3. Quotient and remainder

CS-E4500 Advanced Course on Algorithms
Spring 2019

Petteri Kaski
Department of Computer Science
Aalto University

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

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

2019 — **K A L E N T E R I** — **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

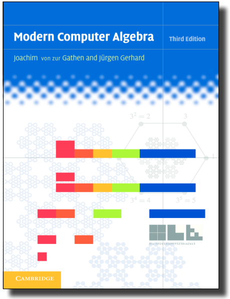
T = Tutorial (model solutions); hall T6, Mon 16–18

Recap of last week

- Evaluation–interpolation duality of polynomials
- Multiplication is a pointwise product in the dual
- Transforming between the primal and a (carefully chosen) dual
  —**roots of unity** and the **discrete Fourier transform** (DFT)
- The positional number system for integers
- Factoring a composite-order DFT to obtain a **fast Fourier transform** (FFT)
- Fast cyclic convolution (assuming a suitable root of unity exists)
- Fast negative-wrapping cyclic convolution

Goal: Near-linear-time toolbox for univariate polynomials

- Multiplication
- Division (quotient and remainder) (this week)
- Batch evaluation
- Interpolation
- Extended Euclidean algorithm (gcd)
- 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 29634-0975, 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

Further motivation for this week

- The radix-point representation for rational numbers is at the foundation of **floating-point arithmetic**
- Most scientific and engineering computations today are executed using hardware that implements the **IEEE 754-2008 standard** for floating point arithmetic:

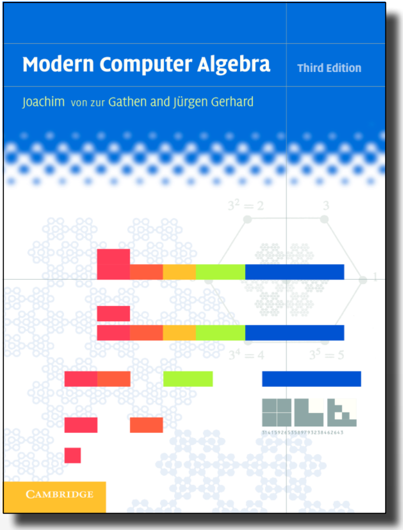
  https://doi.org/10.1109%2FIEEESTD.2008.4610935
- Floating-point numbers and floating-point arithmetic are a fantastic tool, but this tool comes with caveats and must be used with care
- Quick demo:
  IEEE 754-2008 in action

Key content for Lecture 3

- **Division** (**quotient** and **remainder**) for integers and polynomials
- Fast division by **reduction** to fast multiplication
- Integer division via **approximation** of the multiplicative inverse of the divisor
- The radix-point representation and approximation of rational numbers
- **Newton iteration**
- Newton iteration for the multiplicative inverse of the divisor
- **Convergence analysis** for Newton iteration
- Polynomial division via **reversal**
- Newton iteration for the inverse of the reverse of the divisor

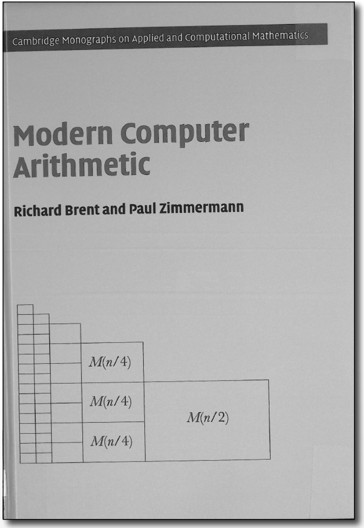
Fast quotient and remainder (polynomials)

(von zur Gathen and Gerhard [11],
Sections 9.1 and 9.4)



Integer and floating-point arithmetic

(Brent and Zimmermann [4])



Division (quotient and remainder)

- We start by recalling polynomial division and integer division
- We also recall that we can multiply fast, both in the case of polynomials and in the case of integers
- Our goal for this lecture is to develop division algorithms that are essentially (up to constants) *as fast as our multiplication algorithms*
- The key idea is to proceed by **reduction** to multiplication
- In preparing the reductions, we recall and encounter many useful concepts ...

