Final Examination
CS 361 Data Structures and Algorithms
Spring, 2003

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 a closed book exam. You are permitted to use only two pages of “cheat sheets” that you have brought to the exam. Nothing else is permitted.

- Do all six 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.

- Good Luck!!!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>120</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. True/False and Theta Notation (20 points)

True or False: (circle one, 2 points each)

(a) **True or False**: Any sorting algorithm takes $\Omega(n \log n)$ time in the worst case? *Solution: False. Only comparison based sorting algorithms*

(b) **True or False**: Randomized Quicksort always takes $O(n \log n)$ time? *Solution: False. $O(n^2)$ time*

(c) **True or False**: Bucket sort takes $\Theta(n)$ time in the best case? *Solution: True*

(d) **True or False**: Mergesort takes $\Theta(n)$ time in the best case? *Solution: False. Best and worst case for Mergesort are $\Theta(n \log n)$*

(e) **True or False**: An array that is in sorted order (i.e., non-decreasing) is a min-heap? *Solution: True: it satisfies the heap property*

Theta Notation: (2 points each)

For each function below, give a $\Theta()$ expression that is as simplified as possible. Justify your answers briefly.

(a) $n^3 \log n - n\sqrt{n} + 100 \log^{10} n$ *Solution: $\Theta(n^3 \log n)$*

(b) $\log^2 n + 10 \log n^{100}$ *Solution: $\Theta(\log^2 n)$, since $10 \log n^{100} = 1000 \log n$ which is asymptotically smaller than $\log^2 n$*

(c) $\sqrt{n} + \log^2 n$ *Solution: $\Theta(\sqrt{n})$ since $\sqrt{n}$ is asymptotically larger than $\log^2 n$*

(d) $n * (\sum_{i=1}^{n} 1/i)$ *Solution: $\Theta(n \log n)$*

(e) $9^{\log_3 n}$ *Solution: $9^{\log_3 n} = 3^{2 \log_3 n} = n^2 = \Theta(n^2)$*
2. Short Answer (20 points total, 5 points each)

(a) Heaps are a type of tree with a specific order property. Define the order property for min-heaps.

*Solution:* All descendants of any node \( r \) must be greater than or equal to \( r \) itself

(b) Binary search tree are a type of tree with a specific order property. Define the order property for binary search trees.

*Solution:* For any node \( r \), all descendants to the left of \( r \) must be \( \leq r \) and all descendants to the right of \( r \) must be greater than \( r \).

(c) Consider a full binary tree of height \( h \), where every internal node has two children and all leaf nodes have the same depth. Question: What is the ratio of the number of leaf nodes to the total number of nodes in such a tree as \( h \) grows large? Hint: First compute the number of leaf nodes, then compute the number of nodes total, then compute the ratio. *Solution:* The number of leaf nodes is \( 2^h \). The number of nodes total is \( \sum_{i=0}^{h} 2^i = 2^{h+1} - 1 \). The ratio as \( h \) gets large is \( 1/2 \)
(d) Consider a hash table with \( m \) cells. Imagine that we insert \( n \) items into the table, in such a way that each item is hashed uniformly at random to one of the \( m \) cells. Question: What is the expected number of items that are hashed to the first cell? Justify your answer (hint: use linearity of expectation).

Solution: For \( i = 1, \ldots, n \) let \( X_i \) be a random variable that is 1 if the \( i \)-th item is hashed to the first cell and 0 otherwise. Note that \( E(X_i) = \frac{1}{m} \) for any \( i \). Let \( X \) be the total number of items hashed to the first cell. Note that \( X = \sum_{i=1}^{n} X_i \). So \( E(X) = E(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} E(X_i) = n/m \)
3. **Recurrences (20 points)**

Consider the recurrence: \( T(n) = 8T(n/2) + n^2 \) (and \( T(k) = \Theta(1) \) for \( k \) a constant)

(a) Use the recurrence tree method to get a “guess” (i.e. simplest possible big-O) on the solution to this recurrence. You need not prove your guess correct.

(b) Now use annihilators (and change of variable) to get a tight upperbound (i.e. simplest possible big-O) on the solution to this recurrence.

(c) Now use the Master Theorem to solve the recurrence (all three bounds should match)

**Solution:**

**Recurrence Tree:** \( T(n) = 8T(n/2) + n^2 \), \( T(n/2) = 8T(n/4) + (n/2)^2 \), \( T(n/4) = 8T(n/8) + (n/4)^2 \). Writing this out in a recurrence tree, we get that the zero level is one \( n^2 \), the first level is eight \( n^2 = 4 \)’s, the second level is 64 \( n^2 = 16 \)’s. In general, the \( i \)-th level sums to \( (8/4)^i n^2 = 2^i n^2 \). There are \( \log_2 n \) levels, so the sum of all of them is:

\[
n^2 \sum_{i=0}^{\log_2 n-1} (2)^i = n^2 \left( \frac{1 - 2^{\log n}}{1 - 2} \right) = \Theta(n^3) \tag{1}
\]

**Annihilators:** Let \( n = 2^i \) and \( t(i) = T(2^i) \). Then

\[
t(i) = 8t(i-1) + 2^{2i} \tag{3}
\]

\[
t(i) = 8t(i-1) + 4^i \tag{4}
\]

The annihilator for this is \((L - 8)(L - 4)\), and thus from the lookup table, the form of the recurrence is:

\[
t(i) = c_1 8^i + c_2 4^i \tag{5}
\]

\[
t(i) = c_1 (2^i)^3 + c_2 (2^i)^2 \tag{6}
\]

