CSE 521
Assignment 5
Due Tuesday, May 6, 2003

1. The FFT can be used as part of an algorithm to multiply polynomials with complex coefficients. This can be done with some loss of accuracy with floating point numbers. However, what if we want to multiply polynomials over the integers with total accuracy. In this case we need an integer like method. To achieve this we can use a large prime number p and rely on the fact that the integers $\mod p$ are a field with some roots of unity. To be more specific, let n be a power of 2 and choose k so that $p = kn + 1$ is a prime. There is a number g such that the set
$$\{g^1 \bmod p, g^2 \bmod p, ..., g^{p-1} \bmod p\} = \{1, 2, \ldots, p-1\},$$
that is, in mathematical terms g is a generator of Z_p^*. For example if $p = 17$ then 2 is a generator of Z_{13}^*, but, for example 5 is not. Now, let $\omega = g^k$.

   (a) Show that ω is a principal n-th root of unity mod p. That is, $\omega^n \bmod p = 1$ and $\omega^i \bmod p \neq 1$ for $0 < 1 < n$. Note that the mapping $i \to g^i$ on the domain $\{1, \ldots, p-1\}$ is a one-to-one function for a g a generator of Z_p^*.

   (b) Argue that the DFT and inverse DFT are well defined (non-singular) with ω chosen as a principle n-th root of unity mod p.

   (c) Assuming that p is $\log n$ bits and that arithmetic operations on $\log n$ bit numbers can be done in constant time, explain how the FFT can be computed in $O(n \log n)$ time. You can assume that p and ω are known. Note that for positive integers a and b, a^b takes exponential in the length of b to compute, because the result is that long.

   (d) Suppose you want to multiply two integer polynomials accurately using the FFT. What values of n, k, and p should be chosen to do this?

2. In this problem we will compute a coefficient form of a polynomial whose roots are $x_0, x_1, \ldots, x_{n-1}$ (with possible repetitions) in $O(n \log^2 n)$ time.

   (a) Design a recursive algorithm to compute a polynomial of degree-bound $n + 1$ whose roots are $x_0, x_1, \ldots, x_{n-1}$.

   (b) Give a recurrence describing the time bound of the algorithm and solve it using the master method described in Section 4.3 of CLRS.

3. We have all learned polynomial division as some point in our lives. The standard algorithm is $O(n^2)$ time where n is a degree-bound on the the polynomials. To be more specific, let $A(x)$ and $B(x)$ be two polynomials with complex coefficients, where $B(x) \neq 0$. There are polynomials $Q(x)$ and $R(x)$ such that $A(x) = Q(x)B(x) + R(x)$ where $\deg(R(x)) < \deg(B(x))$. The polynomial $Q(x)$ is the quotient and $R(x)$ is the remainder. We will write $Q(x)$ as $A(x)/B(x)$ and $R(x)$ as $A(x) \bmod B(x)$. The goal of this problem is to argue that polynomial division can be done in $O(n \log n)$ time. Clearly if we can compute the quotient $Q(x) = A(x)/B(x)$ then we can compute the remainder $R(x) = A(x) - Q(x)B(x)$ with one polynomial multiplication and a linear number of other operations. This our goal reduces to computing the quotient in $O(n \log n)$ time.

   (a) Suppose $B(x) = x^n$, then argue that division by $B(x)$ takes linear time.

   (b) Suppose $\deg(B(x)) = n-1$, then define RECIPROCAL$(B(x)) = x^{2n-2}/B(x)$. Show how to compute $A(x)/B(x)$ using the RECIPROCAL$(B(x))$, one polynomial multiplication, and a linear number of other operations.

   (c) Given the solution to (b) above, our goal reduces to computing RECIPROCAL$(B(x))$ in $O(n \log n)$ time. For simplicity, assume that n is a power of 2 and $\deg(B(x)) = n - 1$. We can express $B(x) = B_0(x) + x^{n/2}B_1(x)$ where $\deg(B_0(x)) \leq n/2 - 1$ and $\deg(B_1(x)) = n/2 - 1$. Define $S(x) =$ RECIPROCAL$(B_1(x))$. Define

$$U(x) = 2S(x)x^{(3/2)n-2} - S(x)^2B(x).$$

   Show that $U(x)B(x) = x^{3n-4} + T(x)$ where $\deg(T(x)) \leq 2n - 3$.

   (d) Use (c) above to give a recursive algorithm for computing RECIPROCAL$(B(x))$ when $\deg(B(x)) + 1$ is a power of 2. The time for your recursive algorithm should be described by the recurrence $T(n) \leq T(n/2) + cn \log n$, with $T(1)$ a constant. Explain why the recurrence is valid.

   (e) Use the master method (Section 4.3) of CLRS to solve the recurrence of part (d).