University of New Mexico
Department of Computer Science

Midterm Examination

CS 362 Data Structures and Algorithms
Spring, 2005

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**

   Multiple Choice:

   The following choices will be used for the multiple choice problems.

   (a) $\Theta(1)$
   (b) $\Theta(\log^* n)$
   (c) $\Theta(\log n)$
   (d) $\Theta(\sqrt{n})$
   (e) $\Theta(n)$
   (f) $\Theta(n \log n)$
   (g) $\Theta(n^2)$
   (h) $\Theta(n^3)$
   (i) $\Theta(2^n)$

   For each of the questions below, choose one of the above possible answers. Please write the letter of your chosen answer to the left of the question.

   (a) $2^{(1/2)\log n}$ *Solution:* $\Theta(\sqrt{n})$
   (b) Worst case cost of a single call to Find-Set using the union-find data structure over n elements *Solution:* $\Theta(\log n)$ *or* $\Theta(n)$ *are both acceptable. It's* $\Theta(\log n)$ *if doing union by rank and* $\Theta(n)$ *otherwise.*
   (c) Amortized cost of an insertion of an element into a dynamic table *Solution:* $\Theta(1)$
   (d) Solution to the recurrence $T(n) = 4T(n/2) + 1$ *Solution:* $\Theta(n^2)$
   (e) Solution to the recurrent $T(n) = 2T(n/2) + 1$ *Solution:* $\Theta(n)$

   True or False: Justify your answer briefly (10 points total). **Circle your final answers.**

   (a) If an operation takes $O(1)$ worst case time, then it takes $O(1)$ amortized time. *Solution: True*
   (b) Greedy algorithms do not always find the correct solutions *Solution: True*
   (c) $\log n$ is $o(\log^2 n)$ *Solution: True*
   (d) $\log n^2$ is $\Omega(\log^2 n)$ *Solution: False*
   (e) A dynamic programming algorithm always uses some type of recurrence relation. *Solution: True*

2. **Annihilators**

Consider the following function:

```
int f (int n){
  if (n==0) return 0;
  else{
    val = f (n-1);
    val += n;
    return val;
  }
}
```

(a) Let $f(n)$ be the value returned by the function f when given input n. Write a recurrence relation for $f(n)$

Solution: $f(n) = f(n-1) + n$

(b) Now give the general form for the solution for $f(n)$ using annihilators. *You need not solve for the constants.*

Solution: First we annihilate the homogeneous part, $f(n) = f(n-1)$*. This is annihilated by* $\boldsymbol{L} - 1$*. Now we must annihilate the nonhomogeneous part* $f(n) = n$*. It's not hard to see that* $(\boldsymbol{L}-1)^2$ *annihilates this nonhomogeneous part. So the annihilator for the entire function is* $(\boldsymbol{L}-1)^3$*. Looking this up in the lookup table, we see that* $f(n)$ *is of the form:*

$$f(n) \quad = \quad c_1 n^2 + c_2 n + c_3 \qquad (1)$$

3. **Dynamic Programming**

Consider a new variant of the string alignment problem defined as follows. *1) Each column containing a blank costs two; 2) a column containing two characters that are different match costs one and a column containing two characters that are the same costs zero; and 3) we want to find an alignment of* minimal *cost.* For example assume we are trying to align the two strings “ALGORITHM” and “AGORITHM”. The alignment below would have cost 2 and would be an optimal alignment since it minimizes the cost.

A	L	G	O	R	I	T	H	M
A		G	O	R	I	T	H	M

(a) The recurrence relation for the optimal cost of aligning two strings A and B in the original variant of the string alignment problem is given in the formula below. $E(i, j)$ is the optimal value of aligning $A[0..i]$ and $B[0..j]$. Give the modifications needed to get a recurrence relation for the optimal cost in the new variant of the problem. *To do this, you will need to make several small changes to the formula below.* Please cross out the values (or words) to change and write the new values next to the crossed out ones.

$$
\begin{aligned}
E(0, j) &= j \text{ for all } j, \\
E(i, 0) &= i \text{ for all } i \\
E(i, j) &= \min \left\{ \begin{array}{l} E(i-1, j) + 1, \\ E(i, j-1) + 1, \\ E(i-1, j-1) + \left\{ \begin{array}{l} 0 \text{ if } A[i] = B[j] \\ 1 \text{ if } A[i] \neq B[j] \end{array} \right\} \end{array} \right\}
\end{aligned}
$$

(b) Now use this new recurrence to find the optimal alignment cost under this new variant for the two strings *ba* and *cb*. Do this by filling in the nine entries in the following dynamic programming table. Also include the arrows used to reconstruct an optimal solution. To the right of the table, give an alignment which achieves the optimal cost.

