MATH 497, MATH 895, CMPT 894.
Assignment 1, Summer 2007

Instructor: Michael Monagan

Please hand in the assignment by 5pm on May 28th.
MATH 497 students should do questions 1 and 2 only.
Late Penalty -20% off for each day late.
For Maple problems, please submit a printout of a Maple worksheet containing Maple code and the execution of examples.

Question 1 Shönhage Strassen Integer Multiplication

(a) Let $p = 2^{2^r} + 1$, not necessarily prime. Let $n = 2^k$ such that $n | 2^{1+r}$. Let $\omega = 2^{(2^{1+r})/n}$. Prove that

 (i) ω is a primitive n'th root of unity in the integers modulo p and

 (ii) $\omega^j = -\omega^{j+n/2}$ in the integers modulo p,

and explicitly verify these for $r = 6$ and $k = 4$ in Maple.

(b) Let a and b be two positive integers satisfying $a < B^m$ and $b < B^m$ where B is the integer base, a constant. The Shönhage Strassen fast integer multiplication algorithm splits a and b into 2^l blocks where $2^{l-1} \leq \sqrt{m} < 2^l$. It writes $a = \sum_{i=0}^{2^l-1} a_i \beta^i$ and $b = \sum_{i=0}^{2^l-1} b_i \beta^i$ where β is a power of B. Thus a is a polynomial in β of degree approximately $\sqrt{m}$ with integer coefficients of length approximately $\sqrt{m}$ base B digits. It applies the FFT to multiply $a(x)$ and $b(x)$ modulo the first integer $p > 2^l \beta^2$ of the form $2^{2^r} + 1$ using $\omega = 2^s$ for an appropriate value of s.

Assuming that multiplications in the integers modulo p are done recursively by the same algorithm, determine the time complexity of the algorithm as a function of m, the length of a and b.

Question 2 The FFT and fast integer multiplication.

Implement the FFT, the forward transform, in Maple. See algorithm 4.4 in the Geddes text. Program it to take as input a list of integer coefficients $[a_0, a_1, ..., a_{n-1}]$ and to output a list of integers. To make your implementation efficient optimize it for $n = 2$.

Check that your implementation is correct by computing the Fourier transform of the following polynomial $f(x)$ using the prime $p = 7 \times 2^{20} + 1$, and then applying the inverse FFT to get back to $f(x)$. You will need a primitive $n = 64$'th root of unity.

```
> p := 7*2^20+1;
> f := Randpoly(50,x) mod p;
> a := [seq(coeff(f,x,i),i=0..50), 0$13];
```

Time your implementation on inputs of suitable degree d and check that the complexity of your implementation is $O(d \log d)$ and NOT $O(d^2)$.

Now design and implement an algorithm which uses the FFT to multiply two large integers a and b. First do this for $B = 2^3 2$ using three machine primes and the Chinese remainder theorem. Second, do this using Schönhage-Strassen algorithm which uses an integer modulus $p = 2^{2^l} + 1$. For the Sch"onhage Strassen algorithm, just use Maple to do arithmetic in the integers modulo p. That is, don't modify the FFT code to be recursive.

Test your algorithm on multiplying

```
> r := rand(2^(10^6)):
> a := r():
> b := r():
```

Do not focus on the efficiency of splitting up a large integer into blocks. Just use the following to write an integer a in base B.

```
Blocks := convert(a,base,B);
```

Question 3 The fast Euclidean algorithm.

Implement the fast HGCD procedure that on input of two integers $a \geq b > 0$ of length n digits outputs a matrix A such that $[c, d]^T := A[a, b]^T$ satisfies $\gcd(a, b) = \gcd(c, d)$ and the length of d is about half the length of a. Now write a procedure GCD that repeatedly calls HGCD to compute the gcd of a and b. Just try to get this to work. Don't worry too much about efficiency.

To obtain the leading half of the digits of a (and b) use

```
> n := ilog2(a); m := iquo(n+1,2);
> a1 := iquo(a,2^m);
> b1 := iquo(b,2^m);
```