Polynomial quotient and remainder

- Let R be a ring
- Let $a = \sum_{i=0}^{n} \alpha_i x^i \in R[x]$ and $b = \sum_{i=0}^{m} \beta_i x^i \in R[x]$ such that $\alpha_n \neq 0$ and $\beta_m = 1$
- That is, $\deg a = n$ and b is monic with $\deg b = m$
- Then, there exist polynomials $q, r \in R[x]$ that satisfy $a = qb + r$ with $\deg r < \deg b$
- We write $a \operatorname{quo} b$ for such a **quotient** q and $a \operatorname{rem} b$ for such a **remainder** r in the division of a by b
- In fact, such q and r are unique (exercise)

Integer quotient and remainder

- Let $\alpha, \beta \in \mathbb{Z}_{\geq 0}$ with $\beta \neq 0$
- Then, there exist integers $\eta, \rho \in \mathbb{Z}_{\geq 0}$ that satisfy $\alpha = \eta\beta + \rho$ with $0 \leq \rho \leq \beta - 1$
- We write $\alpha \operatorname{quo} \beta$ the **quotient** η and $\alpha \operatorname{rem} \beta$ the **remainder** ρ in the division of α by β
- Such η and ρ are unique (exercise)

The classical division algorithm (for polynomials)

- Let $a = \sum_i \alpha_i x^i, b = \sum_i \beta_i x^i \in R[x]$ be given as input with $\deg a = n$, $\deg b = m$, $n \geq m \geq 0$, 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
- The classical algorithm runs in $O((n+m)^2)$ operations in R
- ... But could we do better? After Lecture 2, we know how to multiply in near-linear-time ...

Fast polynomial multiplication

- Let R be a ring
- Given $f, g \in R[x]$ with $\deg f \leq d$ and $\deg g \leq d$ as input, we can compute the product $fg \in R[x]$ in $O(M(d))$ operations in R
- We can take $M(d) = O(d \log d)$ if R has a primitive root of unity that supports an appropriate FFT
- In general, we can take $M(d) = O(d \log d \log \log d)$
- (In Lecture 2 we explored Schönhage–Strassen multiplication that assumes 2 is a unit in R; this algorithm can be generalized so that R is an arbitrary ring.)

Fast integer multiplication

- Given as input $\alpha, \beta \in \mathbb{Z}_{\geq 0}$ represented as at most d-digit integers in a constant base $B \in \mathbb{Z}_{\geq 2}$, we can compute the product $\alpha\beta \in \mathbb{Z}$ in $O(M(d))$ time
- We can take $M(d) = O(d \log d \log \log d)$ [24] or $M(d) = O(d \log d 2^{O(\log^* d)})$ [9, 14]
- (Also recall Problem Set 2 where we reduced multiplication in $\mathbb{Z}$ to multiplication in $\mathbb{Z}_u[x]$.)

First reduction towards division: the quotient suffices

- Division (viewed from 36,000ft, see earlier slides for details):

  Given a, b we need to compute q, r such that $a = qb + r$

- Observation:
  It suffices to compute q since then we can recover $r = a - qb$ by fast multiplication

High-level idea: iterate for the quotient

- Our approach will be to recover the quotient **iteratively**
- In essence, we iterate for a (near) multiplicative inverse of the divisor b such that each iteration increases the accuracy of our (near) inverse
- We want the accuracy (e.g. number of digits or polynomial degree) to increase **geometrically** from n to $2n$ in one iteration
- Once a sufficiently close **approximation** of the inverse is available (n is large enough), we proceed to solve for the quotient
- Each iteration will involve a constant number of multiplications, additions, and subtractions on inputs of size $O(n)$

The cost of a geometric iteration

- We say that a function $T : \mathbb{Z}_{\geq n_0} \to \mathbb{Z}_{\geq 0}$ **grows at least linearly** if for all $n, n_1, n_2 \in \mathbb{Z}_{\geq n_0}$ it holds that $n = n_1 + n_2$ implies $T(n) \geq T(n_1) + T(n_2)$
- *Examples:*
  $T(n) = Cn \log_2 n$ for $n_0 = 1$ and any constant $C > 0$
  $T(n) = Cn \log_2 n \log_2 \log_2 n$ for $n_0 = 2$ and any constant $C > 0$

Lemma 5 (Last step dominates—the previous steps are "for free")

Suppose that T grows at least linearly for $n \geq n_0 \geq 1$ and let 2^{k_0} be the least integer power of 2 at least n_0. Then, for all $k \geq k_0$ we have $\sum_{j=k_0}^{k} T(2^j) \leq T(2^{k+1})$

Proof.

