15-451 Homework 2

Feb 4, 2008

Due: Mon-Fri, February 11-15

1. This is an oral presentation assignment. You should work in groups of three. At some point before **Sunday, February 10 at 11:59pm** your group should sign up for a 1-hour time slot on the signup sheet on the course web page.
2. Each person in the group must be able to present every problem. The TA/Professor will select who presents which problem. The other group members may assist the presenter.
3. Your are not required to hand anything in at your presentation, but you may if you choose.
4. Direct questions to Bryant Lee (bryantl@cs.cmu.edu)

1 Question 1

[**Median of two sorted arrays**] Let A and B be two sorted arrays of n elements each. We can easily find the median element in A - it is just the element in the middle - and similarly we can easily find the median element in B. (Let us define the median of $2k$ elements as the element that is greater than $k-1$ elements and less than k elements.) However, suppose we want to find the median element overall - i.e., the nth smallest in the *union* of A and B. How quickly can we do that? You may assume there are no duplicate elements.

Your job is to give tight upper and lower bounds for this problem, in terms of number of comparisons. Both bounds should be an *exact* number of comparisons, and NOT in terms of big O and big Ω. Specifically, for some function $f(n)$,

1. Give an algorithm whose running time (measured in terms of number of comparisons) is $f(n)$, and
2. Prove a lower bound showing that any comparison-based algorithm must make $f(n)$ comparisons in the worst case.

2 Question 2

[Tight upper/lower bounds]

1. First, show that $2n-1$ comparisons is both a necessary and sufficient number of comparisons to merge two sorted arrays of size n. This is a (graded) warmup for the question below.
2. Consider the following problem.

   INPUT: n^2 distinct numbers in some arbitrary order.

OUTPUT: an $n \times n$ matrix of the inputs having all rows and columns sorted in increasing sorted order.

EXAMPLE: $n = 3$, so $n^2 = 9$. Say the 9 numbers are the digits 1,...,9. Possible outputs include:

$$\begin{vmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{vmatrix} \text{ or } \begin{vmatrix} 1 & 4 & 5 \\ 2 & 6 & 7 \\ 3 & 8 & 9 \end{vmatrix} \text{ or } \begin{vmatrix} 1 & 3 & 4 \\ 2 & 5 & 8 \\ 6 & 7 & 9 \end{vmatrix}$$

It is clear that we can solve this problem in time $O(n^2 logn)$ by just sorting the input (remember that $log(n^2) = O(logn)$ and then outputting the first n elements as the first row, the next n elements as the second row, and so on. Your job in this problem is to prove a matching $\Omega(n^2 logn)$ lower bound in the comparison-based model of computation.

For simplicity, you can assume n is a power of 2.

Some hints: Show that if you could solve this problem using $o(n^2 log(n))$ comparisons (in fact, in less than $n^2 lg(n/e)$ comparisons), then you could use this to violate the $lg(m!)$ lower bound for comparisons needed to sort m elements. You may want to use the fact that $m! > (m/e)^m$.

3 Question 3

[**Amortized analysis**] In class you saw a binary counter that we can increment efficiently. Now suppose we want a binary counter that supports both increment and decrement efficiently. One way to do this is to represent a number using *two* binary strings, $c1$ and $c2$, such that the number is actually $c1 - c2$. Note that numbers can be represented in more than one way.

Suppose you perform a sequence of increments and decrements, starting from (0,0). Write an algorithm for the following operations:

1. Increment(c1, c2). The operation that increments the number represented by $c1 - c2$.
2. Decrement(c1, c2). The operation that decrements the number represented by $c1 - c2$.

Your algorithm should have a constant amortized cost per operation, where we measure cost only by number of bit flips (all other operations are free). What's the worst case for a single operation in the sequence?