Department of Computer Science

Midterm Examination

CS 361 Data Structures and Algorithms
Fall, 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**

   True of False (Circle One):

   (a) True or False: If $f(n)$ is $o(g(n))$ then it's always the case that $f(n)$ is $O(g(n))$.*Solution: True*

   (b) True or False: If $f(n)$ is $O(g(n))$ then it's always the case that $f(n)$ is $o(g(n))$.*Solution: False consider $f(n) = n$ and $g(n) = n$*

   (c) True or False: Heapsort and Quicksort have the same asymptotic space costs *Solution: True: they both require $\Theta(1)$ space in addition to the input array*

   (d) True or False: In a max-heap, the smallest element will always be located in the rightmost leaf node *Solution: False: It will be in a leaf node but not necessarily the rightmost*

   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) Height of a heap containing n nodes *Solution:* $\Theta(\log n)$

   (b) Expected number of heads you will see if you flip a fair coin n times *Solution:* $\Theta(n)$

   (c) $\sum_{i=0}^{n} 2^i$ *Solution:* $\Theta(2^n)$ - $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1 = 2 * 2^n - 1 = \Theta(2^n)$

   (d) Solution to the recurrence $T(1) = 1, T(n) = 4T(n/4) + n$ *Solution:* $\Theta(n \log n)$

   (e) Solution to the recurrence $T(1) = 1, T(n) = 4T(n/2) + n$ *Solution:* $\Theta(n^2)$

   (f) On a planet far,far away, that there are n days in a year and n people in a room. What is the expected number of pairs of people that have the same birthday? *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 - 3*f (n-2);
    val += 2;
    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: $f(n) = 4f(n-1) - 3f(n-2) + 2$

(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, $f(n) = 4f(n-1) - 3f(n-2)$*. 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 $\langle F_{n+2} \rangle = \langle 4F_{n+1} - 3F_n \rangle$*, we know that* $\boldsymbol{L}^2 F - 4\boldsymbol{L}F + 3F = \langle 0 \rangle$*, and thus* $\boldsymbol{L}^2 - 4\boldsymbol{L} + 3 = (\boldsymbol{L} - 3)(\boldsymbol{L} - 1)$ *annihilates* F*.*

Now we must annihilate the nonhomogeneous part $f(n) = 2$*. It's not hard to see that* $\boldsymbol{L} - 1$ *annihilates this nonhomogeneous part. So the annihilator for the entire function* $f(n) = 4f(n-1) - 3f(n-2) + 2$ *is* $(\boldsymbol{L} - 3)(\boldsymbol{L} - 1)^2$*. Looking this up in the lookup table, we see that* $f(n)$ *is of the form:*

$$\begin{aligned} f(n) &= c_0 3^n + (c_1 n + c_2) \\ f(n) &= O(3^n) \end{aligned}$$

2. **Recurrence Relations, continued.**

3. **Asymptotic Notation**

Prove that $(1/2)\log^2 n = \Omega(\log n)$. First write down your goal (i.e. what you need to show to prove a Ω relationship in this case). Then prove that goal.

Solution: Goal: Give positive constants c and n_0 such that $c \log n \leq (1/2)\log^2 n$ for all $n \geq n_0$. The inequality we want then is:

$$\begin{aligned} c \log n &\leq (1/2)\log^2 n \\ c &\leq (1/2)\log n \end{aligned}$$

Where we go from the first step in the above to the second by dividing by $\log n$. The right hand side of the final inequality is increasing as n grows large. Thus if we choose $n_0 = 2$ and $c = 1/2$, it satisfies the inequality for all $n \geq n_0$. In other words, for $c = 1/2$ and $n_0 = 2$, it's the case that $c \log n \leq (1/2)\log^2 n$ for all $n \geq n_0$

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

$$T(n) = T(\lfloor\sqrt{n}\rfloor) * T(\lfloor\sqrt{n}\rfloor)$$

where $T(1) = 1$.

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

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

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

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

5. **Heap 'o Heap 'o Burning Sort**

Professor Pelvis has developed a new sorting algorithm called *Hobsort* (short for Heap 'o Burning Sort). Hobsort first creates two min-heaps, one out of the left half of the input array and one out of the right half of the input array. It then repeatedly compares the root nodes of these two heaps, deletes the smaller root node and appends that value to the output array. Pseudocode for the algorithm is given below. Note that the function *Build-Heap* is the algorithm we went over in class for building a heap out of an arbitrary array. The functions *Minimum* and *Extract-Min* are the same functions we went over in class defined for min-heaps. Assume that the function *Minimum* returns ∞ when called on an empty heap.

```
Hobsort(A)
  \\At the end of this algorithm, Res[1..n] contains the numbers in
  \\the array A[1..n] in sorted order
  H1 = Build-Heap(A[1..n/2])
  H2 = Build-Heap(A[n/2+1..n])
  for(i=1;i<=n;i++){
    if (Minimum(H1)<= Minimum(H2))
       m = Extract-Min(H1);
    else
       m = Extract-Min(H2);
    Res[i] = m;
  }
  return Res[1..n];
}
```

Part 1: Give the asymptotic run time of Hobsort. Show your work.

Solution: The two calls to Build-Heap take $\Theta(n)$ time. The calls to Minimum take $\Theta(1)$ time and Extract-Min takes $\Theta(\log n)$ time. Thus the loop body takes $\Theta(\log n)$ time total. The loop runs n times so the total time taken by the loop is $\Theta(n \log n)$. Thus the total run time is $\Theta(n \log n)$

Part 2: Give the loop invariant that you would use to show that Hobsort is correct.*Solution: After i iterations of the for loop, the array Res[1..i] contains the i smallest elements of A in sorted order; moreover, the $n - i$ largest elements of A are contained in the heaps $H1$ and $H2$.*