5. Extended Euclidean algorithm

CS-E4500 Advanced Course on Algorithms
Spring 2020

Petteri Kaski
Department of Computer Science
Aalto University

Lecture schedule

Tue 7 Jan:	1. Polynomials and integers
Tue 14 Jan:	2. The fast Fourier transform and fast multiplication
Tue 21 Jan:	3. Quotient and remainder
Tue 28 Jan:	4. Batch evaluation and interpolation
Tue 4 Feb:	5. Extended Euclidean algorithm
Tue 11 Feb:	*Break — no lecture*
Tue 18 Feb:	*Exam week — no lecture*
Tue 26 Feb:	6. Interpolation from erroneous data
Tue 3 Mar:	7. Factor graphs and contraction
Tue 10 Mar:	8. Verifiable and error-tolerant inference for factor graphs
Tue 17 Mar:	9. Further applications of factor graphs

CS-E4500 Advanced Course in Algorithms (5 ECTS, III–IV, Spring 2020)

2020 KALENTERI 2020

Tammikuu

1	Ke	Uudenvuodenpäivä
2	To	
3	Pe	
4	La	
5	Su	
6	Ma	Loppiainen
7	Ti	L1
8	Ke	
9	To	Q1
10	Pe	
11	La	
12	Su	D1
13	Ma	Vk 03 T1
14	Ti	L2
15	Ke	
16	To	Q2
17	Pe	
18	La	
19	Su	D2
20	Ma	Vk 04 T2
21	Ti	L3
22	Ke	
23	To	Q3
24	Pe	
25	La	
26	Su	D3
27	Ma	Vk 05 T3
28	Ti	L4
29	Ke	
30	To	Q4
31	Pe	

Helmikuu

1	La	
2	Su	D4
3	Ma	Vk 06 T4
4	Ti	L5
5	Ke	
6	To	Q5
7	Pe	
8	La	
9	Su	
10	Ma	Vk 07
11	Ti	Break
12	Ke	week
13	To	
14	Pe	
15	La	
16	Su	
17	Ma	Vk 08
18	Ti	Exam
19	Ke	week
20	To	
21	Pe	
22	La	
23	Su	D5
24	Ma	Vk 09 T5
25	Ti	Laskiainen
26	Ke	
27	To	Q6
28	Pe	
29	La	

Maaliskuu

1	Su	D6
2	Ma	Vk 10 T6
3	Ti	L7
4	Ke	
5	To	Q7
6	Pe	
7	La	
8	Su	D7
9	Ma	Vk 11 T7
10	Ti	L8
11	Ke	
12	To	Q8
13	Pe	
14	La	
15	Su	D8
16	Ma	Vk 12 T8
17	Ti	L9
18	Ke	
19	To	Q9
20	Pe	Kevätpäiväntasaus
21	La	
22	Su	D9
23	Ma	Vk 13 T9
24	Ti	
25	Ke	
26	To	
27	Pe	
28	La	
29	Su	Kesäaika alkaa
30	Ma	Vk 14
31	Ti	

Huhtikuu

1	Ke	
2	To	
3	Pe	
4	La	
5	Su	Palmusunnuntai
6	Ma	Vk 15
7	Ti	
8	Ke	
9	To	
10	Pe	Pitkäperjantai
11	La	
12	Su	Pääsiäispäivä
13	Ma	2. pääsiäispäivä
14	Ti	
15	Ke	
16	To	
17	Pe	
18	La	
19	Su	
20	Ma	Vk 17
21	Ti	
22	Ke	
23	To	
24	Pe	
25	La	
26	Su	
27	Ma	Vk 18
28	Ti	
29	Ke	
30	To	

Toukokuu

1	Pe	Vappu
2	La	
3	Su	
4	Ma	Vk 19
5	Ti	
6	Ke	
7	To	
8	Pe	
9	La	
10	Su	Äitienpäivä
11	Ma	Vk 20
12	Ti	
13	Ke	
14	To	
15	Pe	
16	La	
17	Su	Kaatuneiden muistopäivä
18	Ma	Vk 21
19	Ti	
20	Ke	
21	To	Helatorstai
22	Pe	
23	La	
24	Su	
25	Ma	Vk 22
26	Ti	
27	Ke	
28	To	
29	Pe	
30	La	
31	Su	Helluntaipäivä

Kesäkuu

1	Ma	Vk 23
2	Ti	
3	Ke	
4	To	
5	Pe	
6	La	
7	Su	
8	Ma	Vk 24
9	Ti	
10	Ke	
11	To	
12	Pe	
13	La	
14	Su	
15	Ma	Vk 25
16	Ti	
17	Ke	
18	To	
19	Pe	
20	La	Juhannus
21	Su	Kesäpäivänseisaus
22	Ma	Vk 26
23	Ti	
24	Ke	
25	To	
26	Pe	
27	La	
28	Su	
29	Ma	Vk 27
30	Ti	