By induction (exercise). $\square$

Roadmap for fast integer division

- The positional number system in base B recalled and revisited —the radix-point representation and approximation of rational numbers
- For $\alpha, \beta \in \mathbb{Z}_{\geq 1}$ given as input, we want a (radix-point) approximation γ for the multiplicative inverse $1/\beta$
- Provided the approximation γ is accurate enough, from the product $\alpha\gamma$ we can recover the quotient $\alpha \operatorname{quo} \beta$ (exercise) and thus the remainder $\alpha \operatorname{rem} \beta$
- To compute γ fast from a d-digit β given as input, we rely on **Newton iteration**
- We present a Newton iteration for a normalized rational divisor; that is, we normalize the integer β to a radix-point ν with $B^{-1} \leq \nu < 1$, then compute an approximate multiplicative inverse μ for ν using Newton iteration, and from μ map back to the desired γ

Approximating the multiplicative inverse of the divisor

- Given $\alpha, \beta \in \mathbb{Z}_{\geq 1}$ as input, we seek to approximate $1/\beta \in \mathbb{Q}$
- We observe in particular that $1/\beta$ is a *rational number*, not an integer
- Thus, first we need means for computing with rational numbers ...
- Let us begin by recalling and revisiting yet further aspects of the positional number system ...

The positional number system for integers (base B)

- Let $B \in \mathbb{Z}_{\geq 2}$
- Suppose that $\alpha \in \mathbb{Z}$ with $0 \leq \alpha \leq B^d - 1$ for some $d \in \mathbb{Z}_{\geq 0}$
- Then, there is a unique finite sequence

$$(\alpha_0, \alpha_1, \ldots, \alpha_{d-2}, \alpha_{d-1}) \in \mathbb{Z}^d_{\geq 0} \tag{20}$$

with $0 \leq \alpha_i \leq B - 1$ for all $i = 0, 1, \ldots, d-1$ such that

$$\alpha = \sum_{i=0}^{d-1} \alpha_i B^{d-1-i} = \alpha_0 B^{d-1} + \alpha_1 B^{d-2} + \ldots + \alpha_{d-3} B^2 + \alpha_{d-2} B + \alpha_{d-1} \tag{21}$$

- We say that the sequence (20) is the (d-digit) representation of the integer α in the positional number system with **base** B (or **radix** B)
- The elements α_i are the **digits** of α
- We say that α_0 is the **most significant** digit and α_{d-1} is the **least significant** digit

Example (base 10)

- Let us represent $123 \in \mathbb{Z}$ in base $B = 10$
- We have
$$123 = 1 \cdot 10^2 + 2 \cdot 10 + 3 \cdot 1$$
- Hence, the sequence $(1, 2, 3)$ represents 123 in base 10

A positional number system for rational numbers?

- Could we extend the positional number system to represent (all) rational numbers?
- Let us make an attempt (that will not succeed for all rational numbers) ...

Radix-point representation (base B)

- Let $B \in \mathbb{Z}_{\geq 2}$
- Let $s \in \{-1, 1\}$, $e \in \mathbb{Z}$, and $d \in \mathbb{Z}_{\geq 1}$
- Let $(\alpha_0, \alpha_1, \ldots, \alpha_{d-1}) \in \mathbb{Z}^d$ such that $0 \leq \alpha_i \leq B - 1$ for all $i = 0, 1, \ldots, d - 1$
- We say that the three-tuple $(s, e, (\alpha_0, \alpha_1, \ldots, \alpha_{d-1}))$ is a **radix-point representation** of the rational number

$$\alpha = sB^e \sum_{i=0}^{d-1} \alpha_i B^{-i} \tag{22}$$

  using d **digits** in **base** B
- We say a representation is **normal** if both $\alpha_0 \neq 0$ and $\alpha_{d-1} \neq 0$
- Any nonzero rational number that has a radix-point representation in base B has a unique normal representation in base B (exercise)
- Define $(1, 0, (0))$ as the unique normal representation for the rational number 0

