15-211: Midterm
March 4, 2003

Instructions

- Fill in the box above with your name, your Andrew ID, and your section. **Do it, now!**
- This exam is closed book, but you are allowed to use one page of notes.
- Clearly mark your answers in the allocated space. If need be, use the back of a page for scratch space. If you have made a mess, cross out the invalid parts of your solution, and circle the ones that should be graded.
- Scan the test first to make sure that none of the 10 pages are missing. The problems are of varying difficulty, you might wish to pick off the easy ones first.
- You have 80 minutes. Good luck.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>125</strong></td>
</tr>
</tbody>
</table>
Problem 1: Asymptotics (15 pts.)

You are familiar with the notions $O$ and $\Omega$ expressing upper and lower bounds, respectively. On occasion it is useful to combine both to get an “exact” bound, up to to constants. To this end one defines

$$f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)).$$

In other words, $f(n) = \Theta(g(n))$ if there are constants $c_1, c_2$ and $n_0$ such that for all $n \geq n_0$:

$$c_1 g(n) \leq f(n) \leq c_2 g(n).$$

A. Answer “true” or “false” to each of the following assertions.

$$15n^3 + 10^{10} n^2 + 17n \log n = \Theta(n^2)$$
$$15n^3 + 10^{10} n^2 + 17n \log n = \Theta(n^3)$$
$$15n^3 + 10^{10} n^2 + 17n \log n = \Theta(n^4)$$
$$10^n = \Theta(2^n)$$
$$n \log n = \Theta(n \sqrt{n})$$

B. You are given the following two pieces of information about a function $f$: $f(n) = \Omega(n \log n)$ and $f(n) = O(n^2)$.

Which one of the following assertions cannot be true? Explain your answer in one (and only one) sentence.

i. $f(n) = \Theta(n \log n)$
ii. $f(n) = \Theta(n \sqrt{n})$
iii. $f(n) = \Theta(n^3)$
Problem 2: AVL Trees (15 pts.)

Construct an AVL tree by inserting the following elements: 50, 30, 20, 60, 80, 40, in this order. You must rotate the tree as soon as the AVL property is violated. Show the tree after each insertion and clearly label the nodes by the elements stored there.

insert 50

insert 30

insert 20
Problem 3: Duplicates (20 pts.)

Given an array $A$ of integers, we would like to determine whether $A$ has any duplicate elements.

A. Give an efficient algorithm to solve this problem using only the equal to comparison. That is, you may only compare two elements in $A$ with the comparison $A[i] == A[j]$. You cannot use any other data structures. The running time of your algorithm should be quadratic in the length of $A$.

```java
public static boolean hasDuplicates( int[] A )
{
}
```

B. Give a single line explanation of why the running time of your algorithm is optimal.

C. Suppose now that you can also use the less than comparison. Is it possible to solve the duplicates problem faster than your algorithm in part 1? If yes, explain the algorithm in one brief sentence and state it’s running time. If no, briefly explain why not.
Problem 4: A Recurrence (20 pts.)

Solve the recurrence

\[
\begin{align*}
T(1) &= 1 \\
T(n) &= n^2 - T(n - 1)
\end{align*}
\]

Hint. There are two ways to tackle this problem: compute the first few terms by hand until you see a pattern, or unroll the recurrence, again till a pattern appears.

Another Hint. Be very careful with the +/- signs!

Either way, make sure to prove the correctness of your claim.

Claim: \( T(n) = \)
Problem 5: Picking a Data Structure (15 pts.)

As you know, different applications impose different requirements on the running time of the methods exported by a data structure. Each problem A, B, C, and D below give examples of such requirements. For each one, list which data structure can meet the requirements. They are either data structures that you have learned about, or that can be easily constructed using such data structures.

Assume you want to build a dictionary to store (key,value) pairs much like the hashtables in Java do. Specifically, you would like to implement a data structure that exports these methods:

- **insert(key,value)** - Add a pair to the dictionary.
- **delete(key)** - Remove the key.
- **lookup(key)** - Determine if the key is in the dictionary.
- **print()** - Enumerate all elements in sorted order by keys of the dictionary.
- **findMin()** - Return the minimum element.
- **removeMin()** - Take the minimum element out.

If a function is not listed, that means its time requirement is unbounded. Be clear in your description of each data structure you describe.

A. **findMin(): O(1)**    **removeMin(): O(log n)**    **insert(): O(log n)**
   **print(): O(n log n)**

B. **insert(): O(log n)**    **lookup(): O(log n)**    **delete(): O(log n)**

C. **insert(): O(1)**    **lookup(): O(1)**    **delete(): O(1)**

D. **insert(): O(1)**    **lookup(): O(1)**    **findMin(): O(1)**
Problem 6: Heabsts (20 pts.)

Given an ordered sequence $a_1 < a_2 < \ldots < a_n$ there are many different binary search trees that produce this sequence when flattened out in an inorder-traversal. Now suppose we introduce an additional requirement on the BST: it has to satisfy the heap order property, producing a heabst. More precisely, we assign a priority $p_i$ to each node $a_i$, and insist that the tree

- is a binary search tree with respect to the values $a_i$, and
- has the heap order property with respect to the priorities $p_i$.

Note that a heabst is not required to have the shape of a heap, though.
For simplicity, assume that all the priorities are distinct.

A. Draw all possible binary search trees for elements $a < b < c$.

B. Which heabst results if we assign the priorities 1, 2, 3 to $a, b, c$, respectively?

Which heabst do we get for priorities 2, 1, 3?
C. Prove that regardless of the choice of priorities it is always possible to form a heap.

[Incidentally, this is not a frivolous problem: this idea is used in real algorithms to keep BSTs from becoming unbalanced.]
Problem 7: Huffman Compression (20 pts.)

A. Fred Hacker has carefully analyzed the letter frequencies in the string

“she sells sea shells on the sea shore”

see the table below. Fill in the missing labels in the outline of the Huffman tree below.

<table>
<thead>
<tr>
<th>SPC</th>
<th>a</th>
<th>c</th>
<th>e</th>
<th>h</th>
<th>l</th>
<th>n</th>
<th>o</th>
<th>r</th>
<th>s</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

B. After implementing the Huffman algorithm, Fred runs his program on a file that contains each upper case letter A through Z at least once, but no other symbols. Fred’s program claims that all code words have length exactly 4 bits. What advice do you have to give Fred? Be polite and explain your reasoning.