Warning:

This version of the sample midterm includes the solutions to the problems.

You might want to look at the version without solutions first to help you gauge your preparedness for the test.

The test begins on the next page.

SHORT ANSWERS

1. Complete the following sentence: The three steps in the divide-and-conquer paradigm are.....

   1. **Divide** the problem into a number of subproblems.
   2. **Conquer** the subproblems by solving them recursively.
   3. **Combine** the solutions to the subproblems into the solution to the original problem.

2. Given the following statements:

   p: $f(n)=\omega(g(n))$
   q: $f(n)=\Omega(g(n))$
   r: $f(n)=O(g(n))$
   s: $f(n)=\Theta(g(n))$

   What is the truth value of the following propositions:

   a. $p \wedge r$

   b. $q \wedge r \Rightarrow s$

   Note that: $p \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$; $q \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} \geq c,\ 0<c<\infty$;
   $r \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c,\ 0<c<\infty$; $s \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = c,\ 0<c<\infty$

   a. It cannot be that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ and $\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c,\ c<\infty$, so the truth value of $\wedge r$ is FALSE.

   b. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \geq c,\ 0<c<\infty$ and $\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c,\ 0<c<\infty$ then it must be that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c,\ 0<c<\infty$, so that $q \wedge r \Rightarrow s$ is TRUE.

3. Mathematically explain why $\frac{1}{3}n^4 - rn^3$ is $O(n^4)$.

   This can be done using either of the two approaches:

   $$\lim_{n\to\infty} \frac{\frac{1}{3}n^4 - rn^3}{n^4} = \lim_{n\to\infty}\left(\frac{1}{3} - \frac{r}{n}\right) = \frac{1}{3} < \infty, \text{ so } \frac{1}{3}n^4 - rn^3 \text{ is } O(n^4).$$

   For $c = 1$, $n \geq n_0 = 1 > 0$, clearly $c\left(\frac{1}{3}n^4 - rn^3\right) = \left(\frac{1}{3}n^4 - rn^3\right) < n^4$, so $\frac{1}{3}n^4 - rn^3$ is $O(n^4)$.

4. A randomized version of quicksort is implemented by first finding a random permutation of the input and then calling quicksort. What is the worst case running time of this algorithm?

   Assuming that a random number in the range [1,k] can be generated in O(1) time, the random permuation can be generated in O(n) time. The worst case running time for quicksort is O(n^2). Thus the total time is O(n)+O(n^2)=O(n^2).

5. Suppose we are dealing with max-heaps (where the largest element resides in the root). State the running times of heapsort for n elements that are already in:

   a. ascending order?

      Time to build the heap: O(n)
      Time to extract the n elements: O($n$log n)
      Total time: O($n$log n)

   b. descending order?

      Time to build the heap: O(n)
      Time to extract the n elements: O($n$log n)
      Total time: O($n$log n)

      Note that the original array is already a heap, but O(n) build time is still used by the algorithm.

6. Randomized-Quicksort uses a random number generator to decide which element to use as the split value (as opposed to just using the first element). During the running of Randomized-Quicksort, how many calls are made to the random number generator in the worst case? Assume that the procedure is used to sort n numbers.

   In the worst case the random number generator always results in the use of either the largest or the smallest value in the "group" as the split value. Thus, the list would always be split into a group of size one (1) and another group with all the other values. As a result, the number of split values that would be selected is $(n–1)$.

7. In any binary tree on n elements, the number of NIL pointers is $(n+1)$ because.....

   Since there are n elements (nodes), there are a total of $2n$ pointers. Further, all nodes *except* the root have a pointer to them, so that there are a total of $(n–1)$ pointers that are not NIL. Thus, the number of nodes that are NIL must be $2n – (n–1) = (n + 1)$.

8. Outline a lower bound proof showing that $\Omega(\log n)$ comparisons are necessary for the problem of finding the value of i such that $x_i = i$ in the increasing sequence $x_1, x_2,..., x_n$ or determining that no such i exists.

   Any comparison of x_i with i will: 1) identify a position where the desired condition holds, 2) identify a position where $i<x_i$ (so that the desired condition can only hold for some position larger than i), or 3) identify a position where $i>x_i$ (so that the desired condition can only hold for some position smaller than i). If each comparison always eliminates the fewest possible elements, then the comparison should always be done on the middle of the remaining elements (to maximize the number eliminated). Thus, each comparison will eliminate half (1/2) of the elements, so that a total of $\log n$ comparisons are necessary in the worst case for finding a value of i or determining that no such value exists.

LONG QUESTIONS

9. Suppose we have an n element min-heap stored in an array. Deleting the root (remove the root, move the "last" leaf to the root, and percolate it down until the heap property is restored) and inserting a new element (add a new "last" leaf, and percolate up until the heap property is restored) both take $O(\log n)$ time. Which of the two operations would you expect to be faster? Justify your answer.

   ("Deleting the root" or "Inserting a new element" with no explanation gets -5 out of 30).

Inserting a new leaf would be faster, for two reasons:

1) A new value added as a leaf requires only one comparison to determine if it should be moved up a level, whereas when a value is deleted two comparisons are needed to precolate the new root down each level.

2) The average height of a value in the heap is:

$$\frac{1}{n}\left(\left(\frac{n}{2}\times 0\right)+\left(\frac{n}{2^2}\times 1\right)+\cdots+\left(\frac{n}{2^{\log n}}\right)\times(\log n\ -\ 1)\right)=\sum_{i=1}^{\log n}\left(\frac{1}{2^i}(i-1)\right)$$

$$=\frac{1}{2}\sum_{i=0}^{\log n}\left(\frac{i}{2^i}\right)<\frac{1}{2}\sum_{i=0}^{\infty}\left(\frac{i}{2^i}\right)=\frac{1}{2}\left(\frac{\frac{1}{2}}{\left(1-\frac{1}{2}\right)^2}\right)=1$$

Thus, in both cases the new value will in expectation move to a position of height 1. Thus, an inserted element will on average move up only 1 level. Similarly, in deletion the moved element would on average move down ($\log n - 1$) levels if it was randomly selected, and in fact it is known to be smaller than at least $\log n$ elements.

10. A hash table of size m is used to store n items, with $n \le m/4$. Open addressing is used for collision resolution.

    a. Assuming uniform hashing, show that for $i = 1, 2, \ldots, n$, the probability that the ith insertion requires strictly more than k probes is at most 2^{-2k}.

    Assuming uniform hashing, the probability that the ith insertion requires strictly more than k probes is
    P[first k probes are to full slots] $< \left(\frac{i}{m}\right)^k < \left(\frac{n}{m}\right)^k < \left(\frac{(m/4)}{m}\right)^k = \left(\frac{1}{4}\right)^k = 2^{-2k}$.

    b. Show that for $i = 1, 2, \ldots, n$, the probability that the ith insertion requires more than $\log n$ probes is at most $1 / n^2$.

    Using the result from part a., the probability that the ith insertion requires more than $\log n$ probes is at most $2^{-2\log n} = \left(2^{-\log n}\right)^2 = \left(\frac{1}{n}\right)^2$
    $= \frac{1}{n^2}$