Introduction and Overview

CS-E4500 Advanced Course on Algorithms
Spring 2019

Petteri Kaski
Department of Computer Science
Aalto University

Please register to the course in Oodi

- What?
- Why?
- How?
- When and where?

What?

Spring 2019

Algorithms for Polynomials and Integers

Short synopsis of lectures (1/3)

- **Polynomials in one variable** are among the most elementary and most useful mathematical objects, with broad-ranging applications from *signal processing* to *error-correcting codes* and advanced applications such as *probabilistically checkable proofs* and *error-tolerant computation*
- One of the main reasons why polynomials are useful in a myriad of applications is that highly efficient algorithms are known for computing with polynomials
- These lectures introduce you to this near-linear-time toolbox and its select applications, with some algorithmic ideas dating back millennia, and some introduced only in the last few years

Short synopsis of lectures (2/3)

- By virtue of the **positional number system**, algorithms for computing with polynomials are closely related to algorithms for computing with **integers**
- In most cases, algorithms for polynomials are conceptually easier and thus form our principal object of study during our weekly lectures, with the corresponding algorithms for integers left for the exercises or for further study

Short synopsis of lectures (3/3)

- A tantalizing case where the connection between polynomials and integers apparently breaks down occurs with **factoring**
- Namely, it is known how to efficiently factor a given univariate polynomial over a finite field into its irreducible components, whereas no such algorithms are known for factoring a given integer into its prime factors
- Indeed, the best known algorithms for factoring integers run in time that scales moderately exponentially in the number of digits in the input
- These lectures introduce you both to efficient factoring algorithms for polynomials and to moderately exponential algorithms for factoring integers

Lecture schedule and more detailed synopsis

(tentative)

Lecture schedule

Tue 15 Jan:	1. Polynomials and integers
Tue 22 Jan:	2. The fast Fourier transform and fast multiplication
Tue 29 Jan:	3. Quotient and remainder
Tue 5 Feb:	4. Batch evaluation and interpolation
Tue 12 Feb:	5. Extended Euclidean algorithm and interpolation from erroneous data
Tue 19 Feb:	*Exam week — no lecture*
Tue 27 Feb:	6. Identity testing and probabilistically checkable proofs
Tue 5 Mar:	*Break — no lecture*
Tue 12 Mar:	7. Finite fields
Tue 19 Mar:	8. Factoring polynomials over finite fields
Tue 26 Mar:	9. Factoring integers

Lecture 1 (Tue 16 Jan): Polynomials and integers

- We start with elementary computational tasks involving polynomials, such as polynomial addition, multiplication, division (quotient and remainder), greatest common divisor, evaluation, and interpolation
- We observe that polynomials admit two natural representations: coefficient representation and evaluation representation
- We encounter the more-than-2000-year-old algorithm of Euclid for computing a greatest common divisor
- We observe the connection between polynomials in coefficient representation and integers represented in the positional number system

Lecture 2 (Tue 23 Jan): The fast Fourier transform and fast multiplication

- We derive one of the most fundamental and widely deployed algorithms in all of computing, namely the fast Fourier transform and its inverse
- We explore the consequences of this near-linear-time-computable duality between the coefficient and evaluation representations of a polynomial
- A key consequence is that we can multiply two polynomials in near-linear-time
- We obtain an algorithm for integer multiplication by reduction to polynomial multiplication

Lecture 3 (Tue 30 Jan): Quotient and remainder

- We continue the development of the fast polynomial toolbox with near-linear-time polynomial division (quotient and remainder)
- The methodological protagonist for this lecture is Newton iteration
- We explore Newton iteration and its convergence both in the continuous and in the discrete settings, including fast quotient and remainder over the integers

Lecture 4 (Tue 6 Feb):
Batch evaluation and interpolation

- We derive near-linear-time algorithms for batch evaluation and interpolation of polynomials using recursive remaindering along a subproduct tree
- In terms of methodological principles, we encounter algebraic divide-and-conquer, dynamic programming, and space-time tradeoffs
- As an application, we encounter secret sharing

Lecture 5 (Tue 20 Feb): Extended Euclidean algorithm and interpolation from erroneous data

- This lecture culminates our development of the near-linear-time toolbox for univariate polynomials
- First, we develop a divide-and-conquer version of the extended Euclidean algorithm for polynomials that recursively truncates the inputs to achieve near-linear running time
- Second, we present a near-linear-time polynomial interpolation algorithm that is robust to errors in the input data up to the information-theoretic maximum number of errors for correct recovery
- As an application, we encounter Reed–Solomon error-correcting codes together with near-linear-time encoding and decoding algorithms

