Test One (Answer Key)
CS495/Ma475: Analysis of Algorithms

This is a closed book, 50 minute examination. It is worth 100 points. You may use one page of notes, on an 8.5x11 inch sheet of paper. There are 11 questions (and one extra credit question) on six (6) pages.
Assume that all constants in your closed forms are greater than zero, unless the question says to argue explicitly about your constants. Also, assume that all logarithms are to base two, unless explicitly stated otherwise.
Put your name on each page of this examination.

Question 1 (10 points) Suppose an algorithm requires $S(n) = 3S(n-1) + 8$ bytes of storage on instances of size $n$. Give a tight asymptotic bound (either big-$O$ or big-$\Theta$) on its space requirements.

Answer This is a non-homogenous linear recurrence:

$$S(n) - 3S(n-1) = 1^n(8n^0)$$

So it has characteristic polynomial

$$p(x) = (x - 3)(x - 1)$$

which has roots 3 and 1. So the closed form is

$$S(n) = c_03^n + c_11^n = c_03^n + c_1$$

Since we are assuming that all relevant constants ($c_0$ and $c_1$ in this case) are positive, we may conclude that

$$S(n) \in \Theta(3^n)$$

Question 2 (8 points) InsertionSort only requires $\Theta(n)$ comparisons on sorted inputs. But when the input is sorted in reverse order, it always needs exactly $O(n^2)$ comparisons. What is the big-$O$ and big-$\Theta$ complexity of InsertionSort? Justify your answers.

Answer Big-$O$ and big-$\Theta$ are both expressed in terms of the worst case input instances. So, the complexity of InsertionSort is both $O(n^2)$ and $\Theta(n^2)$. 
Question 3 (6 points) Prove or disprove each of the following:

\[(n + 3)^2 \in O(n^3)\] \hspace{1cm} (1)

\[2^n \in \Theta(4^n)\] \hspace{1cm} (2)

\[n \log n \in \Omega(n^2)\] \hspace{1cm} (3)

Answer

1.

\[
\lim_{n \to \infty} \frac{(n + 3)^2}{n^3} = \lim_{n \to \infty} \frac{n^2 + 6n + 9}{n^3} = 0
\]

So \((n + 3)^2 \in O(n^3)\), but \((n + 3)^2 \not\in \Omega(n^3)\).

2.

\[
\lim_{n \to \infty} \frac{2^n}{4^n} = \lim_{n \to \infty} \frac{1}{2^n} = 0
\]

So \(2^n \in O(4^n)\), but \(2^n \not\in \Omega(4^n)\) and hence \(2^n \not\in \Theta(4^n)\).

3.

\[
\lim_{n \to \infty} \frac{n \log n}{n^2} = \lim_{n \to \infty} \frac{\log n}{n} = \lim_{n \to \infty} \frac{1}{n} = 0
\]

and so \(n \log n \in O(n^2)\), but \(n \log n \not\in \Omega(n^2)\).

Alternatively, it can be shown by induction (and such a demonstration is needed for full credit) that \(\forall n > 0(n \log n < n^2)\). With this, let \(n_0 = 1\) and \(c = 1\) and apply the definition of big-O.

Question 4 (10 points) Suppose algorithm \(\text{Foo}(\cdot)\) has \(2^n\) instances of size \(n\) (for any \(n\)) and it requires \(n\) steps for \(2^n - 1\) of them, and \(2^n + n\) steps for the one remaining instance. What is the average case complexity of \(\text{Foo}(\cdot)\)?

Answer It is the average number of steps required for all instances of size \(n\), which is

\[
\frac{(n \cdot (2^n - 1)) + (2^n + n)}{2^n} = \frac{n2^n + 2^n}{2^n} = n + 1
\]

So the number of steps in the average case, assuming all input instances equally likely, is \(\Theta(n)\)—far better than the worst case behavior of \(\Theta(2^n)\).
Question 5 (10 points) What is the complexity of the following code? Be sure to show all your work, and identify your assumptions.

```java
Foo(n)
{ if (n<4)
    then return 1;
    else
      { x = n/2; y = x/2;
        return (Foo(x) * Foo(y)) + Foo(y);
      }
}
```

**Answer** Assume that all arithmetic operations are elementary, and count the number of times the comparison \( n < 4 \) is executed. Let \( T(n) \) be the number of comparisons required for \( \text{Foo}(n) \). Then

\[
T(n) = T(n/2) + 2T(n/4) + c
\]

for some constant \( c \). Assume \( n \) is a power of two and restate the recurrence in terms of \( n = 2^k \):

\[
t_k - t_{k-1} - 2t_{k-2} = c
\]

The characteristic polynomial for this is

\[
p(x) = (x^2 - x - 2)(x - 1)
\]

which has roots 1, 2, and -1. So the close form is (for some constants \( c_0, c_1, c_2 \)):

\[
t_k = c_02^k + c_11^k + c_2(-1)^k
\]

Substituting back \( n \) for \( 2^k \) to get the original recurrence gives

\[
T(n) = c_0n + c_1 + (-1)^{\log n}
\]

Since we are assuming the constants are positive, we have

\[
T(n) \in \Theta(n) \text{ when } n \text{ is a power of two}
\]

But \( f(n) = n \) is smooth, since it is non-decreasing and \( f(bn) = bn \in O(n) \), so we may conclude

\[
T(n) \in \Theta(n)
\]
Question 6 (8 points) MergeSort requires $\Theta(n \log n)$ comparisons on inputs of size $n$. Can we conclude that it requires time in $O(n^2)$? In $O(n)$? Why or why not.