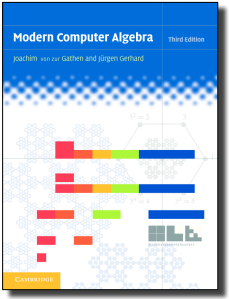
L = Lecture; hall T5, Tue 12–14
Q = Q & A session; hall T5, Thu 12–14
D = Problem set deadline; Sun 20:00
T = Tutorial (model solutions); hall T6, Mon 16–18

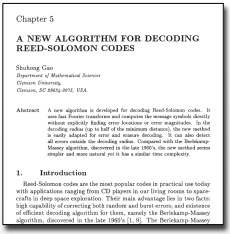
Recap of last week

- Fast **batch evaluation** and **interpolation** of polynomials
- Reduction to fast quotient and remainder
  —divide-and-conquer recursive remaindering along a **subproduct tree**
- **Secret sharing** by randomization

Goal: Near-linear-time toolbox for univariate polynomials

- Multiplication
- Division (quotient and remainder)
- Batch evaluation
- Interpolation
- Extended Euclidean algorithm (gcd) (this week)
- Interpolation from partly erroneous data





Chapter 5

A NEW ALGORITHM FOR DECODING REED-SOLOMON CODES

Shuhong Gao
Department of Mathematical Sciences
Clemson University,
Clemson, SC 29634-1907, USA.

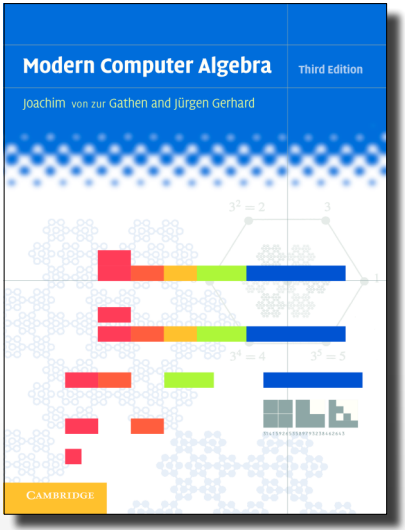
Abstract: A new algorithm is developed for decoding Reed-Solomon codes. It uses fast Fourier transforms and computes the message symbols directly without explicitly finding error locations or error magnitudes. In the decoding radius (up to half of the minimum distance), the new method is easily adapted for error and erasure decoding. It can also detect all errors outside the decoding radius. Compared with the Berlekamp-Massey algorithm, discovered in the late 1960's, the new method seems simpler and more natural yet it has a similar time complexity.

1. Introduction

Reed-Solomon codes are the most popular codes in practical use today with applications ranging from CD players in our living rooms to spacecrafts in deep space exploration. Their main advantage lies in two facts: high capability of correcting both random and burst errors; and existence of efficient decoding algorithm for them, namely the Berlekamp-Massey algorithm, discovered in the late 1960's [1, 9]. The Berlekamp-Massey

Fast extended Euclidean algorithm (for polynomials)

(von zur Gathen and Gerhard [10], Section 11.1)



Fast extended Euclidean algorithm (for integers)

(Möller [17])

MATHEMATICS OF COMPUTATION
Volume 77, Number 261, January 2008, Pages 589–607
S 0025-5718(07)02017-0
Article electronically published on September 12, 2007

ON SCHÖNHAGE'S ALGORITHM AND SUBQUADRATIC INTEGER GCD COMPUTATION

NIELS MÖLLER

Abstract. We describe a new subquadratic left-to-right GCD algorithm, inspired by Schönhage's algorithm for reduction of binary quadratic forms, and compare it to the first subquadratic GCD algorithm discovered by Knuth and Schönhage, and to the binary recursive GCD algorithm of Stehlé and Zimmermann. The new GCD algorithm runs slightly faster than earlier algorithms, and it is much simpler to implement. The key idea is to use a stop condition for HGCD that is based not on the size of the remainders, but on the size of the next difference. This subtle change is sufficient to eliminate the back-up steps that are necessary in all previous subquadratic left-to-right GCD algorithms. The subquadratic GCD algorithms all have the same asymptotic running time, $O(n(\log n)^2 \log\log n)$.

1. Introduction

In this paper, we describe four subquadratic GCD algorithms: Schönhage's algorithm from 1971, Stehlé's and Zimmermann's binary recursive GCD, a hitherto unpublished GCD algorithm discovered by Schönhage in 1987, and a novel GCD algorithm that uses similar ideas in a HGCD framework. The algorithms are compared with respect to running time and implementation complexity. The new algorithm is slightly faster than all the earlier algorithms, and much simpler to implement.