Lecture 6 (Tue 27 Feb): Identity testing and probabilistically checkable proofs

- We investigate some further applications of the near-linear-time toolbox involving randomization in algorithm design and proof systems with probabilistic soundness
- We find that the elementary fact that a low-degree nonzero polynomial has only a small number of roots enables us to (probabilistically) verify the correctness of intricate computations substantially faster than running the computation from scratch
- Furthermore, we observe that proof preparation intrinsically tolerates errors by virtue of Reed–Solomon coding

Lecture 7 (Tue 6 Mar): Finite fields

- This lecture develops basic theory of finite fields to enable our subsequent treatment of factoring algorithms
- We recall finite fields of prime order, and extend to prime-power orders via irreducible polynomials
- We establish Fermat's little theorem for finite fields and its extension to products of monic irreducible polynomials
- We also revisit formal derivatives and taking roots of polynomials

Lecture 8 (Tue 13 Mar): Factoring polynomials over finite fields

- We develop an efficient factoring algorithm for univariate polynomials over a finite field by a sequence of reductions
- First, we reduce to square-free factorization via formal derivatives and greatest common divisors
- Then, we perform distinct-degree factorization of a square-free polynomial via the polynomial extension of Fermat's little theorem
- Finally, we split to equal-degree irreducible factors using probabilistic splitting polynomials

Lecture 9 (Tue 20 Mar): Factoring integers

- While efficient factoring algorithms are known for polynomials, for integers the situation is more tantalizing in the sense that no efficient algorithms for factoring are known
- This lecture looks at a selection of known algorithms with exponential and moderately exponential running times in the number of digits in the input
- We start with elementary trial division, proceed to look at an algorithm of Pollard and Strassen that makes use of fast polynomial evaluation and interpolation, and finally develop Dixon's random squares method as an example of a randomized algorithm with moderately exponential expected running time

Why?

Motivation (1/3)

- The toolbox of near-linear-time algorithms for univariate polynomials and large integers provides a practical showcase of recurrent mathematical ideas in algorithm design such as
  - linearity
  - duality
  - divide-and-conquer
  - dynamic programming
  - iteration and invariants
  - approximation
  - parameterization
  - tradeoffs between resources and objectives
  - randomization

Motivation (2/3)

- We gain exposure to a number of classical and recent applications, such as
  - secret-sharing
  - error-correcting codes
  - probabilistically checkable proofs
  - error-tolerant computation

Motivation (3/3)

- A tantalizing open problem in the study of computation is whether one can factor large integers efficiently
- We will explore select factoring algorithms both for univariate polynomials (over a finite field) and integers

Learning objectives (1/2)

- Terminology and objectives of modern algorithmics, including elements of algebraic, approximation, online, and randomised algorithms
- Ways of coping with uncertainty in computation, including error-correction and proofs of correctness
- The art of solving a large problem by reduction to one or more smaller instances of the same or a related problem
- (Linear) independence, dependence, and their abstractions as enablers of efficient algorithms

Learning objectives (2/2)

- Making use of duality
  - Often a problem has a corresponding **dual** problem that is obtainable from the original (the **primal**) problem by means of an easy transformation
  - The primal and dual control each other, enabling an algorithm designer to use the interplay between the two representations
- Relaxation and tradeoffs between objectives and resources as design tools
  - Instead of computing the exact optimum solution at considerable cost, often a less costly but principled approximation suffices
  - Instead of the complete dual, often only a randomly chosen partial dual or other relaxation suffices to arrive at a solution with high probability

Yet further motivation

- Gives a mathematical foundation towards current research done at Aalto CS (e.g., some of which was presented only recently at ALENEX'18 [14])
- Possibility to continue with
  - summer trainee work
  - MSc thesis work
  - doctoral studies
- Contact the lecturer for details

How?

Prerequisites

- Fundamentals of algorithm design and analysis
  (e.g. CS-E3190 Principles of Algorithmic Techniques)

- Mathematical maturity

Taking the course

- No exam
- Weekly problem sets award points, 4 problems / week
- The total number of points determines the course grade
- 9 weeks of activity

Weekly schedule

- **Lecture:**
  Tuesday 12–14, hall T5
  (best effort to publish each problem set concurrently with lectures)
- **Q & A session (review problem set & discuss):**
  Thursday 12–14, hall T5
  (participation recommended)
- **Deadline for submitting solutions to problem set:**
  Sunday 20:00 (8pm) Finnish time
- **Tutorial (model solutions):**
  Monday 16–18, hall T6

