysis**

Consider the following modified version of the INCREMENT code on page 454 of CLRS3:

```
 1: INCREMENT(A)
 2: i ← 0
 3: while i < A.length and A[i] = 1 do
 4:     A[i] ← 0
 5:     i ← i + 1
 6: end while
 7: if i < A.length then
 8:     A[i] ← 1
 9:     WASTETIME(i)
10: end if
11: return
```

The modification is line 9, where we have added the call to WASTETIME(i). Let $T(i)$ be the worst-case running time of WASTETIME(i). We know from the discussion in CLRS3 on pages 461–462 that INCREMENT runs in amortized time $O(1)$ when $T(i) = 0$.

Use the potential function method paralleling the discussion on pages 461–462 to determine the amortized time of INCREMENT if $T(i) = i$.