CMU 15-252 **Spring 2017**

Homework 6

due March 24th in class

---

1. Read the notes on Fields and Polynomials posted on the course webpage.

2. In this question, we explore the computational complexity of polynomial multiplication.

   (a) Suppose we are given two degree-d polynomials $P(x)$ and $Q(x)$ as a list of their coefficients. Using the definition of polynomial multiplication, what is the running time of evaluating their product in terms of d, assuming a single field operation (addition or multiplication) takes constant time to compute?

   (b) Let's try another method of computing the product. Given two degree-d polynomials $P(x)$ and $Q(x)$ as a list of coefficients, we first evaluate them at $2d+1$ points to convert them to the value representation (in a regular value representation of a degree-d polynomial, we would evaluate the polynomial at $d+1$ points, but here we will need the evaluation at $2d+1$ points). To put it more explicitly, we pick $2d+1$ distinct field elements $a_1, \ldots, a_{2d+1}$, and compute $(P(a_1), \ldots, P(a_{2d+1}))$ and $(Q(a_1), \ldots, Q(a_{2d+1}))$. These are the value representations of $P(x)$ and $Q(x)$ respectively. Then the value representation of the product $PQ(x)$ is $(P(a_1)Q(a_1), \ldots, P(a_{2d+1})Q(a_{2d+1}))$. We convert this back to the coefficient representation using Lagrange interpolation. What is the running time of this method in terms of d (again, assuming field operations take constant time)? And why did we evaluate the polynomials at $2d+1$ points rather than $d+1$ points?

   <u>A note for the interested:</u> Evaluating the polynomials at a carefully chosen set of $2d+1$ points gives rise to *Fast Fourier Transform*, which computes the product of two degree-d polynomials in time $O(d \log d)$. We plan to cover this in the next 252 lecture.

3. This problem is concerned with doing a bit of a generalization of the Karatsuba multiplication algorithm. Suppose we wish to multiply two n-bit numbers A and B. Say we break up A into *three* blocks (unlike the *two* blocks in Karatsuba), writing $A = a_2 2^{2n/3} + a_1 2^{n/3} + a_0$. (Assume n is divisible by 3.) Similarly we break up B as $B = b_2 2^{2n/3} + b_1 2^{n/3} + b_0$. Our goal is to compute $C = A \cdot B$, which of course can be written as

   $$C = c_4 2^{4n/3} + c_3 2^n + c_2 2^{2n/3} + c_1 2^{n/3} + c_0,$$

   where

   $$c_4 = a_2 b_2, \quad c_3 = a_2 b_1 + a_1 b_2, \quad c_2 = a_2 b_0 + a_1 b_1 + a_0 b_2, \quad c_1 = a_1 b_0 + a_0 b_1, \quad c_0 = a_0 b_0. \tag{1}$$

   (a) Explain how, if we can get a hold of $c_0, \ldots, c_4$, we can write out C in $O(n)$ time. (Hint: the little hassle here is understanding how many bits long $c_0, \ldots, c_4$ are, and handling "overflow".)

   (b) Evidently from (1), we could get $c_0, \ldots, c_4$ by doing 9 recursive multiplies of $n/3$-bit numbers, plus some addition. Explain how — if we could somehow get $c_0, \ldots, c_4$ using just 5 recursive multiplies of $n/3$-bit numbers, plus some addition — we could multiply n-bit numbers in time $O(n^{\log_3 5})$. (Hint: in analyzing the recursion, you may assume n is a power of 3. Actually, this is not a costly assumption, since you could artificially pretend n was the next-larger power of 3, and that would only change it by a constant factor.)

(c) Explain why you can get $c_0, \ldots, c_4$ by doing just 5 recursive multiplies of $n/3$-bit numbers[1], plus some additional arithmetic taking $O(n)$ time.
Hint: We'll make use of part (b) of the previous question (Question 2). Think of the polynomial $A(x) = a_2x^2 + a_1x + a_0$ and similarly $B(x)$. Think about polynomial interpolation on 5 values, say $-2, -1, 0, 1, 2$. Notice that even though we're doing integer multiplication, somehow rational numbers get involved...)

[1] Well, maybe $n/3 + 1$ or $n/3 + 2$ bits. Who's counting?