Exercises

- 4 problems each week [$= 4 \times 9 = 36$ graded problems total]
- Each solved problem awards up to 2 points
  (0 – failure, 1 – glorious attack, 2 – solved to near-perfection)
- Get help for solving the problems in the Q&A session
- The tutorial session [Mon after Sun deadline] is for discussing the model solutions & getting commentary on your solutions
- Code of conduct: You must solve the exercises yourself
- Late submissions are not possible

Grading [tentative]

- Total points earned from exercises determine the course grade:

• Grade 0 (=fail):	less than 40% max points
• Grade 1:	at least 40% max points
• Grade 2:	at least 50% max points
• Grade 3:	at least 60% max points
• Grade 4:	at least 70% max points
• Grade 5:	at least 80% max points

- [tentative = can relax grading from this]

Workload [5 ECTS]

Lectures	2 h
Q&A session	2 h
Tutorial session	2 h
Independent work	9 h
Total weekly workload	*15 h*
Total (9 weeks)	**135 h**

When and where?

CS-E4500 Advanced Course in Algorithms (5 ECTS, III–IV, Spring 2019)

2019 **KALENTERI** **2019**

Tammikuu		Helmikuu		Maaliskuu		Huhtikuu		Toukokuu		Kesäkuu	
1	Ti Uudenvuodenpäivä	1	Pe	1	Pe	1	Ma Vk 14 T9	1	Ke Vappu	1	La
2	Ke	2	La	2	La	2	Ti	2	To	2	Su
3	To	3	Su D3	3	Su	3	Ke	3	Pe	3	Ma Vk 23
4	Pe	4	Ma Vk 06 T3	4	Ma Vk 10	4	To	4	La	4	Ti
5	La	5	Ti L4	5	Ti Laskiainen	5	Pe	5	Su	5	Ke
6	Su Loppiainen	6	Ke	6	Ke Break	6	La	6	Ma Vk 19	6	To
7	Ma Vk 02	7	To Q4	7	To	7	Su	7	Ti	7	Pe
8	Ti	8	Pe	8	Pe	8	Ma Vk 15	8	Ke	8	La
9	Ke	9	La	9	La	9	Ti	9	To	9	Su Helluntaipäivä
10	To	10	Su D4	10	Su D6	10	Ke	10	Pe	10	Ma Vk 24
11	Pe	11	Ma Vk 07 T4	11	Ma Vk 11 T6	11	To	11	La	11	Ti
12	La	12	Ti L5	12	Ti L7	12	Pe	12	Su Äitienpäivä	12	Ke
13	Su	13	Ke	13	Ke	13	La	13	Ma Vk 20	13	To
14	Ma Vk 03	14	To Q5	14	To Q7	14	Su Palmusunnuntai	14	Ti	14	Pe
15	Ti L1	15	Pe	15	Pe	15	Ma Vk 16	15	Ke	15	La
16	Ke	16	La	16	La	16	Ti	16	To	16	Su
17	To Q1	17	Su	17	Su D7	17	Ke	17	Pe	17	Ma Vk 25
18	Pe	18	Ma Vk 08	18	Ma Vk 12 T7	18	To	18	La	18	Ti
19	La	19	Ti Exam	19	Ti L8	19	Pe Pitkäperjantai	19	Su Kaatuneiden muistopäivä	19	Ke
20	Su D1	20	Ke	20	Ke Kevätpäiväntasaus	20	La	20	Ma Vk 21	20	To
21	Ma Vk 04 T1	21	To week	21	To Q8	21	Su Pääsiäispäivä	21	Ti	21	Pe Kesäpäivänseisaus
22	Ti L2	22	Pe	22	Pe	22	Ma 2. pääsiäispäivä	22	Ke	22	La Juhannus
23	Ke	23	La	23	La	23	Ti	23	To	23	Su
24	To Q2	24	Su D5	24	Su D8	24	Ke	24	Pe	24	Ma Vk 26
25	Pe	25	Ma Vk 09 T5	25	Ma Vk 13 T8	25	To	25	La	25	Ti
26	La	26	Ti L6	26	Ti L9	26	Pe	26	Su	26	Ke
27	Su D2	27	Ke	27	Ke	27	La	27	Ma Vk 22	27	To
28	Ma Vk 05 T2	28	To Q6	28	To Q9	28	Su	28	Ti	28	Pe
29	Ti L3			29	Pe	29	Ma Vk 18	29	Ke	29	La
30	Ke			30	La	30	Ti	30	To Helatorstai	30	Su
31	To Q3			31	Su Kesäaika alkaa D9			31	Pe		

L = Lecture; hall T5, Tue 12–14
Q = Q & A session; hall T5, Thu 12–14
D = Problem set deadline; Sun 20:00
T = Tutorial (model solutions); hall T6, Mon 16–18