		b	a
c			
b			

Solution:

$$
\begin{aligned}
E(0, j) &= 2 * j \textit{ for all } j, \\
E(i, 0) &= 2 * i \textit{ for all } i \\
E(i, j) &= \min \left\{ \begin{array}{l} E(i-1, j) + 2, \\ E(i, j-1) + 2, \\ E(i-1, j-1) + \left\{ \begin{array}{l} 0 \textit{ if } A[i] = B[j] \\ 1 \textit{ if } A[i] \neq B[j] \end{array} \right\} \end{array} \right\}
\end{aligned}
$$

$$\begin{tikzcd}
 & & b & a \\
 & 0 \arrow[r] \arrow[d] \arrow[dr] & 2 \arrow[r] \arrow[dr] & 4 \\
c & 2 \arrow[d] \arrow[dr] & 1 \arrow[r] \arrow[dr] & 3 \\
b & 4 & 2 & 2
\end{tikzcd}$$

Alignment: $\begin{array}{cc} b & a \\ c & b \end{array}$

4. **Amortized Analysis**

Consider a linked list that has the following operations defined onit:

- *AddLast(x)*: Adds the element x to the end of the list
- *RemoveOdds()*: Removes every element at a location which is an odd number in the list. I.e. removes the first, third, fifth, etc., elements of the list.

Assume these operations have the following costs:

- *AddLast(x)* - cost equals 1
- *RemoveOdds()* - cost equals the number of elements in the list

(a) Assume we perform n operations on the stack. What is the worst case run time of a call to RemoveOdds? Justify your answer.

Solution: Worst case is $O(n)$ which happens when we call AddLast() $n-1$ times and then call RemoveOdds()

(b) *Accounting Method.* Now you will show that the amortized cost of these two operations is small using the taxation (accounting) method.

i. First give the amount that you will charge AddLast() and the amount that you will charge RemoveOdds().

ii. Next show how you will use these charges to pay for the actual costs of these operations.

iii. Finally write down the amortized cost per operation.

Solution: AddLast gets charged 3 dollars. RemoveOdds gets charged 0 dollars. When we call AddLast, we spend 1 dollar immediately to pay for the cost of the call, we then store the remaining two dollars with the item added to the list. When we call RemoveOdds(), every element in the list has two dollars stored with it. We take one dollar from each item to pay the cost of the call to RemoveOdds(). Further, for every item deleted from the list, we take the extra dollar stored at that item and place it on the next item in the list. Thus, at the end of the call to RemoveOdds() all elements in the list still have two dollars stored on them. The amortized cost per operation is thus $O(1)$

(c) *Potential Method.* You will next use the potential method to get the amortized cost per operation. Let L_i be the list after the i-th operation and let $num(S_i)$ be the number of elements in L_i. You will use the following potential function:

$$\phi_i = 2 * num(L_i)$$

i. First show that this potential function is valid

ii. Next use this potential function to calculate the amortized costs of AddLast and RemoveOdds (Recall that $a_i = c_i + \phi_i - \phi_{i-1}$ where a_i is the amortized cost of the i-th operation and c_i is the actual cost)

Solution: The number of items on the list is initally 0 *and is always nonnegative so* ϕ *is valid. First we calculate the amortized cost of AddLast() at time* i*. Note that* $c_i = 1$ *and* $\phi_i - \phi_{i-1} = 2$*. Thus* $a_i = 3$*. Next we calculate the amortized cost of RemoveOdds(). Note that* $c_i = num(S_{i-1})$*. Further note that* $\phi_i - \phi_{i-1} = -num(S_{i-1})$*. Thus* $a_i = 0$*. This implies that the amortized cost of both operations is* $O(1)$*.*

5. **Recurrences**

   Let T be a 3-ary tree (i.e. branching factor 3), with a root r and subtrees $T1$, $T2$ and $T3$ directly below r. We say that T is a *height-balanced* 3-ary tree if:

   - The heights of $T1$, $T2$, and $T3$ differ by at most one. (i.e. the max height of $T1$, $T2$ and $T3$ is at most one more than the min height of $T1$, $T2$ and $T3$)
   - $T1$,$T2$ and $T3$ are themselves all height balanced 3-ary trees

   Let $T(n)$ be the smallest number of nodes needed to obtain a height-balanced 3-ary tree of height n

   (a) Write the recurrence for $T(n)$

   Solution: To get a height-balanced tree of height n with the smallest number of nodes, need one subtree of height $n-1$ and the other two subtrees can be of height $n-2$. Thus $T(n) = T(n-1) + 2T(n-2) + 1$.

   (b) Now solve this recurrence. You need only give the general form for the solution; you do not need to solve for the constants.

   Solution: $\boldsymbol{L}^2 - \boldsymbol{L} - 2$ *annihilates the homogeneous terms and* $\boldsymbol{L} - 1$ *annihilates the non-homogeneous terms. Factoring, we get that the whole annihilator is* $(\boldsymbol{L}-2)(\boldsymbol{L}+1)(\boldsymbol{L}-1)$. *The Lookup table tells us that the general solution to the recurrence is then* $T(n) = c_1 2^n + c_2(-1)^n + c_3$. *This was not required but if you want to solve for the constants, note that* $T(0) = 1$, $T(1) = 2$ *and* $T(2) = 5$. *Solving* 3 *equations and 3 unknowns gives that* $T(n) = (4/3)2^n + (1/6)(-1)^n - (1/2)$. *Despite all the fractions, this formula always gives an integer. Try it out for higher values of n. Cool, huh?*