MATH 895 Assignment 1, Fall 2021

Instructor: Michael Monagan

Please hand in the assignment by 11pm Wednesday September 20th.
Late Penalty -20% off for up to 36 hours 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 sections 8.2,8.3,9.1 of von zur Gathen and Gerhard.

Question 1 : In-place FFT algorithms (15 marks).

Implement FFT1(n, A, ω, p, k) and FFT2(n, A, ω, p, k), the two FFT algorithms in Maple. Here A an Array of size n, ω has order n, p is a prime and k is an index k into A where you are working, i.e., if my C code accesses `A[i]` your Maple code will access `A[k+i]`. The first time you call FFT1 you will use $k = 0$. FFT1 and FFT2 should run "in-place", that is, the input polynomial is stored in the array A of size n and the output is in A and no temporary array is allocated.

Use FFT1 and FFT2 to compute $c(x) = a(x)b(x)$ modulo $p = 97$ where

$$a(x) = 1 + 39x + 57x^3 + 11x^4 + 19x^6 + x^8 \quad \text{and} \quad b(x) = 7 + 22x^3 + 44x^4 + 17x^6 + 9x^7.$$

Use ω of order $n = 16$. Now use FFT1 and FFT2 to multiply $a(x)$ and $b(x) + x^8$ using three FFTs using ω of order $n = 16$ not $n = 32$. You will need to do some additional operations to recover the product.

Question 2 : Analysis of the FFT (5 marks).

Let K be a field and ω be a primitive 4'th root of unity in K. Let $a = a_0 + a_1x + a_2x^2 + a_3x^3$ and $A = [a_0, a_1, a_2, a_3] \in K^4$. The FFT computes $F = [a(1), a(\omega), a(\omega^2), a(\omega^3)]^T$. This polynomial evaluation can be expressed as an affine transformation. Let V_4 be the 4×4 Vandermonde matrix

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

Then the FFT computes $V_4 A^T$, that is, $F = V_4 A^T$. For both FFT algorithms (FFT1 and FFT2) factor the matrix V_4 into a product of three matrices so that $V_4 = UVW$ where one of the matrices will be a permutation matrix. The two factorizations explain how the two algorithms both compute $V_4 A = F$. Check that the two permutation matrices are inverses of each other.

Question 3 : Fast Division (15 marks)

Consider computing the quotient of $a \div b$ in $F[x]$. To use the fast method we need to compute f^{-1} to $O(x^n)$ where $n = \deg a - \deg b + 1$ and $f = b^r$. Write a Maple procedure `FastNewton(f,x,n,p)` that computes f^{-1} to $O(x^n)$ for $F = \mathbb{Z}_p$ using a Newton iteration. Use `Expand(...) mod p;` for the polynomial multiplications so you get Maple's fast multiplication. 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 `rem(b,x^n,x)`. Use `convert(taylor(b,x,n),polynom)` instead which is more efficient.

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

```
> p := 11;
> f := 3+x+4*x^3+x^5;
> FastNewton(f,x,6,p);
```

$$10\,x^5 + 7\,x^4 + 10\,x^3 + 9\,x^2 + 6\,x + 4$$

Now write a Maple procedure `FastQuo(a,b,x,p)` to compute the quotient of $a \div b$ in $\mathbb{Z}_p[x]$ fast. Test your procedure on the following inputs

```
> p := 9973; d := 1000;
> while d < 10^5 do
>     a := Randpoly(degree=2*d-1,x) mod p;
>     b := Randpoly(degree=d,x) mod p;
>     q := CodeTools[Usage]( FastQuo(a,b,x,p) );
>     if q <> Quo(a,b,x) mod p then print(BUG); fi;
>     d := 2*d;
> od:
```

You will need to compute reciprocal polynomials efficiently. Let n be the degree of the f. To compute the f^r efficiently use `fr := expand( x^n*subs(x=1/x,f) );`

Question 4 : Complexity of Fast Division (5 marks)

Let $f \in F[x]$ and let $D(n)$ be the number of multiplications in F for computing a power series of f^{-1} 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

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

Assume that $2M(n/2) < M(n)$, that is, multiplication is super linear.
Conclude that the cost of the Newton iteration is equivalent to approx. 3 multiplications.