1. Polynomials and Integers

CS-E4500 Advanced Course on Algorithms
Spring 2019

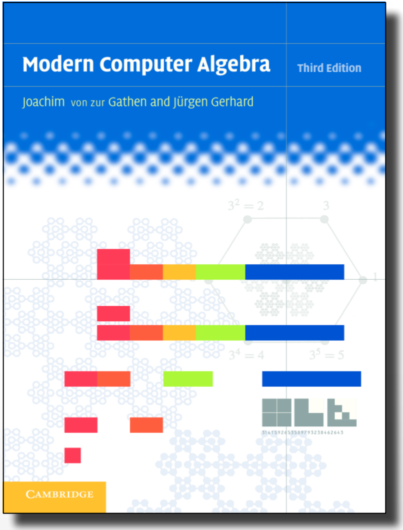
Petteri Kaski
Department of Computer Science
Aalto University

Key content for Lecture 1

- A boot camp of basic concepts and definitions in algebra
- Polynomials in one variable (univariate polynomials)
- Basic tasks and first algorithms for univariate polynomials
    - addition
    - multiplication
    - division (quotient and remainder)
    - evaluation
    - interpolation
    - greatest common divisor
- Evaluation–interpolation -duality of polynomials
- The (traditional) extended Euclidean algorithm and its analysis

A boot camp of basic concepts and definitions in algebra

(von zur Gathen and Gerhard [11],
Sections 2.2–3.2, 25.1–4)



Group

- A **group** is a nonempty set G with a binary operation $\cdot : G \times G \to G$ satisfying
  1. for all $a, b, c \in G$ we have $(a \cdot b) \cdot c = a \cdot (b \cdot c)$, (Associativity)
  2. there exists a $1 \in G$ such that $a \cdot 1 = 1 \cdot a = a$ for all $a \in G$, (Identity)
  3. for all $a \in G$ there exists an $a^{-1} \in G$ with $a \cdot a^{-1} = a^{-1} \cdot a = 1$ (Inverses)
- A group G is **commutative** if for all $a, b \in G$ we have $a \cdot b = b \cdot a$
- *Examples*:
  - $(\mathbb{Z}, +, 0)$ and $(\mathbb{Z}_m, +, 0)$ for $m \in \mathbb{Z}_{\geq 2}$ are commutative groups
  - $(\mathbb{Q} \setminus \{0\}, \cdot, 1)$ and $(\mathbb{Z}_m^\times, \cdot, 1)$ for $\mathbb{Z}_m^\times = \{1 \leq a < m : \gcd(a, m) = 1\}$ are commutative groups

(Commutative) ring

- A **ring** R is a set with two binary operations $+ : R \times R \to R$ and $\cdot : R \times R \to R$ satisfying
    1. R together with $+$ is a commutative group with identity 0,
    2. $\cdot$ is associative,
    3. R has an identity element 1 for $\cdot$,
    4. for all $a, b, c \in R$ we have $a(b + c) = (ab) + (ac)$ and $(b + c)a = (ba) + (ca)$
- A ring R is **commutative** if $\cdot$ is commutative
- A ring R is **nontrivial** if $0 \neq 1$
- Unless mentioned otherwise, in what follows we always assume that a ring R is both commutative and nontrivial
- *Examples*:
  $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}_m$ for $m \in \mathbb{Z}_{\geq 2}$

Example: $\mathbb{Z}_5$ (the integers modulo 5)

- One way to represent a (finite) ring is to give the addition and multiplication tables for the operations operations $+$ and $\cdot$
- In the two tables below, the entries at row x column y are $x+y$ and $x \cdot y$, respectively

$$\begin{array}{c|ccccc} + & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 2 & 3 & 4 \\ 1 & 1 & 2 & 3 & 4 & 0 \\ 2 & 2 & 3 & 4 & 0 & 1 \\ 3 & 3 & 4 & 0 & 1 & 2 \\ 4 & 4 & 0 & 1 & 2 & 3 \end{array} \qquad \begin{array}{c|ccccc} \cdot & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 2 & 3 & 4 \\ 2 & 0 & 2 & 4 & 1 & 3 \\ 3 & 0 & 3 & 1 & 4 & 2 \\ 4 & 0 & 4 & 3 & 2 & 1 \end{array} \tag{1}$$

Example: $\mathbb{Z}_6$ (the integers modulo 6)

