CS361 Midterm

Solutions

1. (10 points) Given $f(n) = n^{\log n}$ and $g(n) = 2^n$, is $f = o(g)$, $\Theta(g)$, or $\omega(g)$? Explain your answer.

   Answer. $f = o(g)$. There are several ways to see this: here's one. Recall that if $\log f = o(\log g)$ then $f = o(g)$, although the reverse is not true. But

   $$\log f = \log(n^{\log n}) = (\log n)^2 \ ,$$

   while

   $$\log g = \log 2^n = n \ .$$

   Since $(\log n)^2 = o(n)$ we're done.

   Another way to see this is to calculate the ratio, and use the fact that $n^{\log n} = 2^{(\log n)^2}$:

   $$\frac{f(n)}{g(n)} = \frac{2^{(\log n)^2}}{2^n} = 2^{(\log n)^2 - n} \ .$$

   When n is large, $(\log n)^2 - n$ is very negative, so this ratio goes to zero.

2. (20 points) Some standard software libraries use Quicksort for numbers and strings, and Mergesort for user-defined classes. Explain why (in both cases).

   Answer. Quicksort uses more comparisons than Mergesort does; $2n \ln n$ vs. $n \log_2 n$, which since $2 \ln 2 \approx 1.39$ is about 40% more.

   Thus if comparisons take a long time to do, and the lion's share of the running time comes from doing comparisons (like apply a user-defined comparison operator) then Mergesort will be faster.

   However, for simple objects like numbers and strings, comparisons are fast and easy, and other contributions to the running time become important. In particular, Quicksort moves things around in memory less than Mergesort does (partly because it sorts in place rather than copying things into additional memory), so Quicksort is superior.

3. (30 points total) 3-Mergesort is a version of Mergesort which divides a list of size n into *three* lists of size $n/3$, sorts them recursively, and then merges the sorted lists together.

   (a) (5 points) How many comparisons does the three-way merge operation need to merge three sorted lists of size $n/3$ into the final list of size n?

   Answer. Each step of the Merge operation has to choose the smallest of 3 elements (i.e. which of the 3 lists to take the next entry from). This takes 2 comparisons. Thus the Merge operation takes at worst $2n$ comparisons. (We can be a little cleverer and remember comparisons from previous steps, but there's not guarantee that this will always help.)

   (b) (10 points) Using your answer to (a), what is the recurrence for the number $f(n)$ of comparisons that 3-Mergesort does on a list of size n? Ignore the difference between n and $n-1$, but get the multiplying constants right.

   Answer. Calling ourselves three times on lists of size $n/3$, plus a driving term of $2n$, gives

$$f(n) = 3f(n/3) + 2n \ .$$

   (c) (5 points) What is the Θ-solution to this recurrence? Justify your answer.

   Answer. The homogeneous solution is $\Theta(n)$ and so is the driving term, so $f(n) = \Theta(n \log n)$ — just like Mergesort.

   (d) (10 points) Find the constant hidden in Θ. Is 3-Mergesort faster or slower than the traditional Mergesort? By how much? (Feel free to give your answer symbolically or numerically.)

   Answer. Trying a solution of the form $f(n) = An \log_3 n$ gives

$$\begin{aligned} An \log_3 n &= 3A(n/3)\log_3(n/3) + 2n \\ &= An(\log_3 n - 1) + 2n \\ &= An \log_3 n - An + 2n \end{aligned}$$

   Canceling $An \log_3 n$, dividing by n, and solving for A gives $A = 2$, so $f(n) = 2n \log_3 n$.

   Traditional Mergesort takes $n \log_2 n$ comparisons. To compare their running times,

$$\frac{2n \log_3 n}{n \log_2 n} = 2 \log_3 2 = \log_3 4$$

   Thus 3-way Mergesort takes $\log_3 4 \approx 1.26$ as many comparisons as traditional Mergesort, so it is about 26% slower.

   Note that 3-Mergesort *has to be* slower than Mergesort: since $n \log_2 n \approx \log_2 n!$, the number of comparisons that Mergesort does is essetially optimal by the "Twenty Questions" argument.

4. (20 points) Suppose I have a hash table with m locations. I want to store n items in it; assume that the hash function is random. Roughly how large (i.e., within Θ) can n be before I start to get *triple* collisions, where three items get hashed to the same location? Roughly how large is this value of n if $m = 1,000,000$?

   Answer. The expected, or average, number of triple collisions is just the number of possible triples, $\binom{n}{3}$, times the probability that a given triple all go to the same place, which is $1/m^2$. Since $\binom{n}{3} = \Theta(n^3)$, this is

$$\binom{n}{3} \frac{1}{m^2} = \Theta\left(\frac{n^3}{m^2}\right) \ .$$

   This is $\Theta(1)$, suggesting that triple collisions start to appear, when $n^3 = m^2$, or when $n = m^{2/3}$. For $m = 1,000,000 = 10^6$, this gives $n = 10^4 = 10,000$ as a rough estimate.

5. (20 points) Consider the recurrence

$$f(n) = Af(n/2) + n$$

   where A is some constant. The type of solution for $f(n)$ depends on the value of A. Describe how. Specifically, give solutions for $f(n)$ within Θ that hold for each range of A.

   Answer. Let's first decide when the driving term n matters. The homogeneous solution, i.e., the solution to

$$f(n) = Af(n/2) \ ,$$

   is of the form n^a. To find a, we substitute the guess $f(n) = n^a$, and get

$$n^a = A(n/2)^a = An^a/2^a \ .$$

   Cancelling n^a, we get $2^a = A$, or $a = \log_2 A$.

   Now, if $A < 2$, we have $a < 1$, and the homogeneous solution is $n^a \ll n$. In this case the driving term dominates, so $f(n) = \Theta(n)$.

   On the other hand, if $A > 2$, then $a > 1$, and the homogeneous solution is $n^a \gg n$. Thus the driving term doesn't matter, and $f(n) = n^a = n^{\log_2 A}$.

   Finally, the borderline case is $A = 2$ (just like Mergesort). Now $a = 1$, the homogeneous solution is n, and the driving term kicks this up to $f(n) = \Theta(n \log n)$.