Example (base 10)

- Let us represent $1234657/10000 \in \mathbb{Q}$ in base $B = 10$
- We have
$$\begin{aligned}\frac{1234567}{10000} &= 1 \cdot 10^2 + 2 \cdot 10 + 3 \cdot 1 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2} + 6 \cdot 10^{-3} + 7 \cdot 10^{-4} \\ &= 10^2 \left(1 \cdot 1 + 2 \cdot 10^{-1} + 3 \cdot 10^{-2} + 4 \cdot 10^{-3} + 5 \cdot 10^{-4} + 6 \cdot 10^{-5} + 7 \cdot 10^{-6} \right)\end{aligned}$$
- Hence, the sequence $(1, 2, (1, 2, 3, 4, 5, 6, 7))$ is the (normal) representation of $1234657/10000$ using $d = 7$ digits and exponent $e = 2$ in base $B = 10$
- $(1, 2, (1, 2, 3, 4, 5, 6, 7))$ is rather cumbersome to write, so often one resorts to notational shorthands such as 123.4567 or $1.23456 \cdot 10^2$ where the **radix point** "." is used to separate the integer and fractional parts of the representation (with the base $B = 10$ tacitly understood unless indicated otherwise)

A positional number system for rational numbers?

- Could we extend the positional number system to represent (all) rational numbers?
- For any base B, there exist rational numbers that do not admit radix-point representation in base B (exercise)
- For example, $1/3$ cannot be represented in base $B = 10$
- However, for any rational number $\tau \in \mathbb{Q}$, one we represent a rational number *arbitrarily close* to τ using radix-point representation
- For example, $3.3333333333333333333 \cdot 10^{-1}$ in base $B = 10$ is already rather close to $1/3$

Properties of radix-point numbers (1/2)

- Let us fix the base $B \in \mathbb{Z}_{\geq 2}$
- Let us write $\mathbb{Q}_B$ for the set of all rational numbers that do admit a radix-point representation in base B
- It is immediate that we have $\mathbb{Z} \subseteq \mathbb{Q}_B$
- For all $\alpha, \beta \in \mathbb{Q}_B$, we have the **closure** properties $\alpha + \beta \in \mathbb{Q}_B$, $-\alpha \in \mathbb{Q}_B$, and $\alpha\beta \in \mathbb{Q}_B$
- However, as we have seen, for all $\alpha \in \mathbb{Q}_B$ it does *not* hold in general that $1/\alpha \in \mathbb{Q}_B$ (indeed, recall from the previous example that $3 \in \mathbb{Q}_{10}$ and $1/3 \notin \mathbb{Q}_{10}$)

Example: Closure under multiplication

- Let $\alpha, \beta \in \mathbb{Q}_B$ have radix point representations

$$\alpha = sB^e \sum_{i=0}^{c-1} \alpha_i B^{-i}$$

$$\beta = tB^f \sum_{i=0}^{d-1} \beta_i B^{-i}$$

- We have

$$\alpha\beta = stB^{e+f-d+1-c+1}\left(\sum_{i=0}^{c-1} \alpha_i B^{c-1-i} \sum_{i=0}^{d-1} \beta_i B^{d-1-i}\right)$$

- The expression in parentheses is a multiplication of two *integers* in base B
- Since the integer product is representable in base B, we have that $\alpha\beta$ admits a radix-point representation in base B (by shifting the position of the radix point)

Properties of radix-point numbers (2/2)

- From the previous example we also observe that if we multiply a c-digit representation with a d-digit representation, the product has a representation using at most cd digits
- Indeed, the largest integer that one can represent using d digits in base B is $(B-1)\sum_{j=0}^{d-1} B^j = B^d - 1$
- *Question/work point:*
  How about closure under addition? Hint: again reduce to integers, and be careful with the number of digits you need to represent the sum
- For *exact* arithmetic, the increase from c and d digits to cd digits at each multiplication quickly becomes very expensive when evaluating an arithmetic expression consisting of several operations
- We need a way to control this expense; that is, instead of *exact* arithmetic, we will be content on *approximation* where we can control the *accuracy* of the approximation ...