- Below are the addition and multiplication tables for $\mathbb{Z}_6$

$$
\begin{array}{c|cccccc}
+ & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 1 & 2 & 3 & 4 & 5 \\
1 & 1 & 2 & 3 & 4 & 5 & 0 \\
2 & 2 & 3 & 4 & 5 & 0 & 1 \\
3 & 3 & 4 & 5 & 0 & 1 & 2 \\
4 & 4 & 5 & 0 & 1 & 2 & 3 \\
5 & 5 & 0 & 1 & 2 & 3 & 4
\end{array}
\qquad
\begin{array}{c|cccccc}
\cdot & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 2 & 3 & 4 & 5 \\
2 & 0 & 2 & 4 & 0 & 2 & 4 \\
3 & 0 & 3 & 0 & 3 & 0 & 3 \\
4 & 0 & 4 & 2 & 0 & 4 & 2 \\
5 & 0 & 5 & 4 & 3 & 2 & 1
\end{array}
\tag{2}
$$

- Compare the *multiplication* tables for $\mathbb{Z}_6$ (above) and $\mathbb{Z}_5$ (see (1)) — what qualitative differences can you spot?

Example: $\mathbb{Z}_{10}$ (the integers modulo 10)

- Here is a yet further example, the integers modulo 10

$$
\begin{array}{c|cccccccccc}
+ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \\
2 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 \\
3 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 \\
4 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 \\
5 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 \\
6 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 \\
7 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
8 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
9 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8
\end{array}
\qquad
\begin{array}{c|cccccccccc}
\cdot & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
2 & 0 & 2 & 4 & 6 & 8 & 0 & 2 & 4 & 6 & 8 \\
3 & 0 & 3 & 6 & 9 & 2 & 5 & 8 & 1 & 4 & 7 \\
4 & 0 & 4 & 8 & 2 & 6 & 0 & 4 & 8 & 2 & 6 \\
5 & 0 & 5 & 0 & 5 & 0 & 5 & 0 & 5 & 0 & 5 \\
6 & 0 & 6 & 2 & 8 & 4 & 0 & 6 & 2 & 8 & 4 \\
7 & 0 & 7 & 4 & 1 & 8 & 5 & 2 & 9 & 6 & 3 \\
8 & 0 & 8 & 6 & 4 & 2 & 0 & 8 & 6 & 4 & 2 \\
9 & 0 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1
\end{array}
\tag{3}
$$

- What patterns can you identify from the multiplication table?

Field, unit, associate

- A **unit** in a ring R is an element $u \in R$ for which there exists a multiplicative inverse $v \in R$ with $uv = 1$
- The set $R^{\times}$ of all units of R is a group under multiplication
- A ring R is a **field** if all nonzero elements of R are units
- *Examples:* (of fields)
  $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Z}_p$ for p prime
- We say that $a \in R$ is an **associate** of $b \in R$ and write $a \sim b$ if there exists a unit $u \in R$ such that $a = ub$
- $\sim$ is an equivalence relation on R

Examples / work points

- Study the multiplication table for $\mathbb{Z}_5$ in (1)
  — how can you identify which elements are units?
- Based on the units that you identify, conclude that $\mathbb{Z}_5$ is a field
- By studying the multiplication table for $\mathbb{Z}_6$ in (2), conclude that $\mathbb{Z}_6$ is *not* a field by identifying a nonzero element in $\mathbb{Z}_6$ that does not have a multiplicative inverse
- Study (2) and (3). Which elements are units in $\mathbb{Z}_6$? How about in $\mathbb{Z}_{10}$?
- Determine the equivalence classes for the associate relation $\sim$ in $\mathbb{Z}_5$, $\mathbb{Z}_6$, and $\mathbb{Z}_{10}$

Polynomials over a ring (1/2)

- Let R be a ring and let x be a formal indeterminate
- A **polynomial** $a \in R[x]$ in x over R is a finite sequence $(\alpha_0, \alpha_1, \ldots, \alpha_n)$ of elements of R (the **coefficients** of a) which we write as

$$a = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_{n-1} x^{n-1} + \alpha_n x^n = \sum_{i=0}^{n} \alpha_i x^i$$

- A polynomial a is **nonzero** if there exists a $j = 0, 1, \ldots, n$ with $\alpha_j \neq 0$
- For nonzero a, we assume that $\alpha_n \neq 0$ and say that $n = \deg a$ is the **degree** of a; the coefficient $\alpha_n = \mathrm{lc}(a)$ is the **leading coefficient** of a
- For zero a, it is convenient to assume that $a = (0)$ and set $\deg a = -\infty$
- A nonzero polynomial is **monic** if $\mathrm{lc}(a) = 1$

Polynomials over a ring (2/2)

