CMPS 201 Practice Exam 1

CMPS 201 Practice Exam 1 Oct. 9, 2002

70 mins., 100 Pts. Closed book, + Master Theorem Answer on this Exam

Questions about meaning can be asked, to clarify any question. Budget your time. Be accurate! Doing most problems, some completely correct, is better than doing all problems and none completely correct.

Points indicated on these problems may not be accurate.

A statement of the Master Theorem will be given on closed book exams. It will be as in the text with errata applied. Students should know the inequality axioms in Chapter 1, which will not be given on the exam.

1 Prove or Give Counter-Example

Any unstarred exercise in chapters 1–5 might be selected, where it is of the form “prove or disprove” or “prove or give a counter-example” and does not require a very long answer.

2 Induction (20 pts)

Let $F(0) = 1$, $F(1) = 1$, and $F(n) = F(n - 1) + 2 F(n - 2)$ for integers $n \geq 2$. Prove (directly, by induction) the correct statement among the following (only one is correct):

1. $F(n) \geq \frac{1}{100} \left( \frac{3}{2} \right)^n$ for all integers $n \geq 0$.
2. $F(n) \leq 100 \left( \frac{5}{2} \right)^n$ for all integers $n \geq 0$.

Hint: You might develop the proof for both statements simultaneously, using $\bigcirc$ and $\square$ for the relation and the coefficient, until you see what specific facts are needed to complete the correct proof.

3 Median of 5 (10 pts)

Describe the main ideas of how to find the median of 5 elements with a minimum number of comparisons. You may use pictures, labeling the elements A, B, C, D, and E. Do not write code.

4 Second-Largest Key (10 pts)

Describe the main ideas of an adversary strategy to demonstrate a lower bound on the number of comparisons needed to determine the second-largest key in a set of $n$ keys, using comparisons. Describe the bookkeeping needed. What lower bound does it force? Explain why. (It should be clear how to convert your explanation into a proof, but a formal proof is not needed.)

(Alternative: Find both the maximum and minimum keys)
5  Heap Operations (10 pts)

A (5 pts).  Illustrate the operation of deleteMax by showing successive pictures.  
Indicate clearly what elements get compared and/or moved.

B (5 pts).  Illustrate the operation of heapInsert for key 50 by showing successive pictures.  
Indicate clearly what elements get compared and/or moved.

6  Quick Sort and Merge Sort

The problem will be similar in format to the preceding one, but you will need to demonstrate the operation of quicksort or mergesort, and their subroutines, partition and merge.
7 Sorting Debate (15 pts)

To prepare for a debate in which you do not know which side you will need to argue, prepare a 2D table that gives one or two significant advantages (not an exhaustive comparison) of the sorting method (in each row, leftmost column) over its competitors (listed in subsequent columns).

1. The four methods to compare are heap, insertion, merge, and quick sorts.
2. The advantages may be in time and/or space.
3. You may consider best, average, and worst cases to make your debating point.
4. Be specific about growth rates ($\Theta()$) involved.
5. Constant-factor advantages are not required (and are discouraged). If you do mention constant factors, you must state what operations you are considering and what the actual factors are.
6. The correct answer might be “none” for certain cases. “None” is acceptable if the only differences are in constant factors.

There should be a total of 15 entries among the 12 nondiagonal cases, counting “nones”, if any, and not mentioning constant factors.

Example: in bubble sort’s row and wander sort’s column the entry might be \texttt{“$\Theta(n^2)$ worst case time vs. $\Theta(n!)$”}.

<table>
<thead>
<tr>
<th>Advantage of heap sort</th>
<th>over heap sort</th>
<th>over insertion sort</th>
<th>over merge sort</th>
<th>over quick sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantage of insertion sort</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advantage of merge sort</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advantage of quick sort</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8 Asymptotic Values (25 pts)

Give Big-$\Theta$ expressions for each problem; for full credit they must be fully simplified. For all recurrences, the base case is $T(1) = 1$. 
9 Tripartite Sort (20 pts)

Analyze the running time of the following procedure. Assume smallsort() requires constant time when called on subranges of 2 or less, as it is in this case.

```c
void tri_sort(A, low, hi) {
    if (hi - low < 3)
        smallsort(A, low, hi);
    else
        mid1 = low + (hi - low) / 3;
        mid2 = hi - (hi - low) / 3;
        tri_sort(A, low, mid1);
        tri_sort(A, mid1, hi);
        tri_sort(A, low, mid2);
}
```

Be sure to define the meanings of symbols you use, and briefly explain the steps.

