CS 3343 (Fall 2013) Exam 1
October 10, 2013
4:00 - 5:15pm (75 minutes)

Name: ________________ ID: ________________

• Don’t forget to put your name and ID on the cover page
• This exam is closed-book
• If you have a question, stay seated and raise your hand.
• Please try to write legibly – if I cannot read it, you may not get credit.
• Do not waste time – if you cannot solve a question immediately, skip it and return to it later.
• Try your best to answer each question. Partial credits will be given if you show that you have some ideas – but not according to the length of your answer.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Comparing functions</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2) Proof by definition</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3) Analyzing iterative algorithm</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4) Master method</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5) Recursion tree</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6) Substitution method</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7) Analyzing recursive algorithm</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>
1. Comparing functions
(a) (10 points) Order the following functions according to their order of growth from the lowest to the highest. If you think that two functions are of the same order (i.e. \( f(n) \in \Theta(g(n)) \)), put them in the same group.
\[ n^{2.01}, \ n + n^2 \log n, \ 2^n, \ n \log n, \ \log((n + 1)!), \ 2^{2n}, \ 5n + \log n, \ n^3 - 2n + 5. \]

(b) (10 points) For each pair of functions in the table below, determine whether \( f(n) \in O(g(n)) \), \( f(n) \in \Omega(g(n)) \), \( f(n) \in \Theta(g(n)) \), or all of them. It is NOT necessary to justify your answer.

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( O )</th>
<th>( \Omega )</th>
<th>( \Theta )</th>
<th>( g(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) ( n^2 - \log n )</td>
<td></td>
<td></td>
<td></td>
<td>( 10n^2 + n \log n )</td>
</tr>
<tr>
<td>2) ( n^2 + n + 10 \log n )</td>
<td></td>
<td></td>
<td></td>
<td>( n^2 - \sqrt{n} )</td>
</tr>
<tr>
<td>3) ( n \log n )</td>
<td></td>
<td></td>
<td></td>
<td>( n \sqrt{n} )</td>
</tr>
<tr>
<td>4) ( (n - 5) \log_3 n )</td>
<td></td>
<td></td>
<td></td>
<td>( n^2 + \log n )</td>
</tr>
<tr>
<td>5) ( \log_3(n^5) )</td>
<td></td>
<td></td>
<td></td>
<td>( 2 \log_2(n + 10) )</td>
</tr>
</tbody>
</table>

2. (10 points) Using the definition of \( O \), prove that \( 5n^2 + 5n + 1 \in O(n^2) \).
3. (10 points) Analysis of iterative algorithm. You are given the following sorting algorithm:

```plaintext
//Input: An array A of n real numbers
ExchangeSort(A[1..n]) {
    for (i = 1; i <= n - 1; i++) {
        for (j = i+1; j <= n; j++) {
            if (A[i] < A[j])
                exchange(A[i], A[j]);
        }
    }
}
```

a. In order to argue for the correctness of the algorithm, what can be claimed at the end of \( i \)-th iteration of the outer for loop? (Hint: what properties are satisfied by the first \( i \) numbers in the array?)

b. How many times will line 4 be executed, as a function of \( n \), in the best case and worst case, respectively?

c. How many times will line 5 be executed, as a function of \( n \), in the best case and worst case, respectively?

d. What is the asymptotic running time of this algorithm?
4. (20 points) Assume that $T(1) \in \Theta(1)$. Solve the following recurrence functions using the **master method**. If the master method cannot be applied, state the reason. Justify your answer.

a. $T(n) = 3T(n/3) + n^2$;

b. $T(n) = 4T(n/2) + n^2 + 3 \log n$;
c. $T(n) = 9T(n/3) + n/\log n$;

d. $T(n) = T(3n/2) - n^3$;

e. $T(n) = T(2n/3) + n\log n$;
5. (10 points) Recursion tree method.

Assume that $T(1) \in \Theta(1)$. Solve the following recurrence function using the \textbf{recursion tree method} to get an \textbf{asymptotically tight} bound.

$$T(n) = 3T(n/2) + n$$
6. (10 points) Substitution method.

Assume that $T(1) \in \Theta(1)$ and $T(n) = T(n/2) + 2T(n/4) + n$. Prove $T(n) \in O(n)$ using the substitution method.
7. (20 points) Analysis of recursive algorithms. Consider that you have to design an algorithm for a task. The input is an array with size $n$. You figured out that you can use divide and conquer to solve the task. You are comparing two choices of designing the algorithm. In the first choice, you divide the array into two subarrays with equal sizes, and recursively apply the algorithm to one subarray. In the second choice, you divide the array into four subarrays with equal sizes, and recursively apply the algorithm twice to two of the four subarrays. The following pseudocode describes the two algorithms respectively.

```
AlgX (A[1..n])
    if (n <= 1) return;
    p = floor(n/2);
    AlgX (A[1..p]);
end

AlgY (A[1..n])
    if (n <= 1) return;
    q = floor(n/4);
    AlgY (A[1..q]);
    AlgY (A[q+1..2q]);
    Combine results;
end
```

a. Write down and solve the recurrence relation for the running time of $\text{AlgX}$.

b. Assume that combining results in $\text{AlgY}$ can be done in $\Theta(1)$ time. Write down and solve the recurrence relation for the running time of $\text{AlgY}$.

c. If combining results in $\text{AlgY}$ takes $\Theta(\sqrt{n})$ time, what would be the running time of $\text{AlgY}$?
d. Above we have assumed that parameter passing during procedure calls takes constant time, regardless of the size of the array. This is valid in most systems because a pointer or reference to the array is passed, not the array itself. Now let’s say that we have to pass the parameter by actually creating a copy of the subarray during procedure call and the time is $f(n) = \Theta(n)$ for passing an array of size $n$. Answer questions (a)-(c) again.

e. In AlgY, if instead of dividing $A$ into 4 subarrays, let’s say we divide it into $2^k$ subarrays, where $k$ is an integer. We then recursively apply the algorithm $k$ times, each time to a different subarray. What will be the running time of the algorithm in terms of $n$ and $k$, under the different assumptions mentioned in (b)-(d)? For each scenario, does the efficiency of the algorithm improve or become worse when $k$ increases? Justify your answer.
Extra page
Extra page
Extra page
Extra page
Extra page