- The set $R[x]$ equipped with the usual notions of addition and multiplication of polynomials (recalled in what follows) is a ring with additive identity (0) and multiplicative identity (1) for $0, 1 \in R$
- As a notational convention when working with polynomials, we use symbols x, y, z, w late in the Roman alphabet for formal indeterminates, and symbols $a, b, c, \ldots, s, t$ early in the Roman alphabet for polynomials
- We use symbols $\alpha, \beta, \gamma, \ldots, \omega$ in the Greek alphabet for elements in R

Complexity of an algorithm

- When studying algorithms that compute with given elements of $R[x]$, we adopt the convention of counting the number of **arithmetic operations** in R as a measure of the "running time" of an algorithm
- Arithmetic operations in R include addition, subtraction, multiplication and taking a multiplicative inverse (of a unit)
- We focus on **worst-case running time** (worst-case number of arithmetic operations in R) as a function of the degree(s) of the input polynomial(s) in $R[x]$
- We will work with asymptotic notation $O(\ )$ and $\tilde{O}(\ )$

Addition of polynomials

- Let $a = \sum_i \alpha_i x^i, b = \sum_i \beta_i x^i \in R[x]$ be given as input with $\deg a = n$ and $\deg b = m$
- The sum $c = a + b = \sum_i \gamma_i x^i \in R[x]$ is the polynomial with $\deg c \leq \max(n, m)$ defined for all $i = 0, 1, \ldots, \max(n, m)$ by
$$\gamma_i = \alpha_i + \beta_i \in R$$
- Given a, b as input, it is immediate that we can compute c in $O(\max(n, m))$ operations in R
- Subtraction and multiplication with a given element of R are defined analogously

Multiplication of polynomials

- Let $a = \sum_i \alpha_i x^i, b = \sum_i \beta_i x^i \in R[x]$ be given as input with $\deg a = n$ and $\deg b = m$
- The product $c = ab = \sum_i \gamma_i x^i \in R[x]$ is the polynomial with $\deg c \leq n + m$ defined for all $i = 0, 1, \ldots, n + m$ by

$$\gamma_i = \sum_{j=0}^{i} \alpha_j \beta_{i-j} \in R$$

- Given a, b as input, it is immediate that we can compute c in $O((n + m)^2)$ operations in R
- ... but could we do better? The output consists of only $O(n + m)$ elements of R ...

Polynomial division (quotient and remainder)

- Let $a = \sum_i \alpha_i x^i, b = \sum_i \beta_i x^i \in R[x]$ be given as input with $\deg a = n$, $\deg b = m$, $n \geq m \geq 0$, and suppose that $\beta_m \in R$ is a unit
- We want to compute $q, r \in R[x]$ with $a = qb + r$ and $\deg r < m$
- The classical division algorithm:
  1. $r \leftarrow a, \mu \leftarrow \beta_m^{-1}$
  2. **for** $i = n - m, n - m - 1, \ldots, 0$ **do**
  3. **if** $\deg r = m + i$ **then** $\eta_i \leftarrow \mathrm{lc}(r)\mu$, $r \leftarrow r - \eta_i x^i b$
     else $\eta_i \leftarrow 0$
  4. **return** $q = \sum_{i=0}^{n-m} \eta_i x^i$ and r
- We leave checking that $a = qb + r$ and $\deg r < m$ as an exercise; given a, b as input, it is immediate that we can compute q, r in $O((n + m)^2)$ operations in R
- ... but could we do better? The output consists of only $O(n + m)$ elements of R ...

Example (quotient and remainder)

- $a = x^4 + x^3 + x^2 + 1 \in \mathbb{Z}_2[x], \quad b = x^2 + 1 \in \mathbb{Z}_2[x]$
- $n = 4, m = 2$
- $\mu = \beta_m^{-1} = 1^{-1} = 1 \in \mathbb{Z}_2$
- Tracing the **for**-loop for $i = n - m, n - m - 1, \ldots, 0$, we have

i	η_i	r
		$x^4 + x^3 + x^2 + 1$
2	1	$x^3 + 1$
1	1	$x + 1$
0	0	$x + 1$

- $q = \eta_2 x^2 + \eta_1 x + \eta_0 = x^2 + x, \quad r = x + 1$

Evaluation (at a single point)

- Let $a = \sum_i \alpha_i x^i \in R[x]$ and $\xi \in R$ be given as input with $\deg a = n$
- We want to compute $a(\xi) = \sum_{i=0}^n \alpha_i \xi^i \in R$
- **Horner's rule**:

$$a(\xi) = (\cdots(((\alpha_n \xi + \alpha_{n-1})\xi + \alpha_{n-2})\xi + \alpha_{n-3})\xi + \cdots \alpha_1)\xi + \alpha_0$$

- Using Horner's rule, it takes $O(n)$ operations in R to compute $a(\xi)$

Batch evaluation (at m points)

- Let $a = \sum_i \alpha_i x^i \in R[x]$ and $\xi_1, \xi_2, \ldots, \xi_m \in R$ be given as input with $\deg a = n$
- We want to compute $a(\xi_1), a(\xi_2), \ldots, a(\xi_m) \in R$
- Repeated application of Horner's rule achieves this in $O(mn)$ operations in R
- ... but could we do better yet again? ...

