CS41 Homework 7

This homework is due at 11:59PM on Wednesday, November 2. Write your solution using LaTeX. Submit this homework in a file named `hw7.tex` using **github**.

This is a partnered homework. You should primarily be discussing problems with your homework partner. It's ok to discuss approaches at a high level with others. However, you should not reveal specific details of a solution, nor should you show your written solution to anyone else. The only exception to this rule is work you've done with a lab teammate *while in lab*. In this case, note (in your **homework submission poll**) who you've worked with and what parts were solved during lab.

1. **Integer Multiplication.** Recall the Karatsuba algorithm for integer multiplication for class, which multiplies two n-digit base-N numbers a, b by:

$$\begin{aligned} a \cdot b &= (a_L N^{n/2} + a_r)(b_L N^{n/2} + b_R) \\ &= a_L b_L N^n + (a_L b_R + a_R b_L) N^{n/2} + a_R b_R \\ &= A N^n + (C - A - B) N^{n/2} + B, \end{aligned}$$

   where the three $(n/2)$-digit multiplications are:

   - $A = a_L \cdot b_L$
   - $B = a_R \cdot b_R$
   - $C = (a_L + a_R)(b_L + b_R)$

   Consider an algorithm for integer multiplication of two n-digit base-N numbers where each number is split into three parts, each with $n/3$ digits.

   (a) Design such an algorithm, similar to the integer multiplication we did in class. Your algorithm should describe how to multiply two integers using only six multiplications (instead of the straightforward nine).

   (b) Determine an asymptotic upper bound for the running time of your algorithm. (Write it as a recurrence, and then solve the recurrence.)

   (c) Is this algorithm asymptotically faster than Karatsuba multiplication? That is, is it better to use the algorithm that breaks an integer into three parts, or two parts?

2. **Counting significant inversions** (K&T 5.2)

   Recall the problem of finding the number of inversions between two rankings. As we saw, we are given a sequence of n numbers $a_1, a_2, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair of indices $i < j$ such that $a_i > a_j$.

   We previously used counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a *significant inversion* if $i < j$ and $a_i > 2a_j$. Give an $O(n \log n)$ algorithm to count the number of significant inversions.

3. **Database Queries** (K&T 5.1) You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values (so there are $2n$ values total). You'd like to determine the *median* of this set of $2n$ values, defined as the n-th smallest value.

   The only way you can access these values is through *queries* to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the k-th smallest value it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

   - Design an algorithm that finds the median value using at most $O(\log n)$ queries. Full pseudocode is not necessary, but you must clearly explain how it works, and you must handle all edge cases; e.g., do not assume that n is even.
   - Show that your algorithm correctly returns the median.
   - Prove that your algorithm uses only $O(\log n)$ queries.

4. **(extra challenge) Implementing Karatsuba Multiplication**

   We saw in class that Karatsuba Multiplication has a polynomially better runtime ($O(n^{\log_2(3)})$ vs $O(n^2)$) than the algortihm you likely learned in school. How does this asymptotic improvement translate in real-world runtime?

   Your task in this problem is to implement Karatsuba Multiplication, and then test your implementation by timing your Karatsuba multiplication vs the standard algorithm. It is likely that the numbers you're multiplying will need to be quite large for the runtime improvements to take effect. How many digits do the inputs need to be before Karatsuba begins to have significant runtime gains?

   Note: If you tackle this extra challenge, I encourage you to implement this in Python, which has built-in support for arbitrarily large numbers.