10 Quick Sort Variant (20 pts)

You and a friend are collaborating on an implementation of quicksort. Your friend is going to write a “souped up” version of partition. You have agreed on the following specs:

1. The header is: int partition(String elts[], int low, int high);
2. The returned value will be in the range low through high, (call it b).
3. Upon return, elts[low .. b-1] will be less than elts[b] (the pivot), and elts[b+1 .. high] will be greater than or equal the pivot.

Your job was to write quicksort() (see header below). Unfortunately, the night before the project is due, you get an e-mail message from your friend:

**Help! partition() has a bug, and I have to leave to catch a plane.**

Here is what I did:

I tried to improve the choice of pivot with the following idea. Instead of just taking elts[low] as the pivot, I search forward from low+1 until I find
elts[i] < elts[i-1]. Then I use elts[i] as the pivot, instead of elts[low]. I figured that this would avoid ever using the maximum element as the pivot, which was one of the worst cases. The bug is that the condition may not hold for any i <= high. I didn’t know what to do, so I just return the "impossible" value, b = high+1 when this happens. I hope you can deal with it.

Your new job is to write a version of quicksort that works with your friend’s code, anyway. The header, or prototype:

```c
void quicksort(String elts[], int low, int high);
```

Assuming partition is coded efficiently and your quicksort is coded efficiently under the conditions given, what are the best case, average case, and worst case \( \Theta \) running times for your quicksort on \( n = (high - low + 1) \) elements? Justify the best and worst cases by giving an example input for each.

### 11 Sort Correctness (10 pts)

The objective of sort4, given below, is to sort sets of exactly 4 elements. Decide whether it works or not, and give a solid argument (almost a proof) to support your position.

Decide based on the design. Don’t look for a syntax error. swap does the obvious operation.

```c
sort4(A)
{
    int x;
    if (A[1] > A[3]) swap(A, 1, 3);
    {
        x = 1;
        swap(A, 3, 4);
    }
    else
    
        x = 0;

    if (x == 0)
    {
        if (A[1] > A[4]) swap(A, 1, 4);
    }
    else
    {
        if (A[1] > A[2]) swap(A, 1, 2);
    }
}
```

### 12 Improved Merge Sort (15 pts)

Merge Sort calls merge as a subroutine. The standard implementation of merge uses how many comparisons to merge two sorted subranges of length \( n/2 \) each? Give the leading term with the correct coefficient.

A (2 pts) In the worst case:
B (3 pts) In the best case:
The project paying your support depends heavily on sorting efficiency. The professor in charge gives you a paper to read (downloaded off the web) that describes a new merging strategy. The abstract states that it uses only $\lceil \frac{2n}{3} + 2 \rceil$ comparisons (for $n > 1000$) to merge two sorted subranges of length $n/2$ each, in the worst case. The idea is to skip around, like binary search, with some additional bookkeeping.

C (4 pts) Based on the abstract’s claim, how many comparisons would merge sort require in the worst case if it used this merging strategy? Give the leading term with the correct coefficient.

D (6 pts) The paper is 50 pages of complicated code and analysis. Can you give the professor a good reason why you should not spend this quarter (and maybe the next) figuring out this paper and implementing it?

13 Efficient Medians (25 pts)

The generalized median-of-medians method breaks a set of $n$ elements into sets of $s$, where $s$ is a small integer. Let $m(s)$ denote the number of comparisons required to find the median of $s$ elements, worst case. The recurrences for comparisons needed in several particular cases, are:

$$T_{2k+1}(n) = \frac{(m(k) + k)n}{2k + 1} + T_{2k+1} \left( \frac{2n}{4k + 2} \right) + T_{2k+1} \left( \frac{(3k + 1)n}{4k + 2} \right)$$

$$T_3(n) = \frac{(m(3) + 1)n}{3} + T_3 \left( \frac{2n}{6} \right) + T_3 \left( \frac{4n}{6} \right)$$

$$T_5(n) = \frac{(m(5) + 2)n}{5} + T_5 \left( \frac{2n}{10} \right) + T_5 \left( \frac{7n}{10} \right)$$

$$T_7(n) = \frac{(m(7) + 3)n}{7} + T_7 \left( \frac{2n}{14} \right) + T_7 \left( \frac{10n}{14} \right)$$

A (3 pts) What are the optimum values of $m(3)$ and $m(5)$?

B (10 pts) Ignoring truncation issues when $n$ is not a multiple of 5, find a constant $C_5$ for which $T_5(n) = C_5 n + O(1)$, and show work to verify your answer.

C (12 pts) Ignoring truncation issues when $n$ is not a multiple of 3, what can you say about the solution of $T_3(n)$, particularly in comparison to $T_5(n)$? Show work to justify your conclusion.