Test One (Answer Key)
CS395/Ma395: Analysis of Algorithms

This is a closed book, closed notes, 70 minute examination. It is worth 100 points. There are twelve (12) questions on five (5) pages.

Please put your name on each page of this examination.

Answer  Some of these answers have far more detail than was required from the test question. For example, there are questions that ask for only the recurrence, but the answer has a full solution. This is to help you use this answer key to prepare for future exams.
Question 1 [6 points] Prove or disprove each of the following:

1. \( n \log n \in \Omega(n^2) \)
2. \( 3^n \in \Theta(2^n) \)
3. \((n + 2)^2 \in O(n^4)\)

Answer

1. 
\[
\lim_{n \to \infty} \frac{n \log n}{n^2} = \lim_{n \to \infty} \frac{\log n}{n} = \lim_{n \to \infty} \frac{1}{n} = 0
\]
(by simple manipulations and L'Hôpital's rule) and so \( n \log n \in O(n^2) \), but \( n \log n \not\in \Omega(n^2) \).

2. 
\[
\lim_{n \to \infty} \frac{3^n}{2^n} = \lim_{n \to \infty} \left( \frac{3}{2} \right)^n = \infty
\]
So \( 2^n \in O(3^n) \), but \( 2^n \not\in \Omega(3^n) \) and hence \( 3^n \not\in \Theta(2^n) \).

3. 
\[
(n + 2)^2 = n^2 + 4n + 4
\]
By the max rule, this is in \( O(n^2) \), and \( O(n^2) \subset O(n^4) \) since \( \lim_{n \to \infty} \frac{n^2}{n^4} = 0 \).
So \((n + 2)^2 \in O(n^4) \), but \((n + 2)^2 \not\in \Omega(n^4) \).

Question 2 [5 points] Show that the function \( f(n) = n^2 \) is 2-smooth.

Answer We need to show that \( f(n) \) is eventually non-decreasing, and that \( f(2n) \in O(f(n)) \), where \( f(n) = n^2 \).
\[
f(n) = n^2 \leq n^2 + 2n + 1 = (n + 1)^2 = f(n + 1) \quad \text{for all } n \geq 0,
\]
so \( f \) is clearly eventually nondecreasing.
Also, \( f(2n) = (2n)^2 = 4n^2 \in O(n^2) \) since one can ignore the constant factor of 4.
Question 3 [10 points] Suppose algorithm Foo() 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 Foo()?

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 4 [10 points] Suppose that algorithm Bar(n) uses at most $\Theta(2^n)$ in the worst case, but that $k$ repetitions of Bar() always requires $\Theta(3^k)$ time. What is the worst case and amortized complexity of Foo()?

Answer Worst case is, by definition, $\Theta(2^n)$. But amortized complexity is $\Theta(3^k/k) = \Theta(1)$, per invocation.

Question 5 [5 points] Suppose the algorithm MyAlg is $O(n^2 | n$ is a power of 2). What can you say about the complexity of MyAlg, given the results of question 2?

Answer We need to show that $f(2n) \in O(f(n))$, where $f(n) = n^2$. But $f(2n) = (2n)^2 = 4n^2 \in O(n^2)$ since one can ignore the constant factor of 4.
Question 6 [10 points] Express the complexity of the following algorithm as a recurrence relation (do not solve the recurrence!)

Procedure fu(n);
{  if (n > 0)
    { 
      LoadsOfFun = 0;
      for i = 1 to n { LoadsOfFun++; }
      return (fu(n/2) + fu(n/2) + LoadsOfFun); // ‘/’ is integer division
    }
  else return 1; }

Answer Let $T(n)$ be the time for this algorithm in input $n$. Then the time required is

$$T(n) = n + 2T(n/2) + c$$

This is the time for the loop, and for the two recursive calls. It assumes that addition and multiplication are elementary operations, and that the instances sizes are the values of $n$.

Question 7 [4 points] Suppose you test algorithm Foo on all inputs up to size $n = 10$, and it requires time as follows:

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>17</td>
<td>33</td>
<td>65</td>
<td>129</td>
<td>257</td>
<td>513</td>
<td>1025</td>
</tr>
</tbody>
</table>

Can you conclude that Foo is in $\Theta(2^n)$? 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 $n = 10$ 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.
Question 8 [10 points] Analyze the worst case complexity of the following algorithm.

Procedure Foo(n);
{ for i = 1 to n do
   { for j = 1 to n do
      { for k = j to n do { cout << "Hello!"; } }
   }
}

Answer  The barometer will be the number of "Hello" messages printed, and we assume all operations are unit time.