Example: Addition

- Let us work in base $B = 3$
- Suppose that $\alpha = 1.121001112 \cdot 3^2$ and $\beta = 2.222221202 \cdot 3^6$
- Aligning the radix points, addition reduces to integer addition (in base B):

$$\begin{array}{rr} & 112.1001112 \\ + & 2222221.202 \\ \hline & 10000111.0021112 \end{array}$$

- The result is thus $\alpha + \beta = 1.00001110021112 \cdot 3^7$

Example: Multiplication

- Let us work in base $B = 5$
- Suppose that $\alpha = 3.011002342 \cdot 5^4$ and $\beta = 1.340011441 \cdot 5^4$
- Multiplication reduces to integer multiplication (in base B):

$$
\begin{array}{lr}
 & 3011002342 \cdot 5^{-5} \\
\cdot & 1340011441 \cdot 5^{-5} \\
\hline
 & 1014034103032013242 2 \cdot 5^{-10}
\end{array}
$$

- The result is thus $\alpha\beta = 1.0140341030320132422 \cdot 5^9$

Cutting expenses—rounding

- A principled way of cutting the expense of maintaining a d-digit radix-point representation is to cut the number of digits from d digits to ℓ digits for some $1 \leq \ell \leq d$
- Being "principled" of course amounts to making sure that the ℓ-digit representation is a "close approximation" of the d-digit representation
- This general process of cutting expenses at intermediate steps of a computation using "close approximations" is also known as **rounding**
- We will restrict to a straightforward but blunt form of rounding, namely *truncation* ...

Truncation

- Let $\alpha \in \mathbb{Q}_B$ with

$$\alpha = sB^e \sum_{i=0}^{d-1} \alpha_i B^{-i}$$

- For $\ell \in \mathbb{Z}_{\geq 1}$, the **truncation** of α to ℓ **digits** is the rational number

$$\alpha_{\underline{\ell}} = sB^e \sum_{i=0}^{\min(\ell,d)-1} \alpha_i B^{-i} \tag{23}$$

- That is, in effect we cut out all but the ℓ most significant digits of α to obtain $\alpha_{\underline{\ell}}$

Example: Truncation

- Let us truncate 123.4567 in base $B = 10$
- We have

$$
\begin{aligned}
123.4567_{\underline{7}} &= 123.4567 \\
123.4567_{\underline{6}} &= 123.456 \\
123.4567_{\underline{5}} &= 123.45 \\
123.4567_{\underline{4}} &= 123.4 \\
123.4567_{\underline{3}} &= 123 \\
123.4567_{\underline{2}} &= 120 \\
123.4567_{\underline{1}} &= 100
\end{aligned}
$$

Accuracy of truncation

- Let $\alpha \in \mathbb{Q}_B$ with $\alpha = sB^e \sum_{i=0}^{d-1} \alpha_i B^{-i}$ and let $\ell = 1, 2, \ldots$
- Let us measure the loss in accuracy when truncating from α to $\alpha_{\underline{\ell}}$ by $\delta \in \mathbb{Q}$ with

$$\alpha_{\underline{\ell}} = \alpha + \delta$$

- We have

$$|\delta| = |\alpha - \alpha_{\underline{\ell}}| = |sB^e \sum_{i=\ell}^{d-1} \alpha_i B^{-i}|$$

  Thus,

$$|\delta| \leq B^e(B-1) \sum_{i=\ell}^{d-1} B^{-i} = \begin{cases} B^{e-\ell+1} - B^{e-d+1} & \text{if } \ell \leq d \\ 0 & \text{if } \ell \geq d \end{cases} \tag{24}$$

