Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Fall 2004
Homework 2a
Due on Thursday, 09/09/04 (beginning of class)

1. **[25 Points] Fast Multiplication Circuits!** In this problem we consider the task of multiplying two n-bit numbers, x and y. Note that size is in terms of the number of bits, n. You can assume bit-shifts and additions have a cost proportional to the number of bits being operated upon. For simplicity, assume that n is a power of 2 (This is generally the case anyhow in typical computer architectures, but we could easily relax this assumption similar to the way we fixed mergesort).

   (a) Explain why the time complexity of the standard algorithm (performing normal multiplication, only in base 2 rather than base 10) for this problem is $\Theta(n^2)$.

   Some fast multiplication circuits use the following clever divide-and-conquer algorithm. Divide x and y in half. Thus, $x = a2^{n/2} + b$ and $y = c2^{n/2} + d$ where a, b, c, d are $n/2$-bit numbers. The product of x and y, call it z, can now be computed by the following steps:

   (a) $temp1 = (a + b) \cdot (c + d)$
   (b) $temp2 = a \cdot c$
   (c) $temp3 = b \cdot d$
   (d) $z = temp2 \cdot 2^n + (temp1 - temp2 - temp3) \cdot 2^{n/2} + temp3$.

   (a) Briefly explain why this algorithm really gives us the desired product.
   (b) Observe that the terms $(a+b)$ and $(c+d)$ may have either $\frac{n}{2}$ bits or $\frac{n}{2}+1$ bits. Assume (for now) that they each have exactly $\frac{n}{2}$ bits. The above algorithm performs 3 multiplications of $n/2$ bit numbers plus some additions and shifts. (Multiplying a binary number by a number 2^{ℓ} can be accomplished by simply performing ℓ bit shifts to the left! This takes $\Theta(\ell)$ time.) Describe a recursive algorithm that uses this idea and write a recurrence relation for the time complexity, $T(n)$.
   (c) Find the asymptotic time complexity of the divide-and-conquer algorithm by using a recursion tree analysis. Show each step of your computation. Your final answer should be in the form $O(n^c)$ where c is an **actual number**. (In other words, don't leave c as some mathematical expression.)
   (d) Describe how the algorithm can be fixed to take care of the case that $a + b$ and $c + d$ are possibly $n/2 + 1$-bit numbers. (The algorithm should not be slower asymptotically after making this fix.)

2. **[20 Points] Extra Credit! The Diogenes Problem!**

   IMPORTANT: If you decide to do this problem, **turn it in on a separate sheet of paper**. *EC is not due until Friday at 6pm (in my office).*

   Do Problem 4-6 (page 87) in CLR, parts (b) and (c). By the way, who was Diogenes and what is he doing in this problem?