The paper is organized as follows: First we review the development of integer GCD algorithms in recent years. Section 2 describes the general structure and flavor of the subquadratic GCD algorithms, the idea of using a half-GCD function, and the resulting asymptotic running time. In Section 3, we briefly describe one variant of Schönhage's 1971 algorithm, and in Section 4, we describe the binary recursive GCD algorithm. The objective of these two sections is to provide sufficient details so that the new algorithm can be compared to earlier algorithms; we define the corresponding half-GCD functions, but we don't provide correctness proofs or detailed analysis.

Section 5 describes a GCD algorithm modeled on Schönhage's algorithm for reduction of binary quadratic forms [8], and in Section 6, this algorithm is reorganized into half-GCD form, resulting in a novel GCD algorithm. Section 7 describes the implementation of the different GCD algorithms, their running times, and code complexity.

Received by the editor November 19, 2005.
2000 *Mathematics Subject Classification.* Primary 11A05, 11Y16.

Key content for Lecture 5

- **Extended Euclidean algorithm** for polynomials recalled and expanded
  - The **quotient sequence**, the **Bézout coefficients**, and the **halting threshold**
- Fast extended Euclidean algorithm for polynomials by **divide and conquer**
  - The two polynomial operands **truncated** to a prefix of the highest-degree monomials determine the prefix of the quotient sequence (exercise)

Extended Euclidean algorithm (for polynomials)

- Let F be a field and let $f, g \in F[x]$ with $\deg f \geq \deg g \geq 0$
- 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**
     $q_i \leftarrow r_{i-1} \operatorname{quo} r_i$
     $r_{i+1} \leftarrow r_{i-1} - q_i r_i$
     $s_{i+1} \leftarrow s_{i-1} - q_i s_i$
     $t_{i+1} \leftarrow t_{i-1} - q_i t_i$
     $i \leftarrow i + 1$
  3. $\ell \leftarrow i - 1$
     return ℓ, r_i, s_i, t_i for $i = 0, 1, \ldots, \ell + 1$, and q_i for $i = 1, 2, \ldots, \ell$
- We want a faster algorithm

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 $x^5 + x^4 + 1$

Terminology

- The sequence $q_1, q_2, \ldots, q_\ell$ is the **quotient sequence** produced by the algorithm
- The polynomial r_i is the **remainder** at iteration i
- The polynomials s_i and t_i are the **Bézout coefficients** at iteration i
- The Bézout coefficients satisfy $r_i = s_i r_0 + t_i r_1$

Desiderata for a fast algorithm

- Let F be a field and let $f, g \in F[x]$ with $d \geq \deg f \geq \deg g \geq 0$
- Desired output:
  The quotients $q_1, q_2, \ldots, q_h$ and two consecutive rows r_h, s_h, t_h and $r_{h+1}, s_{h+1}, t_{h+1}$ for a choice of $h = 1, 2, \ldots, \ell$
- Using $O(M(d) \log d)$ operations in F

The degree sequences m_i and n_i

- It will be convenient to work with the following two sequences
- For $i = 1, 2, \ldots, \ell + 1$ let

$$m_i = \deg q_i$$

  where, for convenience, we let $m_{\ell+1} = \infty$
- For $i = 0, 1, \ldots, \ell + 1$, let

$$n_i = \deg r_i$$

  recalling that $n_{\ell+1} = \deg 0 = -\infty$
- By assumption, we have $\deg r_0 \geq \deg r_1 \geq 0$
- Since we have $r_{i+1} = r_{i-1} - q_i r_i$ and $\deg r_i > \deg r_{i+1}$ for all $i = 1, 2, \ldots, \ell$, it follows that

$$n_{i-1} = n_i + m_i$$

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	m_i	n_i
0	$x^5 + x^4 + x^3 + x^2 + x + 1$	1	0			5
1	$x^5 + x^4 + 1$	0	1	1	0	5
2	$x^3 + x^2 + x$	1	1	$x^2 + 1$	2	3
3	$x^2 + x + 1$	$x^2 + 1$	x^2	x	1	2
4	0	$x^3 + x + 1$	$x^3 + 1$		∞	$-\infty$

- 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 $x^5 + x^4 + 1$

The halting threshold $h = h(k)$

- Given a threshold parameter $k = 0, 1, \ldots, n_0$ as input, we want the algorithm to halt at iteration $h = h(k)$ determined by

$$m_1 + m_2 + \ldots + m_h \leq k$$

and

$$m_1 + m_2 + \ldots + m_h + m_{h+1} > k$$

- In particular, we observe that $0 \leq h \leq \ell$
- Let us also define $h(k) = 0$ for $k < 0$ and $h(k) = \ell$ for $k > n_0$