- In particular, for all $\ell = 1, 2, \ldots$ we have $|\delta| < B^{e-\ell+1}$

Example: Accuracy of truncation

- Let us again truncate 123.4567 in base $B = 10$
- Since $e = 2$, we have

$$
\begin{aligned}
|123.4567 - 123.4567_{\underline{7}}| &= 0 < 10^{2-7+1} = 10^{-4} \\
|123.4567 - 123.4567_{\underline{6}}| &= 0.0007 < 10^{2-6+1} = 10^{-3} \\
|123.4567 - 123.4567_{\underline{5}}| &= 0.0067 < 10^{2-5+1} = 10^{-2} \\
|123.4567 - 123.4567_{\underline{4}}| &= 0.0567 < 10^{2-4+1} = 10^{-1} \\
|123.4567 - 123.4567_{\underline{3}}| &= 0.4567 < 10^{2-3+1} = 10^{0} \\
|123.4567 - 123.4567_{\underline{2}}| &= 3.4567 < 10^{2-2+1} = 10^{1} \\
|123.4567 - 123.4567_{\underline{1}}| &= 23.4567 < 10^{2-1+1} = 10^{2}
\end{aligned}
$$

Summary—rational numbers with controlled expense

- Let us summarize where we are before proceeding further
- Radix-point numbers in $\mathbb{Q}_B$ enable us to compute with arbitrarily close approximations of rational numbers in $\mathbb{Q}$
- Computation in $\mathbb{Q}_B$ takes place by easy reductions to *integer* algorithms for addition, negation, and multiplication
- In particular, we can choose to compute *exactly* in $\mathbb{Q}_B$ as long as we mind the cost of an increase in the number of digits that we need to maintain
- This cost can be controlled by *rounding* (for example, truncating) intermediate results to fewer digits
- *As algorithm designers we can trade off between accuracy and cost of computation by rounding (truncating) to an appropriate number of digits*

Approximating the multiplicative inverse of the divisor

- Let us restate our goal towards fast integer division
- Given $\alpha, \beta \in \mathbb{Z}_{\geq 1}$ as input, we seek to approximate $1/\beta \in \mathbb{Q}$
- We observe in particular that $1/\beta$ is a *rational number*
- We now have a means for working with rational numbers, namely the radix-point number system in base $B \in \mathbb{Z}_{\geq 2}$, with $B = O(1)$
- That is, our goal is to approximate $1/\beta$ with a radix-point number $\gamma \in \mathbb{Q}_B$
- We have that γ and $1/\beta$ are close to each other if and only if $\gamma\beta$ is close to 1
- In what follows our goal is, given as input $t \in \mathbb{Z}_{\geq 1}$ and $\nu \in \mathbb{Q}_B$ with $B^{-1} \leq \nu < 1$, to compute a $\mu \in \mathbb{Q}_B$ with $|1 - \mu\nu| \leq B^{-t}$ in time $O(M(t))$
- (This running time will be sufficient to obtain an $O(M(n))$-time division algorithm for two given integers $\alpha, \beta \in \mathbb{Z}_{\geq 1}$ with at most n digits each in base B)

Key idea: Iteration for improved approximation

- Let us set issues of computational cost aside for a moment and look at how to obtain better and better approximations for $1/\nu$
- That is to say, suppose we have available an approximation $\mu \in \mathbb{Q}$ of $1/\nu$ with
$$|1 - \mu\nu| \leq \epsilon$$
for some $0 \leq \epsilon < 1$
- We would like to compute from μ an improved approximation $\hat{\mu} \in \mathbb{Q}$ with, say,
$$|1 - \hat{\mu}\nu| \leq \epsilon^2$$
- One way to achieve such transformation $\mu \mapsto \hat{\mu}$ is to use **Newton iteration** ...

A remark in passing

- While an improvement from ϵ to ϵ^2 in accuracy may at first look innocent, at $\epsilon \leq 1/B$ it in fact *doubles* the accuracy in terms of *the number of digits* at every step
- For example with $B = 10$, starting with $\epsilon = 0.1$ and iterating, we have

$$
\begin{aligned}
\epsilon &= 0.1 \\
\epsilon^2 &= 0.01 \\
\epsilon^4 &= 0.0001 \\
\epsilon^8 &= 0.0000001 \\
\epsilon^{16} &= 0.0000000000000001 \\
\epsilon^{32} &= 0.00000000000000000000000000000001 \\
\epsilon^{64} &= 0.0000000000000000000000000000000000000000000000000000000000000001 \\
&\vdots
\end{aligned}
$$

Newton iteration (1/3)