Interpolation

- Let F be a field
- Let distinct $\xi_0, \xi_1, \ldots, \xi_n \in F$ and $\eta_0, \eta_1, \ldots, \eta_n \in F$ be given as input
- We want to compute the unique polynomial $f \in F[x]$ of degree at most n that satisfies

$$f(\xi_0) = \eta_0, \quad f(\xi_1) = \eta_1, \quad \ldots, \quad f(\xi_n) = \eta_n$$

- A classical algorithm (with complexity bounded by a polynomial in n) for this task will be studied in the exercises
- ... but could we do better yet again? ...

Integral domain

- An element $a \in R$ in a ring R is a **zero divisor** if there exists a nonzero $b \in R$ with $ab = 0$
- A ring D is an **integral domain** if there are no nonzero zero divisors
- *Examples*: (of integral domains)
  $\mathbb{Z}$, any field (exercise: units are not zero divisors), $F[x]$ for a field F
- *Work point*:
  Using (1), (2), and (3), determine all zero divisors in $\mathbb{Z}_5$, $\mathbb{Z}_6$, and $\mathbb{Z}_{10}$, respectively

Greatest common divisor

- Let R be a ring and let $a, b \in R$
- We say that a **divides** b and write $a|b$ if there exists a $q \in R$ with $aq = b$
- For $a, b, c \in R$ we say that c is a **greatest common divisor** (or gcd) of a and b if
  1. $c|a$ and $c|b$,
  2. for all $d \in R$ if $d|a$ and $d|b$, then $d|c$
- A greatest common divisor need not exist, and need not be unique
- In an integral domain, any two greatest common divisors are associates

Euclidean domain

- An integral domain E together with a function $d : E \to \mathbb{Z}_{\geq 0} \cup \{-\infty\}$ is a **Euclidean domain** if for all $a, b \in E$ with $b \neq 0$ there exist $q, r \in E$ with $a = qb + r$ and $d(r) < d(b)$
- We say that $q = a \operatorname{quo} b$ is a **quotient** and $r = a \operatorname{rem} b$ a **remainder** in the division of a by b
- We assume that we have available as a subroutine a **division algorithm** that for given $a, b \in E$ with $b \neq 0$ computes $q, r \in E$ with $a = qb + r$ and $d(r) < d(b)$
- *Examples:* (of Euclidean domains)
  - $\mathbb{Z}$ with $d(a) = |a| \in \mathbb{Z}_{\geq 0}$
    - Quotient and remainder can be determined with a division algorithm for integers
  - $F[x]$ for a field F with $d(a) = \deg a$
    - Quotient and remainder can be determined with a division algorithm for polynomials

Traditional Euclidean algorithm

- Let E be an Euclidean domain
- Let $f, g \in E$ be given as input
- We seek to compute a greatest common divisor of f and g
  - Since E is an integral domain, any two greatest common divisors of f and g are related to each other by multiplication with a unit
  - The Euclidean algorithm both (a) shows that greatest common divisors *exist* and (b) gives a way of *computing* a greatest common divisor by iterative remainders
- Traditional Euclidean algorithm:
  1. $r_0 \leftarrow f$, $r_1 \leftarrow g$
  2. $i \leftarrow 1$,
     while $r_i \neq 0$ **do** $r_{i+1} \leftarrow r_{i-1} \text{ rem } r_i$, $i \leftarrow i + 1$
  3. **return** r_{i-1} (a greatest common divisor)
- *Why does this algorithm always stop?* (Hint: $d(r_{i+1}) < d(r_i)$)