The halting threshold $h = h(k)$

- Equivalently, since $n_i = n_{i-1} - m_i$ for $i = 1, 2, \ldots, \ell + 1$, we have

$$n_h \geq n_0 - k$$

and

$$n_{h+1} < n_0 - k$$

- That is, the algorithm halts at the unique iteration $h = 0, 1, \ldots, \ell$ when the degree of r_{h+1} for the first time decreases below $n_0 - k$

Truncating a polynomial

- Let
$$f = \varphi_n x^n + \varphi_{n-1} x^{n-1} + \ldots + \varphi_1 x + \varphi_0 \in F[x]$$
with **leading coefficient** $\operatorname{lc} f = \varphi_n \neq 0$
- For $k \in \mathbb{Z}$, define the **truncated polynomial**
$$f \restriction k = \varphi_n x^k + \varphi_{n-1} x^{k-1} + \ldots + \varphi_{n-k+1} x + \varphi_{n-k} \in F[x]$$
where we set $\varphi_i = 0$ for $i < 0$ as necessary
- For $k \geq 0$ we have that $f \restriction k$ is a polynomial of degree k whose coefficients are the $k+1$ highest coefficients of f
- For $k < 0$ we have $f \restriction k = 0$
- For all $i = 0, 1, \ldots$ we have $(f x^i) \restriction k = f \restriction k$

Example: Truncating a polynomial

- Let us work with the polynomial

$$f = 2 + 9x + 10x^2 + 4x^3 \in \mathbb{Z}_{11}[x]$$

- We obtain the truncations

$$\begin{aligned}
&\vdots \\
f\restriction -2 &= 0 \\
f\restriction -1 &= 0 \\
f\restriction 0 &= 4 \\
f\restriction 1 &= 10 + 4x \\
f\restriction 2 &= 9 + 10x + 4x^2 \\
f\restriction 3 &= 2 + 9x + 10x^2 + 4x^3 \\
f\restriction 4 &= 2x + 9x^2 + 10x^3 + 4x^4 \\
f\restriction 5 &= 2x^2 + 9x^3 + 10x^4 + 4x^5 \\
&\vdots
\end{aligned}$$

Coinciding pairs of polynomials

- Let $f, g, \tilde{f}, \tilde{g} \in F[x] \setminus \{0\}$ with $\deg f \geq \deg g$ and $\deg \tilde{f} \geq \deg \tilde{g}$
- For $k \in \mathbb{Z}$, we say that (f, g) and $(\tilde{f}, \tilde{g})$ **coincide up to** k and write $(f, g) \equiv_k (\tilde{f}, \tilde{g})$ if

$$f \restriction k = \tilde{f} \restriction k$$
$$g \restriction (k - (\deg f - \deg g)) = \tilde{g} \restriction (k - (\deg \tilde{f} - \deg \tilde{g}))$$

- Exercise:
  If $(f, g) \equiv_k (\tilde{f}, \tilde{g})$ and $k \geq \deg f - \deg g$, then $\deg f - \deg g = \deg \tilde{f} - \deg \tilde{g}$

Example: Coinciding pairs of polynomials

- The pairs

$$f = 7 + 2x + x^2 + x^3 + 10x^4 + 7x^5 + x^6 + 5x^7 + 9x^8 + 5x^9 + 7x^{10} \in \mathbb{Z}_{11}[x]$$
$$g = 3 + 7x + 4x^2 + 2x^3 + 2x^4 + 6x^5 + 3x^6 + 2x^7 + 4x^8 \in \mathbb{Z}_{11}[x]$$

  and

$$\tilde{f} = 1 + 5x + 9x^2 + 5x^3 + 7x^4 \in \mathbb{Z}_{11}[x]$$
$$\tilde{g} = 3 + 2x + 4x^2 \in \mathbb{Z}_{11}[x]$$

  coincide up to 4

- Indeed, we have $\deg f = 10$, $\deg g = 8$, $\deg \tilde{f} = 4$, and $\deg \tilde{g} = 2$, with

$$f \restriction 4 = \tilde{f} \restriction 4 = 1 + 5x + 9x^2 + 5x^3 + 7x^4$$
$$g \restriction 2 = \tilde{g} \restriction 2 = 3 + 2x + 4x^2$$

Quotients of coinciding pairs of polynomials

- The following lemma enables us to design a divide-and-conquer extended Euclidean algorithm by truncating the operands to division

Lemma 8 (Sufficiently coinciding pairs of polynomials have identical quotients)
Suppose that $(f, g) \equiv_{2k} (\tilde{f}, \tilde{g})$ *for* $k \in \mathbb{Z}$ *with* $k \geq \deg f - \deg g \geq 0$. *Define* $q, r, \tilde{q}, \tilde{r} \in F[x]$ *by division with quotients and remainders as follows*

$$f = qg + r, \qquad \deg r < \deg g,$$
$$\tilde{f} = \tilde{q}\tilde{g} + \tilde{r}, \qquad \deg \tilde{r} < \deg \tilde{g}.$$

Then, $q = \tilde{q}$ *and at least one of the following holds* $(g, r) \equiv_{2(k-\deg q)} (\tilde{g}, \tilde{r})$ *or* $r = 0$ *or* $k - \deg q < \deg g - \deg r$.

