Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
H. Ghodselahi, Y. Maus October 29, 2014

Algorithm Theory, Winter Term 2014/15
Problem Set 2

hand in (hard copied) by Thursday, 10:00, November 06, 2014, either before the lecture or in the box corresponding to your group in building no. 51.

Exercise 1: Multiplication of Polynomials with FFT (7+2 points)

Given are the following two polynomials:

$$p(x) = x^3 + 2x^2 + 3x + 1,$$
$$q(x) = x + 1.000 \cdot 10^4$$

a) Compute $p(x)^2$ with the help of the FFT algorithm. Write down all intermediate results. To simplify notation and calculations use 8-th roots of unity.

Those unfamiliar with complex numbers should ask fellow students for some help - calculating roots of unity and multiplying 2 complex numbers is all you need for this exercise.

b) Compute $DFT^{-1}(DFT(q))$ and round all occurring numbers to 4 significant digits (in base 10).[1]

Exercise 2: Fast Potentiation of Polynomials(1+2 points)

The following algorithm computes x^{2^ℓ} for a real number x and $\ell \in \mathbb{N}$:

Algorithm 1: FastPotentiate(x, ℓ)

```
while ℓ > 0 do
    x := x · x;
    ℓ = ℓ − 1;
end
return x
```

Assuming that multiplication of floats can be done in $\mathcal{O}(1)$ time, algorithm FastPotentiate(x, ℓ) requires time $\mathcal{O}(\ell)$.

Now, let p a polynomial of degree n and $\ell \in \mathbb{N}$. We use the idea of the above algorithm to obtain two different algorithms to compute p^{2^ℓ}, which we state in pseudo code:

Algorithm 2: PolyPower1(p, ℓ)

```
set z to optimal value;
Compute (b_0, ..., b_{z−1}) := DFT(p);
for i := 0 to z − 1 do
    b_i := FastPotentiate(b_i, ℓ)
end
return DFT^{−1}(b)
```

[1] E.g. $1.0004 \cdot 10^4$ would be rounded to $1.000 \cdot 10^4$.

Algorithm 3: PolyPower2(p, ℓ)

```
while ℓ > 0 do
    set z to optimal value;
    (b_0, ..., b_z) := DFT(p);
    for i := 0 to z do
        b_i := b_i · b_i
    end
    p := DFT^{-1}(b);
    ℓ = ℓ − 1
end
return p
```

a) Determine the (optimal) value of z in PolyPower1. Which roots of unity are (optimally) needed for all (in Algorithms PolyPower1 and PolyPower2) invocations of the FFT algorithm?

b) Analyze the running time of both algorithms. (Assume that the time for multiplying two floats is in $\mathcal{O}(1)$ and the time to run the FFT algorithm is in $\mathcal{O}(n \cdot \log n)$ when n-th roots of unity are used.