Traditional extended Euclidean algorithm

- Let $f, g \in E$ be given as input from an Euclidean domain E
- Traditional extended Euclidean algorithm:
  1. $r_0 \leftarrow f$, $s_0 \leftarrow 1$, $t_0 \leftarrow 0$,
     $r_1 \leftarrow g$, $s_1 \leftarrow 0$, $t_1 \leftarrow 1$
  2. $i \leftarrow 1$,
     while $r_i \neq 0$ **do**
     $q_i \leftarrow r_{i-1} \text{ quo } r_i$
     $r_{i+1} \leftarrow r_{i-1} - q_i r_i$
     $s_{i+1} \leftarrow s_{i-1} - q_i s_i$
     $t_{i+1} \leftarrow t_{i-1} - q_i t_i$
     $i \leftarrow i + 1$
  3. $\ell \leftarrow i - 1$
     return ℓ, r_i, s_i, t_i for $i = 0, 1, \ldots, \ell + 1$, and q_i for $i = 1, 2, \ldots, \ell$

Example (over $\mathbb{Z}$)

- Let $f = 1234 \in \mathbb{Z}$ and $g = 12 \in \mathbb{Z}$
- We obtain

i	r_i	s_i	t_i	q_i
0	1234	1	0	
1	12	0	1	102
2	10	1	-102	1
3	2	-1	103	5
4	0	6	-617	

- In particular $\ell = 3$ and $r_\ell = 2$ is a greatest common divisor of 1234 and 12

Example (over $\mathbb{Z}_2[x]$)

- Let $f = x^5 + x^4 + x^3 + x^2 + x + 1 \in \mathbb{Z}_2[x]$ and $g = x^5 + x^4 + 1 \in \mathbb{Z}_2[x]$
- We obtain

i	r_i	s_i	t_i	q_i
0	$x^5 + x^4 + x^3 + x^2 + x + 1$	1	0	
1	$x^5 + x^4 + 1$	0	1	1
2	$x^3 + x^2 + x$	1	1	$x^2 + 1$
3	$x^2 + x + 1$	$x^2 + 1$	x^2	x
4	0	$x^3 + x + 1$	$x^3 + 1$	

- In particular $\ell = 3$ and $r_\ell = x^2 + x + 1$ is a greatest common divisor of $x^5 + x^4 + x^3 + x^2 + x + 1$ and $x^5 + x^4 + 1$

Analysis using invariants (in this week's problem set)

- Suppose on input $f, g \in E$ we obtain the output ℓ, r_i, s_i, t_i for $i = 0, 1, \ldots, \ell + 1$, and q_i for $i = 1, 2, \ldots, \ell$
- Introduce the matrices
$$R_0 = \begin{bmatrix} s_0 & t_0 \\ s_1 & t_1 \end{bmatrix} \in E^{2\times 2}, \qquad Q_i = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \in E^{2\times 2} \quad \text{for } i = 1, 2, \ldots, \ell,$$
and $R_i = Q_i Q_{i-1} \cdots Q_1 R_0 \in E^{2\times 2}$ for $i = 0, 1, \ldots, \ell$
- The following invariants hold for all $i = 0, 1, \ldots, \ell$:
  1. $R_i \begin{bmatrix} f \\ g \end{bmatrix} = \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix}$.
  2. $R_i = \begin{bmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{bmatrix}$.
  3. r_ℓ is a greatest common divisor of r_i and r_{i+1}.
  4. $s_i f + t_i g = r_i$.

Recap of key content in Lecture 1

- A boot camp of basic concepts and definitions in algebra
- Polynomials in one variable (univariate polynomials)
- Basic tasks and first algorithms for univariate polynomials
  - addition
  - multiplication
  - division (quotient and remainder)
  - evaluation
  - interpolation (exercise)
  - greatest common divisor
- Evaluation–interpolation -duality of polynomials (exercise)
- Analysis of the extended Euclidean algorithm via invariants (exercise)

What next?

- Register to the course in Oodi if you have not already done so
  (or e-mail the lecturer in case you missed the registration period)
- Problem Set 1 available in MyCourses
- Q&A session on Thursday (12–14 hall T5)
- Problem Set 1 deadline Sun 20 Jan 20:00, Finnish time
  (submit a single PDF file — submission instructions in problem sheet)

Addendum (1/2)

- To get a hands-on perspective to the concepts and algorithm designs, it is in most cases useful to do some quick-and-dirty programming using your own favorite programming language and/or computer algebra system
- E.g. the lecturer often uses the Scala programming language for drafting out concepts and designs

  `https://www.scala-lang.org`
- Here is a `git` repository that contains a quick-and-dirty, first-draft Scala implementation (with very limited documentation) of selected concepts in this lecture:

  `https://github.com/pkaski/cs-e4500-2018.git`

Addendum (2/2)

- Computer algebra systems that you may want to try out include
  - Mathematica (https://download.aalto.fi/index-en.html)
  - GAP (https://www.gap-system.org)
  - Magma (http://magma.maths.usyd.edu.au/magma/)
  - Sage (http://www.sagemath.org)
  - ...