The cout << will execute once for every selection of i, j, k, regardless of their order (since only one order, namely i < j < k is relevant). There are $O\left(\binom{n}{3}\right) = O(n^3)$ ways for this to happen.

Or, notice that the two inner loops produce $n + (n - 1) + \cdots + 2 + 1 = \sum_{i=1}^{n} = n(n+1)/2$ couts.

These two loops get executed $n$ times, so the total number of couts is $n \cdot n(n+1)/2 = (n^3 + n^2)/2 \in \Theta(n^3)$ total.

Alternatively, make a table of the possible values for the three index variables:

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>n</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
</tbody>
</table>

So the complexity is $\Theta(n^3)$.
Question 9 [10 points] What is the characteristic polynomial for the following recurrence?

\[ t_n = 6t_{n-1} - 9t_{n-2} + n2^n \]

**Answer** This yields

\[ t_n - 6t_{n-1} + 9t_{n-2} = n2^n \]

The characteristic polynomial for this recurrence is

\[ (x^2 - 6x + 9)(x - 2)^2 = (x - 3)^2(x - 2)^2 \]

Question 10 [10 points] The recurrence \( t_{n+1} - 5t_n + 6 = n2^n \) has a characteristic polynomial of \((x - 3)(x - 2)^3\)

What is the closed form for this recurrence?

**Answer**

This has root 3 with multiplicity 1 and root 2 with multiplicity 3, so the closed form solution is

\[ t_n = c_03^n + c_12^n + c_2n2^n + c_3n^22^n \]

for some constants \( c_0, c_1, c_2, \) and \( c_3 \). This is clearly \( O(3^n) \).
Question 11 [10 points] prove that the $i$th binomial tree, $B_i$, has $2^i$ nodes.

Answer The proof is by recursion on $i$.

Base case ($i = 0$). $B_0$ has 1 node by definition, and $1 = 2^0$.

Assume the $i$th binomial tree has $2^i$ nodes for $0 \leq i \leq k$, and prove that $B_{k+1}$ has $2^{k+1}$ nodes. The number of nodes in $B_{k+1}$ is one, for the root, plus one for each subtree of the root. By definition, these subtrees are $B_0$, $B_1 \ldots B_k$.

So, using the inductive assumption, $B_{k+1}$ has $1+2^0+2^1+\cdots+2^k = 1+(2^{k+1}−1) = 2^{k+1}$ nodes, which was to be shown.

Many people noticed that is is possible to define binomial trees a different way. Our definition in class was: $B_0$ is a single node, $B_{i+1}$ is a root node with each of $B_0 \ldots B_i$ as a subtree. The above proof follows that definition.

Alternatively, one can define binomial trees as follows: $B_0$ is a single node, $B_{i+1}$ is $B_i$ with a copy of $B_i$ added as a subtree to the root of the first $B_i$. Note that these are different definitions, but they define the same objects (the proof of that fact is a good exercise, don’t just take it for granted! The proof is by induction.)

Given this second definition, one can prove that $B_i$ has $2^i$ nodes as follows:

Base case ($i = 0$). $B_0$ has a single node by definition, and $1 = 2^0$.

Assume the $i$th binomial tree has $2^k$ nodes, and prove that $B_{k+1}$ has $2^{k+1}$ nodes. The number of nodes in $B_{k+1}$ by definition is $2 \cdot 2^k$, using the inductive assumption, since $B_{k+1}$ comprises two copies of $B_k$. But $2 \cdot 2^k = 2^{k+1}$, which was to be shown.

This can also be proven, rather cleverly, by expressing the size of $B_i$ as a recurrence: $s(0) = 1$ and $s(i + 1) = 2s(i)$. One can then solve this recurrence. The characteristic polynomial is: $x - 2 = 0$, which has a closed form of $c_1 2^n + c_2 = 0$. Solving for the constants, given the initial condition (which is necessary in this case) gives that $c_2 = 0$ and $c_1 = 1$. So, the closed form for the recurrence is $s(i) = 2^i$, which was to be shown!

Note: binomial trees are not the same thing as binary trees. Several of you confused the two, even though I announced during the exam that the question was indeed about the former!
Question 12  [10 points] Why is the \texttt{insert} operation for regular heaps (not binomial heaps) a $O(\log n)$ operation, when the heap has $n$ elements in it?

\textbf{Answer}  Because regular heaps are essentially complete binary trees, and so the height of the data structure is $O(\log n)$. The height of the structure is the maximum number of data exchanges required to move an element from the next available slot to where it belongs, in order to restore the heap property.

I gave extra credit for proving that the height of a full binary tree is $\log n$. 