Proof.
Exercise

□

Example: Quotient of coinciding pairs of polynomials

- The pairs

$$f = 7 + 2x + x^2 + x^3 + 10x^4 + 7x^5 + x^6 + 5x^7 + 9x^8 + 5x^9 + 7x^{10} \in \mathbb{Z}_{11}[x]$$
$$g = 3 + 7x + 4x^2 + 2x^3 + 2x^4 + 6x^5 + 3x^6 + 2x^7 + 4x^8 \in \mathbb{Z}_{11}[x]$$

  and

$$\tilde{f} = 1 + 5x + 9x^2 + 5x^3 + 7x^4 \in \mathbb{Z}_{11}[x]$$
$$\tilde{g} = 3 + 2x + 4x^2 \in \mathbb{Z}_{11}[x]$$

  coincide up to 4, with $4 \geq \deg f - \deg g = 2$

- Accordingly (by Lemma 8), the quotients agree:

$$f \operatorname{quo} g = 9 + 10x + 10x^2$$
$$\tilde{f} \operatorname{quo} \tilde{g} = 9 + 10x + 10x^2$$

Quotient sequences of coinciding pairs of polynomials

- Now let us study what happens in the extended Euclidean algorithm if we execute it for two inputs, (r_0, r_1) and $(\tilde{r}_0, \tilde{r}_1)$, with $\deg r_0 \geq \deg r_1 \geq 0$ and $\deg \tilde{r}_0 \geq \deg \tilde{r}_1 \geq 0$:

$$
\begin{aligned}
r_0 &= q_1 r_1 + r_2, & \tilde{r}_0 &= \tilde{q}_1 \tilde{r}_1 + \tilde{r}_2 \\
r_1 &= q_2 r_2 + r_3, & \tilde{r}_1 &= \tilde{q}_2 \tilde{r}_2 + \tilde{r}_3 \\
&\vdots & &\vdots \\
r_{i-1} &= q_i r_i + r_{i+1}, & \tilde{r}_{i-1} &= \tilde{q}_i \tilde{r}_i + \tilde{r}_{i+1} \\
&\vdots & &\vdots \\
r_{\ell-1} &= q_\ell r_\ell, & \tilde{r}_{\tilde{\ell}-1} &= \tilde{q}_{\tilde{\ell}} \tilde{r}_{\tilde{\ell}}
\end{aligned}
$$

- In particular, our interest is on the case $(r_0, r_1) \equiv_{2k} (\tilde{r}_0, \tilde{r}_1)$ ...

Quotient sequences of coinciding pairs of polynomials

- We can now study the execution on two *coinciding* inputs (r_0, r_1) and $(\tilde{r}_0, \tilde{r}_1)$ with $\deg r_0 \geq \deg r_1 \geq 0$ and $\deg \tilde{r}_0 \geq \deg \tilde{r}_1 \geq 0$ as follows

Lemma 9 (Identical quotient sequences up to the halting threshold)
Let $k \in \mathbb{Z}$ with $(r_0, r_1) \equiv_{2k} (\tilde{r}_0, \tilde{r}_1)$. Then, $h(k) = \tilde{h}(k)$ with $q_i = \tilde{q}_i$ for all $i = 1, 2, \ldots, h(k)$.

Proof.
Exercise □

Example: Quotient sequences of coinciding pairs

- Let us run the extended Euclidean algorithm for a pair of polynomials in $\mathbb{Z}_{11}[x]$:

i	q_i	r_i	s_i	t_i
0		$7 + x + 3x^2 + 5x^3 + 9x^4 + 10x^5 + 7x^6$	1	0
1	4	$4 + 10x + 7x^2 + 4x^3 + 7x^4 + 4x^5 + 10x^6$	0	1
2	$4 + 2x$	$2 + 5x + 8x^2 + 3x^4 + 5x^5$	1	7
3	$4 + 10x$	$7 + 8x + 9x^2 + 10x^3 + 6x^4$	$7 + 9x$	$6 + 8x$
4	$2 + 3x$	$7 + 2x + 2x^2 + 2x^3$	$6 + 4x + 9x^2$	$5 + 7x + 8x^2$
5	$10 + 9x$	$4 + 5x + 10x^2$	$6 + 5x + 3x^2 + 6x^3$	$7 + x + 7x^2 + 9x^3$
6	$4 + 8x$	$4x$	$1 + 10x + x^3 + x^4$	$1 + 6x^2 + x^3 + 7x^4$
7	x	4	$2 + x + 2x^3 + 10x^4 + 3x^5$	$3 + 4x + 5x^2 + x^3 + 8x^4 + 10x^5$
8		0	$1 + 8x + 10x^2 + x^3 + 10x^4 + x^5 + 8x^6$	$1 + 8x + 2x^2 + 7x^3 + 6x^4 + 3x^5 + x^6$