The reverse transformation gives that

\[
T(n) = c_1 n^3 + c_2 n^2
\]

This is \( \Theta(n^3) \)

**Master Theorem:** \( T(n) = 8T(n/2) + n^2 \) is of the form \( T(n) = aT(n/b) + f(n) \) where \( a = 8, b = 2 \) and \( f(n) = n^2 \). Note that \( af(n/b) = 8(n/2)^2 = 2n^2 \), and this is larger than \( f(n) \) by a constant factor. Thus in the recurrence tree, the leaf nodes dominate, and so the solution is of the form \( T(n) = \Theta(n^{\log_2 b}) = \Theta(n^3) \)
3. Recurrences (20 points), continued.
4. Annihilators

Consider the recurrence \( T(n) = 2T(n - 1) - T(n - 2) + 4, \ T(0) = 0, \ T(1) = 0 \). Solve this recurrence exactly using annihilators. Don’t forget to check your answer.

**Solution:** Consider the homogeneous part first. Let \( T_n = 2T(n - 1) - T(n - 2), \ and \ T = \langle T_n \rangle \).

Then

\[
\begin{align*}
T &= \langle T_n \rangle \\
LT &= \langle T_{n+1} \rangle \\
L^2T &= \langle T_{n+2} \rangle
\end{align*}
\]

Since \( \langle T_{n+2} \rangle = \langle 2T_{n+1} - T_n \rangle \), we know that \( L^2T - 2LT + T = \langle 0 \rangle \), and thus \( L^2 - 2L + 1 = (L - 1)(L - 1) \) annihilates \( T \). Further we know that \( (L - 1) \) annihilates the non-homogeneous part. Thus the annihilator of the whole sequence is \( (L - 1)^3 \). Thus \( T(n) \) is of the form:

\[
T(n) = c_1n^2 + c_2n + c_3
\]

We know:

\[
\begin{align*}
T(0) &= 0 \quad c_3 \\
T(1) &= 0 \quad c_1 + c_2 \\
T(2) &= 4 \quad 4c_1 + 2c_2
\end{align*}
\]

so \( c_1 = 2, \ c_2 = -2, \ c_3 = 0 \) and thus

\[
T(n) = 2n^2 - 2n
\]

Check: \( T(3) = 2 \cdot 4 - 0 + 4 = 12 \) and \( 2 \cdot 9 - 6 = 12 \).
5. Recursion and Recurrences (20 points)

Consider the following recursive sorting algorithm which takes a list $l$ of numbers:

```java
Zanysort(l) {
  if (l.size() <= 1) {
    return l;
  } else {
    Zanysort the first third of l;
    Heapsort the remaining two thirds of l;
    Merge the two sorted lists together;
  }
}
```

(a) Let $T(n)$ be the run time of Zanysort. Write down a recurrence relation for $T(n)$ (hint: Use $\Theta$ notation in the recurrence relation).

**Solution:**

$T(n) = T(n/3) + \Theta(n \log n)$

(b) Now solve this recurrence relation in terms of tight big-O. Hint: Use the Master Theorem.

**Solution:** $T(n) \leq T(n/3) + k(n \log n)$ for some constant $k$. If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 1$, $b = 3$, $f(n) = kn \log n$. Then $af(n/b) = k(n/3 \log(n/3))$ which is a constant factor smaller than $f(n)$. Hence the root node dominates the recursion tree and so the solution is $T(n) = \Theta(n \log n)$. So surprisingly, this silly sorting algorithm is as good as the best of them.
6. **Loop Invariants (20 points)**

In this question, you will be proving the correctness of the procedure `Tree-Search` using loop invariants. This procedure takes as input a key $k$, and the root, $r$, of a binary search tree. If the key $k$ exists in the tree rooted at $r$, the procedure returns the node with key $k$. Otherwise, the procedure returns nil. The procedure is given below:

```c
Tree-Search(r, k) {
    while (r != nil && k != key(r)) {
        if (k <= key(r)) {
            r = left(r);
        } else {
            r = right(r);
        }
    }
    return r;
}
```

(a) State a loop invariant for the while loop of `Tree-Search`.

*Solution: If the key $k$ is in the original tree then the key $k$ is in the subtree rooted at $r$.*

(b) Establish initialization, maintenance and termination for your loop invariant.

*Solution: Initialization: Before the first iteration of the while loop, the invariant is obviously true since the subtree rooted at $r$ is the entire tree.

Maintenance: Let $r'$ be the value of $r$ at the beginning of some fixed iteration of the while loop. Note that we know by induction that if the key $k$ is in the original tree, it is in the subtree rooted at $r'$. Now if we execute the while loop, it must be the case that $key(r')$ is not equal to $k$. Thus if $k$ is in the subtree rooted at $r'$, it must be in either the left or right subtree. By the binary search tree property, $k$ must be in the left subtree if $k \leq key(r')$ and in the right subtree otherwise. Thus the body of the while loop sets $r$ to the correct value, and the invariant is maintained.

Termination: Assume the invariant holds right after exit of the while loop. Note that we only exit the while loop if $r$ is nil or $key(r)$ is $k$. Thus, if the key is in the original tree, $r$ can not be nil, so $key(r)$ is $k$ and so the algorithm does in fact find the key $k$. 
6. Loop Invariants (20 points), continued.