Computer Algebra 2: homework for the winter holidays

Due date: 08 January 2019 at midnight (Linz time)

1 General instructions

This exercise sheet contains implementation exercises left as homework. It is quite long, so if necessary, pick the exercises that you find interesting and focus on doing those perfectly.

A LaTeXdocument may be attached to your submission with additional explanations, remarks and answers to questions.

Early submissions will receive early feedback, and resubmission is allowed (and encouraged).

All functions must be documented, see this page[1] for guidelines. At the very minimum, the documentation *must* include a short description of the function, its input(s) and one or several examples.

I strongly recommend that you use Sage[2] for the implementations, and even if you don't know or master it, this kind of exercises is a good way to get familiar with it. I will however not penalize it if a student does not know Sage and chooses to use another software instead. Please note that in that case, I will expect the code to be *excessively* documented to help my reading.

I am of course available for any questions that you may have during the realization of this work, including basic question on the computer algebra system.

2 Problems

Problems 2 and 5 are straightforward implementations of algorithms which were detailed in class. Problems 1 and 4 are implementations of algorithms which were described in class, but where some details may have been left out. Problem 3 is about describing new algorithms, based on the ones seen in class.

Whenever an algorithm requires computation over $\mathbb{F}_p$ for a prime p, you may use built-in functions for the arithmetic. *Those algorithms should work for all p.*

[1] https://tinyurl.com/sagedoc
[2] https://www.sagemath.org/

For example, the required function in Problem 1 should have the following specifications:

- Input: $f, g \in \mathbb{F}_p[X]$
- Output: $h = fg$

And calling the function like this (in Sage) should work:

```
sage: p = 65521
sage: R.<x> = PolynomialRing(GF(p))
sage: f = R.random_element(10)
sage: g = R.random_element(10)
sage: h = SchonhageStrassen(f,g)
sage: print(h)
...
```

Problem 1.

1. Implement Schönhage-Strassen's algorithm for multiplication in $\mathbb{F}_p[X]$, where p is a prime. *The function is not allowed to call the built-in polynomial multiplication routine.*
2. What is the largest degree N such that two polynomials of degree n can be multiplied in less than 10 minutes with your algorithm?
3. Measure the runtime of the algorithm on input of varying size (at least up to N), and plot the results. Verify that the complexity is indeed $O(n \log(n) \log(\log(n)))$, and give an experimental estimate for the constant. Run the same experiment with the built-in polynomial multiplication, how does it compare?

Problem 2. Let p be a prime. Let $\varphi : \mathbb{F}_p[X] \to \mathbb{F}_p[X]$ be a degree preserving ring morphism. Write a function `evaluate_transform` which takes as input φ and a polynomial f, and computes $\varphi(f)$. *The function is not allowed to call φ on any polynomial with positive degree. The function should make use of the algorithms described in Chapter 9.*

Problem 3.

1. Describe the integer equivalent of the algorithms of Chapter 9, for fast simultaneous modular reduction and fast Chinese remaindering.
2. Let $\varphi : \mathbb{Z} \to \mathbb{Z}$ be a ring morphism. Write a function `evaluate_map` which takes as input φ and an integer n, and computes $\varphi(n)$. *The function may use the fast built-in integer multiplication.*
3. The function of the previous question should not call φ on any large input, but "large" may be hard to define precisely. Let $F(n)$ be the complexity of a call of F on an input of *size* n. What is the theoretical complexity of the algorithm?
4. Verify, by measuring runtimes with large n, that the experimental complexity approaches the theoretical complexity. This experiment depends on the choice of φ, what kind of property can you give φ to make the results more significant?

Problem 4.

1. Implement Gram-Schmidt orthogonalization and the LLL algorithm, using calls to the Gram-Schmidt algorithm for steps 7 and 10. *The functions may use the built-in functions for fast integer arithmetic.*
2. Refine the implementation to perform steps 7 and 10 in quadratic time.
3. Pick one of the applications presented in class or in the notes, and implement it. *If necessary in order to reach interesting examples, you may use the built-in implementation of LLL.*

Problem 5. Let p be a prime. Implement Berlekamp's algorithm for factorization over $\mathbb{F}_p[X]$. *The function may use the built-in algorithms for polynomial arithmetic, in particular for computing gcd's.*