- Here is a run on a pair that coincides with the first pair up to length $2k = 4$:

i	q_i	r_i	s_i	t_i
0		$3 + 5x + 9x^2 + 10x^3 + 7x^4$	1	0
1	4	$7 + 4x + 7x^2 + 4x^3 + 10x^4$	0	1
2	$4 + 2x$	$8 + 3x^2 + 5x^3$	1	7
3	$4 + 10x$	$8 + 10x + 6x^2$	$7 + 9x$	$6 + 8x$
4	$6x$	$9 + x$	$6 + 4x + 9x^2$	$5 + 7x + 8x^2$
5	$8 + 7x$	8	$7 + 6x + 9x^2 + x^3$	$6 + 2x^2 + 7x^3$
6		0	$5 + 6x + 5x^2 + 6x^3 + 4x^4$	$1 + 9x + 3x^2 + 7x^3 + 6x^4$

- Observe that the quotient sequences agree up to total degree $\deg q_1 + \deg q_2 + \ldots + \deg q_{h(k)} \leq k$ with $h(k) = 3$

A divide-and-conquer extended Euclidean algorihtm

- We now use Lemma 9 to design a fast divide-and-conquer version of the extended Euclidean algorihtm
- For a given input $(r_0, r_1) \in F[x]^2$ with $\deg r_0 \geq \deg r_1 \geq 0$ and halting parameter $k \geq 0$, the key idea is to truncate the input using the "$\restriction$"-operator and build the quotient sequence $q_1, q_2, \ldots, q_{h(k)}$ using two recursive calls with halting parameter at most $\lfloor k/2 \rfloor$ each
- That is, the idea essentially to use the first recursive call to recover $q_1, q_2, \ldots, q_{h(\lfloor k/2 \rfloor)}$, then compute (as needed) the next quotient $q_{h(\lfloor k/2 \rfloor)+1}$ explicitly, and then make a second recursive call (as needed) to recover the rest of the quotient sequence $q_1, q_2, \ldots, q_{h(k)}$
- With careful implementation, this leads to an algorithm that runs in $O(M(k) \log k)$ operations in F
- Before describing the algorithm in detail, let us recall some further terminology ...

Invariants of the extended Euclidean algorithm

- Recall the matrices

$$R_0 = \begin{bmatrix} s_0 & t_0 \\ s_1 & t_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad Q_i = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \quad \text{for } i = 1, 2, \ldots, \ell$$

  and $R_i = Q_i Q_{i-1} \cdots Q_1 R_0 \in F[x]^{2\times 2}$ for $i = 0, 1, \ldots, \ell$ from the analysis of the traditional extended Euclidean algorithm in Problem Set 1

- We recall that for all $i = 0, 1, \ldots, \ell$ we have $R_i = \begin{bmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{bmatrix}$ and $R_i \begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix}$

- Our algorithm design will be such that on input (r_0, r_1) and k it produces as output (i) the value $h(k)$, (ii) the quotient sequence $q_1, q_2, \ldots, q_{h(k)}$, and (iii) the matrix $R_{h(k)}$ ...

Truncating inputs to the extended Euclidean algorithm

- Let us write $h(k), q_1, q_2, \ldots, q_{h(k)}, R_{h(k)} \leftarrow \mathrm{extgcd}(k, r_0, r_1)$ to indicate that the algorithm produces the output $h(k), q_1, q_2, \ldots, q_{h(k)}, R_{h(k)}$ on input k, r_0, r_1 with $\deg r_0 \geq \deg r_1 \geq 0$
- Lemma 9 now implies that we have

$$\mathrm{extgcd}(k, r_0, r_1) = \mathrm{extgcd}\Big(k, r_0 \restriction 2k, r_1 \restriction (2k - (\deg r_0 - \deg r_1))\Big) \tag{30}$$

- In particular, we can assemble the output recursively so that the input polynomials to each recursive call are truncated in degree to the minimum enabled by (30)
- We are now ready for the detailed pseudocode of the algorithm ...

