University of New Mexico
Department of Computer Science

Midterm Examination

CS 362 Data Structures and Algorithms
Spring, 2008

Name:
Email:

- Print your name and email, *neatly* in the space provided above; print your name at the upper right corner of *every* page. Please print legibly.
- This is an *closed book* exam. You are permitted to use *only* two pages of "cheat sheets" that you have brought to the exam and a calculator. *Nothing else is permitted.*
- Do all five problems in this booklet. *Show your work!* You will not get partial credit if we cannot figure out how you arrived at your answer.
- Write your answers in the space provided for the corresponding problem. Let us know if you need more paper.
- Don't spend too much time on any single problem. The questions are weighted equally. If you get stuck, move on to something else and come back later.
- If any question is unclear, ask us for clarification.

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

1. **Short Answer**

For each of the questions below, please give the answer in terms of theta notation. Circle your final answer.

(a) Amount of time necessary to solve the 0-1 knapsack problem using dynamic programming when there are n items and the capacity of the knapsack is W.

(b) In the analysis we went over in class on the union find data structure with path compression, number of dollars paid into the leader and child accounts for each call to Find-Set.

(c) In the analysis we went over in class on the union find data structure with path compression, number of dollars paid into the block and path accounts over n calls to Find-Set.

(d) Solution to the recurrence $T(n) = 8T(n/2) + \sqrt{n}$

(e) Solution to the recurrence $T(n) = T(n-1) + 1$

(f) Solution to the recurrence $T(n) = 4T(n-1) - 3T(n-2) + 1$

True or False (10 points total). **Circle your final answers.**

(a) If an operation takes $O(1)$ expected time, then it takes $O(1)$ amortized time

(b) Any problem that can be solved with a greedy algorithm can also be solved with recursive algorithm

(c) $\log n$ is $o(n^{.01})$

(d) $\log n$ is $\omega(\log \log n)$

2. **Induction**

   Consider the recurrence $f(n) = f(n-1) * f(n-1)$, $f(1) = 2$. Show using induction that $f(n)$ is $\Omega(2^n)$

3. **Making Change**

Imagine you live in a country where the coin denominations are 1 cent, 4 cents, and 5 cents. Consider the problem where you are given some value n and you want to make change for this value, using the smallest number of coins.

Part 1: Show that the greedy algorithm (use the largest value coins first) for making change fails for these denominations

Part 2: Describe a dynamic programming algorithm for finding the smallest number of coins needed in this country to make change for any value of n. Analyze your algorithm.

4. **Amortized Analysis**

Assume you are creating an array data structure that has a fixed size of n. You want to backup this array after every so many insertion operations. Unfortunately, the backup operation is quite expensive, it takes n time to do the backup. Insertions without a backup just take 1 time unit.

Part 1: How frequently can you do a backup and still guarantee that the amortized cost of insertion is $O(1)$?

Part 2: Prove that you can do backups in $O(1)$ amortized time. Use the *potential method* for your proof.

5. **Recurrences**

In this problem, you will use recurrence relations to analyze an interesting magic trick. The trick is done as follows. Choose any two integers and write them one after another. Now form a third number by subtracting the first number from 5/2 times the second number. Form a fourth number by subtracting the second from 5/2 times the third, etc. until you have a sequence of twenty numbers. Now divide the twentieth number by the nineteenth. The value you get should be very close to 2. Can you explain why this trick works? Hint: Write down a recurrence relation $f(n)$ for the n-th number in the sequence. Now get the general solution to this recurrence relation using annihilators. Next, figure out what is a good approximation to this solution for large n.