University of New Mexico
Department of Computer Science

Midterm Examination

CS 361 Data Structures and Algorithms
Spring, 2004

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 in this multiple choice problem.

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

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) Runtime of (deterministic) quicksort on an input array that is already sorted *Solution:* $\Theta(n^2)$

(b) Worst case runtime of heapsort *Solution:* $\Theta(n \log n)$

(c) Amount of extra space required by heapsort (not counting the space to store the array to be sorted) *Solution:* $\Theta(1)$

(d) $\sum_{i=0}^{n} \frac{n}{4^i}$ *Solution:* $\Theta(n)$ - $\sum_{i=0}^{n} \frac{n}{4^i} = n \sum_{i=0}^{n} \frac{1}{4^i} = \Theta(n)$*; we know that* $\sum_{i=0}^{n} \frac{1}{4^i}$ *is* $\Theta(1)$ *since it is a geometric series*

(e) Expected value of the *sum* of n roles of a six-sided die *Solution:* $\Theta(n)$

(f) Expected number of items falling in the first bucket during a run of bucketsort *Solution:* $\Theta(1)$

(g) Number of nodes in a heap of height n *Solution:* $\Theta(2^n)$

(h) Solution to the recurrence $T(1) = 1, T(n) = 2T(n/2) + n$ *Solution:* $\Theta(n \log n)$

(i) Solution to the recurrence $T(1) = 1, T(n) = 8T(n/2) + n$ *Solution:* $\Theta(n^3)$

(j) Solution to the recurrence $T(1) = 1, T(n) = T(n/2) + n$ *Solution:* $\Theta(n)$

2. **Recurrence Relations**

Consider the following function:

```
int f (int n){
  if (n==0) return 0;
  else if (n==1) return 1;
  else{
    int val = 4*f (n-1);
    val = val - 4*f (n-2);
    val += 1;
    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: $f(n) = 4f(n-1) - 4f(n-2) + 1$

(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,* $f(n) = 4f(n-1) - 4f(n-2)$. *Let* $F_n = f(n)$, *and* $F = \langle F_n \rangle$. *Then*

$$\begin{aligned} F &= \langle F_n \rangle \\ \boldsymbol{L}F &= \langle F_{n+1} \rangle \\ \boldsymbol{L}^2 F &= \langle F_{n+2} \rangle \end{aligned}$$

Since $\langle F_{n+2} \rangle = \langle 4F_{n+1} - 4F_n \rangle$, *we know that* $\boldsymbol{L}^2 F - 4\boldsymbol{L}F + 4F = \langle 0 \rangle$, *and thus* $\boldsymbol{L}^2 - 4\boldsymbol{L} + 4 = (\boldsymbol{L} - 2)(\boldsymbol{L} - 2)$ *annihilates* F.

Now we must annihilate the nonhomogeneous part $f(n) = 1$. *It's not hard to see that* $\boldsymbol{L} - 1$ *annihilates this nonhomogeneous part. So the annihilator for the entire function* $f(n) = 4f(n-1) - 4f(n-2) + 1$ *is* $(\boldsymbol{L} - 2)^2(\boldsymbol{L} - 1)$. *Looking this up in the lookup table, we see that* $f(n)$ *is of the form:*

$$\begin{aligned} f(n) &= (c_1 n + c_2)2^n + c_3 \\ f(n) &= O(n2^n) \end{aligned}$$

In fact, if you solve for the constants, you'll see that $T(n) = n2^{n-1}$

2. **Recurrence Relations, continued.**

3. **Asymptotic Notation**

Prove that $n = \Omega(\sqrt{n}\log n)$.

Solution: Goal: Give positive constants c *and* n_0 *such that* $c\sqrt{n}\log n \le n$ *for all* $n \ge n_0$. *The inequality we want then is:*

$$\begin{aligned} c\sqrt{n}\log n &\le n \\ c\log n &\le \sqrt{n} \\ c &\le \frac{\sqrt{n}}{\log n} \end{aligned}$$

The right hand side of this inequality is increasing as n *grows large. Thus if we choose* $n_0 = 2$ *and* $c = 1$, *it satisfies the inequality for all* $n \ge n_0$. *In other words, for* $c = 1$ *and* $n_0 = 2$, *it's the case that* $c\sqrt{n}\log n \le n$ *for all* $n \ge n_0$

4. **Substitution Method**
Consider the following recurrence:

$$T(n) = T(\lfloor n/2 \rfloor) * T(\lfloor n/2 \rfloor)$$

where $T(1) = 1$.

Show that $T(n) \leq 2^n$ by induction. Include the following in your proof: 1)the base case(s) 2)the inductive hypothesis and 3)the inductive step.
(Recall that $\lfloor n/2 \rfloor \leq n/2$)

Solution: Base Case: $T(1) = 2$ *which is in fact no more than* 2^1.
Inductive Hypothesis: For all $1 \leq j < n$, $T(j) \leq 2^j$
Inductive Step: We must show that $T(n) \leq 2^n$, *assuming the inductive hypothesis.*

$$\begin{aligned} T(n) &= T(\lfloor n/2 \rfloor) * T(\lfloor n/2 \rfloor) && (1) \\ &\leq 2^{\lfloor n/2 \rfloor} * 2^{\lfloor n/2 \rfloor} && (2) \\ &\leq 2^{n/2} * 2^{n/2} && (3) \\ &= 2^n && (4) \end{aligned}$$

where the inductive hypothesis allows us to make the replacements in the second step.

5. **Loop Invariants**

Assume we have an array $A[1..n]$ and an index i between 1 and n, and that the binary trees rooted at $Left(i)$ and $Right(i)$ are (max) heaps but that $A[i]$ can be smaller than its children. Recall that the algorithm *Heapify(A,i)* ensures that after its call, the tree rooted at $A[i]$ is a (max) heap.

Now consider the following procedure, *Build-Heap(A)* which uses the algorithm, *Heapify*. This procedure is given some arbitrary array $A[1..n]$. It claims that after its call, the array $A[1..n]$ becomes a heap.

```
Build-Heap(A)
  for(i=n/2;i>=1;i--){
    Heapify(A,i)
  }
}
```

Part 1: Give the loop invariant which you would use to show that *Build-Heap* is correct. *Solution: At the start of each iteration of the for loop, each node* $i+1,..,n$ *is the root of a max-heap.*

Part 2: Show the *termination* condition for your loop invariant. *Solution: At termination,* $i = 0$*. By the loop invariant, each node* $1,..,n$ *is the root of a max-heap. In particular node* 1 *is.*