A divide-and-conquer extended Euclidean algorithm I

- Let F be a field and let $k \in \mathbb{Z}$ and $r_0, r_1 \in F[x]$ with $\deg r_0 \geq \deg r_1$ and $r_0 \neq 0$ be given as input

1. If $k < \deg r_0 - \deg r_1$ holds, then return with output $h(k) \leftarrow 0$ and $R_{h(k)} \leftarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
2. If $k = 0$ and $\deg r_0 = \deg r_1$ hold, then return with output $h(k) \leftarrow 1$, $q_1 = \frac{\operatorname{lc} r_0}{\operatorname{lc} r_1}$, and $R_{h(k)} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -\frac{\operatorname{lc} r_0}{\operatorname{lc} r_1} \end{bmatrix}$
3. Set $k_1 \leftarrow \lfloor k/2 \rfloor$
4. Make the first recursive call $h_1, q_1^{(1)}, q_2^{(1)}, \ldots, q_{h_1}^{(1)}, R^{(1)} \leftarrow \operatorname{extgcd}\big(k_1, r_0 \upharpoonright 2k_1, r_1 \upharpoonright (2k_1 - (\deg r_0 - \deg r_1))\big)$
5. Compute the matrix-vector product $\begin{bmatrix} \tilde{r}_{h_1} \\ \tilde{r}_{h_1+1} \end{bmatrix} \leftarrow R^{(1)} \begin{bmatrix} r_0 \upharpoonright 2k \\ r_1 \upharpoonright (2k - (\deg r_0 - \deg r_1)) \end{bmatrix}$

A divide-and-conquer extended Euclidean algorithm II

6. If $\deg q_1^{(1)} + \deg q_2^{(1)} + \ldots + \deg q_{h_1}^{(1)} + \deg \tilde{r}_{h_1} - \deg \tilde{r}_{h_1+1} > k$ holds, then return with output $h(k) \leftarrow h_1$, $q_1, q_2, \ldots, q_{h(k)} \leftarrow q_1^{(1)}, q_2^{(1)}, \ldots, q_{h_1}^{(1)}$, and $R_{h(k)} \leftarrow R^{(1)}$
7. Compute the quotient $q_{h_1+1} \leftarrow \tilde{r}_{h_1} \operatorname{quo} \tilde{r}_{h_1+1}$ and the matrix $Q_{h_1+1} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -q_{h_1+1} \end{bmatrix}$
8. Compute the remainder $\tilde{r}_{h_1+2} \leftarrow \tilde{r}_{h_1} - q_{h_1+1}\tilde{r}_{h_1+1}$
9. Set $k_2 \leftarrow k - (\deg q_1^{(1)} + \deg q_2^{(1)} + \ldots + \deg q_{h_1}^{(1)} + \deg q_{h_1+1})$
10. Make the second recursive call $h_2, q_1^{(2)}, q_2^{(2)}, \ldots, q_{h_2}^{(2)}, R^{(2)} \leftarrow \operatorname{extgcd}\left(k_2, \tilde{r}_{h_1+1} \restriction 2k_1, \tilde{r}_{h_1+2} \restriction (2k_1 - (\deg \tilde{r}_{h_1+1} - \deg \tilde{r}_{h_1+2}))\right)$
11. Return with output $h(k) \leftarrow h_1 + 1 + h_2$, $q_1, q_2, \ldots, q_{h(k)} \leftarrow q_1^{(1)}, q_2^{(1)}, \ldots, q_{h_1}^{(1)}, q_{h_1+1}, q_1^{(2)}, q_2^{(2)}, \ldots, q_{h_2}^{(2)}$, and $R_{h(k)} \leftarrow R^{(2)} Q_{h_1+1} R^{(1)}$

Remarks and analysis

- Caveat: In Step 1 we may have $\deg r_1 = -\infty$ (that is, $r_1 = 0$) and in Step 6 we may have $\deg \tilde{r}_{h_1+1} = -\infty$ (that is, $\tilde{r}_{h_1+1} = 0$)
- After Step 1 it holds that $k \geq \deg r_0 - \deg r_1 \geq 0$, after Step 2 it holds that $k \geq 1$ and $\deg r_0 > \deg r_1 \geq 0$; thus, $0 \leq k_1 \leq k - 1$
- After Step 5 we have

$$\deg q_1^{(1)} + \deg q_2^{(1)} + \ldots + \deg q_{h_1}^{(1)} \leq k_1$$

  and, also recalling that $k_1 = \lfloor k/2 \rfloor$,

$$\deg q_1^{(1)} + \deg q_2^{(1)} + \ldots + \deg q_{h_1}^{(1)} + \deg \tilde{r}_{h_1} - \deg \tilde{r}_{h_1+1} \geq k_1 + 1 \geq \lceil k/2 \rceil$$

- Assuming that $\tilde{r}_{h_1+1} \neq 0$, we have $\deg q_{h_1+1} = \deg \tilde{r}_{h_1} - \deg \tilde{r}_{h_1+1}$
- Thus, $k_2 \leq \lfloor k/2 \rfloor \leq k - 1$
- The algorithm runs in $T(k) \leq T(k_1) + T(k_2) + O(M(k)) \leq 2T(\lfloor k/2 \rfloor) + O(M(k))$ operations in F; that is, $T(k) = O(M(k) \log k)$ operations in F

