15-451 Algorithms, Fall 1999

Midterm **Name:** _____

This test is open book and one sheet of notes is allowed. You have 80 minutes and there is a total of 80 points. Questions are of varying difficulty.

(20 pts) 1. Suppose we had code lying around for implementing a stack, and we now wanted to implement a *queue*. One way to do this is to use *two* stacks $S1$ and $S2$. To insert into our queue, we push into stack $S1$. To remove from our queue we first check if $S2$ is empty and if so we dump $S1$ into $S2$. Then we pop from $S2$. For instance, if we first insert an A and then insert a B, then $S1$ will look like (B A) (top of stack is to the left). Then if we do a remove, we will first dump $S1$ into $S2$ so that $S1$ is now empty and $S2$ looks like (A B). Then we pop the A from $S2$. Here is pseudocode:

- insert(x): $S1$.push(x)
- remove(): If $S2$ is empty then dump($S1, S2$). Return $S2$.pop().
- dump($S1, S2$): while $S1$ is not empty do $S2$.push($S1$.pop()).

Let's say that each push costs \$1 and each pop costs \$1, and performing a dump when $S1$ has n elements costs $2n$ dollars (since we do n pushes and n pops).

(a) Suppose that (starting from an empty queue) we insert 3 elements, then remove 2 of them, then insert 3 more, and then remove 2 again. What is the total cost of these 10 operations, and how many elements are in each stack at the end?

(b) Circle the smallest correct answer. Starting from an empty queue, the amortized cost per operation of this strategy is at most: \$1 \$2 \$3 \$4 \$5

(c) Give a proof of your answer to part (c) using the banker's (piggy bank) method.

(20 pts) 2. Hashing

(a) Here is a set of four hash functions from the universe $U = \{0, 1, 2, 3\}$ to $\{0, 1\}$. Is this set universal? Why or why not? Be specific.

	0	1	2	3
h1	0	0	1	1
h2	0	1	0	1
h3	1	0	1	0
h4	0	1	0	0

(b) Let h_1 and h_2 be two hash functions from a universe U to the range $\{0, \ldots, M-1\}$. In class we proved that if $|U| \geq NM$, then there exists a set $S \subset U$ of N elements that all hash to the same location under h_1.

Prove that if $|U| \geq NM^2$ then there exists a set $S \subset U$ of N elements that all hash to the same location under h_1 *and* all hash to the same location under h_2.

(25 pts) 3. Short-answer questions

(a) We've proven that sorting n elements requires $\Omega(n \log n)$ comparisons. However, you *can* sort 3 elements with 3 comparisons. Why is this not a contradiction?

(b) Solve the recurrence $T(n) = 2T(n/3) + T(n/4) + n$ in Theta notation. Briefly explain your answer.

(c) Suppose you have a deck of 52 cards numbered 1 to 52 in order. You then shuffle the cards until the ordering is completely random. What is the expected number of cards that end in the same position as they started? Explain.

(d) If all the edges in a graph have different lengths, then there is only one Minimum Spanning Tree.

i. Prove that the MST must contain the shortest edge and the second-shortest edge. (Assume $|E| \geq 2$)

ii. Give an example of such a graph (with at least 3 edges) in which the MST does *not* contain the 3rd-shortest edge.

(15 pts) 4. Consider a test with n problems. Problem i has a point value v_i, and requires t_i minutes to solve (and there is no partial credit). In class we looked at the Knapsack problem which corresponds to the question: "given an integer T, what is the maximum value V achievable in T minutes?" The running time of this algorithm was $O(nT)$. Your goal is to come up with an algorithm whose running time is $O(nV)$. For convenience, we will flip the problem around and ask: "given an integer V, what is the *minimum* time T needed to achieve value at least V?" (e.g, what is the least work we need to do to get an A).

(a) Let $Time(i, v)$ denote the minimum time needed to achieve value $\geq v$ using only problems $\{1, \ldots, i\}$ (or infinity if it is not possible to achieve value v). Write a recurrence for $Time(i, v)$:

$Time(i, v) =$

We'll do the base cases for you:

$Time(i, 0) = 0$ for all i. (And, $Time(i, v) = 0$ if $v \leq 0$.)

$Time(0, v) = \infty$ if $v > 0$

(b) Give pseudocode for a dynamic programming (bottom-up or top-down) algorithm to solve this version of the knapsack problem in time $O(nV)$.