Answer  $n \log n \in O(n^2)$ but $n \log n \not\in O(n)$, since $\lim_{n\to\infty} \frac{(n \log n)}{n^2} = 0$ and $\lim_{n\to\infty} \frac{(n \log n)}{n} = \infty$. So, we can conclude the first, but not the second.

Question 7 (10 points) What is the complexity of the following code? Be sure to show all your work, and identify your assumptions.

```
Foo(n)
{ if (n<1)
    then return 1;
else
    { x = Foo(n-1);
      for i = 1 to n do
      { if (x < i) then x = x + 1; }
      return x + Foo(n-1);
    }
}
```

Answer  We assume all arithmetic is elementary, and count the number of times the comparison (n<1) is made. This is the recurrence

$$t_n = 2t_{n-1} + bn + c$$

for some constants $b$ and $c$. This has the characteristic polynomial

$$p(x) = (x - 2)(x - 1)^2$$

which has roots 2 (of multiplicity 1) and 1 (of multiplicity 2). So the closed form is

$$t_n = c_0 2^n + c_1 1^n + c_2 n 1^n = c_0 2^n + c_2 n + c_1$$

Since we are assuming all constants are positive, this gives us

$$t_n \in \Theta(2^n)$$
Question 8 (8 points) Give a tight asymptotic bound (big-$O$ or big-$\Theta$) bound for

$$f(n) = \sum_{i=1}^{\log n} n$$

**Answer**  This sum is equal to $n \log n$ which is in $\Theta(n \log n)$.

Question 9 (10 points) What is the complexity of the following code? Be sure to show all your work, and identify your assumptions.

```plaintext
Foo(n)
{ z = n;
  while (z > 0) do
  { x = 0;
    for i = 1 to n do {x = x + 1;}
    z = z div 2;
    for j = 1 to (n div 2) do {x = x - 1;}
  }
  return x;
}
```

**Answer**  We assume all arithmetic (even div) is elementary, and count the changes to $x$. Assume $n$ is a power of two.

By inspection, note that each iteration through the loop increments $x$ exactly $n$ times and decrements it $\lfloor n/2 \rfloor$ times. The loop occurs $\log n$ times.

Then $x$ changes the following number of times

$$\log n(n + n/2)$$

This is in

$$\Theta(n \log n)$$

But $f(n) = n \log n$ is smooth, since it is non-decreasing, and $f(bn) = bn(\log(bn)) = bn(\log b + \log n) \in O(n \log n)$. So, the time for the algorithm is in

$$\Theta(n \log n)$$
Question 10 (10 points) In a full trinary tree, each node has exactly three children. Let $S(n)$ be the number of nodes in a full trinary tree at height $n$ (the leaves are at height zero, and the root at height $n$). Express $S(n)$ as a recurrence relation and solve it exactly (that is, determine all relevant constants). Show your work.

**Answer** By definition, $S(n) = 3S(n-1) + 1$, where the first term is the nodes in each of three subtrees, and the “+1” is for the root.

This has characteristic polynomial $(x - 3)(x - 1)$ with roots 3 and 1, so the closed form is

$$S(n) = c_03^n + c_1$$

To find the constants, note that

$$S(0) = 1 = c_0 + c_1$$
$$S(1) = 4 = 3c_0 + c_1$$

Which gives $c_0 = 3/2$ and $c_1 = -1/2$, so the closed form is

$$S(n) = \frac{3^{n+1} - 1}{2}$$

Question 11 (10 points) Suppose that algorithm $\text{Foo}(n)$ uses at most $\Theta(n^2)$, but that $k$ repetitions of $\text{Foo}()$ always requires $\Theta(k)$ time. What is the worst case and amortized complexity of $\text{Foo}()$?

**Answer** Worst case is, by definition, $\Theta(n^2)$. But amortized complexity is $\Theta(k/k) = \Theta(1)$, per invocation.

Question 12 (Extra Credit) Suppose you test algorithm $\text{Foo}$ on all inputs up to size $n = 100$, and it requires time as follows:

<table>
<thead>
<tr>
<th>$n$</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>10,001</td>
</tr>
</tbody>
</table>

Can you conclude that $\text{Foo}$ is in $\Theta(n^2)$? Why or why not?
Answer  No, you cannot. Asymptotic notation is expressed in terms of behavior for sufficiently large \( n \). We have no assurance that \( n = 100 \) is large enough—these could all be special cases for the algorithm. Besides, you cannot analyze an algorithm from empirical behavior, that is analyzing a process, not an algorithm.