Key content for Lecture 5 (recalled)

- **Extended Euclidean algorithm** for polynomials recalled and expanded
  - The **quotient sequence**, the **Bézout coefficients**, and the **halting threshold**
- Fast extended Euclidean algorithm for polynomials by **divide and conquer**
  - The two polynomial operands **truncated** to a prefix of the highest-degree monomials determine the prefix of the quotient sequence (exercise)

Learning objectives (1/2)

- Terminology and objectives of modern algorithmics, including elements of algebraic, 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

References I

[1] M. Agrawal, N. Kayal, and N. Saxena, PRIMES is in P, *Ann. of Math. (2)* 160 (2004), 781–793.
[doi:10.4007/annals.2004.160.781].

[2] P. Austrin, P. Kaski, and K. Kubjas, Tensor network complexity of multilinear maps, in *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA* (A. Blum, Ed.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, pp. 7:1–7:21.
[doi:10.4230/LIPIcs.ITCS.2019.7].

[3] R. C. Baker, G. Harman, and J. Pintz, The difference between consecutive primes. II, *Proc. London Math. Soc. (3)* 83 (2001), 532–562.
[doi:10.1112/plms/83.3.532].

References II

[4] A. Björklund and P. Kaski, How proofs are prepared at Camelot: extended abstract, in *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016* (G. Giakkoupis, Ed.). ACM, 2016, pp. 391–400.
[doi:10.1145/2933057.2933101].

[5] R. Brent and P. Zimmermann, *Modern Computer Arithmetic*, Cambrigde University Press, 2011.
[WWW].

[6] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider, Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility, in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016* (M. Sudan, Ed.). ACM, 2016, pp. 261–270.

References III

[doi:10.1145/2840728.2840746].

[7] D. A. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, fourth ed., Springer, Cham, 2015.
[doi:10.1007/978-3-319-16721-3].

[8] M. Fürer, Faster integer multiplication, *SIAM J. Comput.* 39 (2009), 979–1005.
[doi:10.1137/070711761].

[9] S. Gao, A new algorithm for decoding Reed–Solomon codes, in *Communications, Information, and Network Security* (V. K. Bhargava, H. V. Poor, V. Tarokh, and S. Yoon, Eds.), Springer, 2003, pp. 55–68.

[10] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, third ed., Cambridge University Press, Cambridge, 2013.
[doi:10.1017/CBO9781139856065].

References IV

[11] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, Delegating computation: Interactive proofs for muggles, *J. ACM* 62 (2015), 27:1–27:64. [doi:10.1145/2699436].

[12] D. Harvey, J. van der Hoeven, and G. Lecerf, Even faster integer multiplication, *J. Complexity* 36 (2016), 1–30. [doi:10.1016/j.jco.2016.03.001].

[13] N. Karimi, P. Kaski, and M. Koivisto, Error-correcting and verifiable parallel inference in graphical models, in *Proceedings of AAAI'20*, to appear.

[14] P. Kaski, Engineering a delegatable and error-tolerant algorithm for counting small subgraphs, in *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018, New Orleans, LA, USA, January 7-8, 2018.* (R. Pagh and S. Venkatasubramanian, Eds.). SIAM, 2018, pp. 184–198. [doi:10.1137/1.9781611975055.16].

References V

[15] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, 1998.

[16] S. Lang, *Algebra*, third ed., Springer-Verlag, New York, 2002. [doi:10.1007/978-1-4613-0041-0].

[17] N. Möller, On Schönhage's algorithm and subquadratic integer GCD computation, *Math. Comp.* 77 (2008), 589–607. [doi:10.1090/S0025-5718-07-02017-0].

[18] A. Schönhage, Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2, *Acta Informat.* 7 (1976/77), 395–398. [doi:10.1007/BF00289470].

[19] A. Schönhage and V. Strassen, Schnelle Multiplikation grosser Zahlen, *Computing (Arch. Elektron. Rechnen)* 7 (1971), 281–292.

References VI

[20] A. Shamir, How to share a secret, *Comm. ACM* 22 (1979), 612–613. [doi:10.1145/359168.359176].

[21] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, 1992. [doi:10.1137/1.9781611970999].

[22] M. Walfish and A. J. Blumberg, Verifying computations without reexecuting them, *Commun. ACM* 58 (2015), 74–84. [doi:10.1145/2641562].

[23] R. R. Williams, Strong ETH breaks with Merlin and Arthur: Short non-interactive proofs of batch evaluation, in *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan* (R. Raz, Ed.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 2:1–2:17. [doi:10.4230/LIPIcs.CCC.2016.2].