# MATH 895 Assignment 1, Summer 2013

Instructor: Michael Monagan

Please hand in the assignment by 9:30am Thursday May 23rd.
Late Penalty -20% off for up to one day late. Zero after that.
For Maple problems, please submit a printout of a Maple worksheet containing Maple code and the execution of examples.

References: Sections 4.5–4.9 of Geddes, Czapor and Labahn and/or sections 8.2,8.3,9.1 of von zur Gathen and Gerhard.

## Question 1 Implementing the FFT.

Implement the FFT in $\mathbb{Z}_p$, the forward transform (Algorithm 4.4), in Maple. Program it to take as input a Maple list of integers for $a(x)$, i.e., $[a_0, a_1, ..., a_{m-1}, 0, ..., 0]$ for $a(x)$, and to output a list of integers. Alternatively you may use an Array. To make your implementation reasonably efficient optimize it for the case $n = 2$. If you want to precompute $\omega^i$ for $0 \le i < n/2$ and store them in an array, you may do so. This will save half the multiplications.

Check that your implementation is correct by computing the Fourier transform of the following polynomial $f(x)$ using the prime $p = 7 \times 2^{20} + 1$, then applying the inverse FFT to get back to $f(x)$.

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

You will need a primitive $n = 64$'th root of unity.
Use the Maple command `numtheory[primroot](p)`; to get a primitive element $\alpha$.

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

## Question 2 Integer multiplication using the FFT.

Implement the algorithm which uses the FFT to multiply two large integers $a$ and $b$ using three machine primes $p_1, p_2, p_3$ and then applying the Chinese remainder theorem. Do this using the base $B = 2^{32}$ and the primes $p_1 = 2^{27} \times 17 + 1$, $p_2 = 2^{27} \times 24 + 1$ and $p_3 = 2^{27} \times 26 + 1$. Test your algorithm on multiplying $a \times b$ where

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

To split up a large integer $A$ into blocks base $B$ use the Maple command `convert(A,base,B)`
To apply the Chinese remainder theorem use the Maple command `chrem([c1,c2,c3],[p1,p2,p3])`

## Question 3 Shönhage Strassen integer multiplication.

Implement the Schönhage-Strassen integer multiplication algorithm in Maple. It uses a large integer modulus of the form $p = 2^{2^r} + 1$. To divide by $p$ just use Maple so that you can reuse your FFT code from question 2 here. Also, use Maple for doing the multiplications in $C := [A_i \times B_i \bmod p, \text{ for } i = 1..n]$, i.e. don't make these multiplications recursive. Test your algorithm on the example in question 3.

## Question 4 Fast Division

Consider dividing $a$ by $b$ in $F[x]$ where $\deg a = d$, $\deg b = m$ with $d \geq m > 0$. Program the Newton iteration (Algorithm 4.6) recursively in Maple for $F = \mathbb{Z}_p$ to compute $f^{-1}$ as a power series to $O(x^n)$ where $n = d - m + 1$.

To make the Newton iteration efficient when $n$ is not a power of 2, compute $y = f^{-1}$ recursively to order $O(x^{\lceil n/2 \rceil})$. To truncate a polynomial $b$ modulo $x^n$ you could use the Maple command `rem(b,x^n,x)`. Use `convert(taylor(b,x,n),polynom)` instead.

Test your algorithm on the following problem in $\mathbb{Z}_p[x]$.

```
> p := 101;
> a := randpoly(x,degree=100,dense) mod p;
> b := randpoly(x,degree=50,dense) mod p;
> Quo(a,b,x) mod p;
```

## Question 5 Complexity of Fast Division

Let $f \in F[x]$ and let $D(n)$ be the number of multiplications in $F$ for computing $f^{-1}$ as power series to order $O(x^n)$ using the Newton iteration. Let $M(n)$ be the number of multiplications in $F$ that your favorite multiplication algorithm takes to multiply two polynomials of degree $n-1$ in $F[x]$. For $n = 2^k$ explain why

$$D(n) = D(n/2) + M(n) + M(n/2) + cn$$

for some constant $c > 0$. Now, using $D(1) = d$ for some constants $d > 0$, solve this recurrence relation, show that

$$D(n) < 3M(n) + 2cn + d.$$

Use the fact that $2M(n/2) < M(n)$, i.e., $M(n) > O(n)$.
Thus conclude that the cost of the Newton iteration is roughly 3 multiplications.