- Let us continue to work without considerations of computational cost yet
- Suppose we have a function $\varphi : I \to \mathbb{R}$ for some open interval $I \subseteq \mathbb{R}$ and we seek to find a $\mu \in I$ such that $\varphi(\mu) = 0$
- For example, suppose that $\varphi(x) = 1/x - \nu$ with $I = (0, \infty)$ for some $\nu \in \mathbb{R}_{>0}$
- Let us assume that $\varphi(x)$ is well-behaved in the sense that it is differentiable with a nonzero derivative in I; continuing the previous example, we have $\varphi'(x) = -1/x^2$
- Suppose that we have access to a $\mu \in I$ such that $\varphi(\mu)$ is close to 0
- *How could we obtain a $\hat{\mu} \in I$ such that $\varphi(\hat{\mu})$ is even closer to 0?*

Newton iteration (2/3)

- Using the fact that φ is differentiable, let us linearize $\varphi(x)$ at $x = \mu$
- We obtain the line

$$y = \varphi'(\mu)(x - \mu) + \varphi(\mu)$$

- Let us set $y = 0$ and solve for x to obtain

$$x = \mu - \frac{\varphi(\mu)}{\varphi'(\mu)}$$

- Setting

$$\hat{\mu} \leftarrow \mu - \frac{\varphi(\mu)}{\varphi'(\mu)}$$

  would now intuitively appear like a good choice to improve from μ assuming that φ does not deviate too much from a line between μ and an actual zero of φ

Newton iteration (3/3)

- In our example with $\varphi(x) = 1/x - \nu$ and $\varphi'(x) = -1/x^2$, we obtain
$$x = \mu - \frac{\varphi(\mu)}{\varphi'(\mu)} = \mu - (-1 + \mu\nu)\mu = (2 - \mu\nu)\mu$$
- Thus, we obtain the iteration step
$$\hat{\mu} \leftarrow (2 - \mu\nu)\mu$$
- Let us next verify that this iteration has the desired convergence property ...

Convergence analysis

- Let $0 \leq \epsilon < 1$ and $\nu \in (0, \infty)$
- Suppose that $\mu \in (0, \infty)$ satisfies $\mu\nu = 1 + \delta$ for some $\delta \in \mathbb{R}$ with $|\delta| \leq \epsilon < 1$
- Recall the iteration step

$$\hat{\mu} \leftarrow (2 - \mu\nu)\mu$$

- We thus have

$$\hat{\mu}\nu = (2 - \mu\nu)\mu\nu = (2 - (1 + \delta))(1 + \delta) = (1 - \delta)(1 + \delta) = 1 - \delta^2$$

- That is, one step of the iteration improves the accuracy from ϵ to ϵ^2 as desired
- Caveat: the iteration must be started from a value $\mu \in (0, \infty)$ with $|1 - \mu\nu| < 1$

Accounting for the computational cost

- The previous derivation and analysis assumed no computational cost on the exact arithmetic
- Let us now return to work in $\mathbb{Q}_B$ for $B \in \mathbb{Z}_{\geq 2}$ and $B = O(1)$, keeping track on the number of digits in our radix-point numbers, and taking care to truncate to control cost
- This requires an updated convergence analysis to establish convergence even in the presence of truncations...

Preliminaries: Normalizing the exponent of the divisor

- Rather than work with an integer divisor $\beta \in \mathbb{Z}_{\geq 1}$, it will be convenient to work with a normalized divisor $\nu \in \mathbb{Q}_B$ with $B^{-1} \leq \nu < 1$
- For $\beta = B^e \sum_{i=0}^{d-1} \beta_i B^{-i}$ with $\beta_0 \neq 0$ given as input, let us set $\nu = B^{-e-1}\beta$ to obtain $B^{-1} \leq \nu < 1$
- Note: Setting $\nu = B^{-e-1}\beta$ merely adjusts the exponent in radix-point representation; or, what is the same, moves the position of the radix point
- Suppose we are also given as input a $t \in \mathbb{Z}_{\geq 1}$
- In what follows we present an algorithm that computes a $\mu \in \mathbb{Q}_B$ with $|1 - \mu\nu| \leq B^{-t}$ in time $O(M(t))$
- Once μ is available, we can set $\gamma = B^{-e-1}\mu$ (again, this merely adjusts the exponent) and observe that we have $|1 - \gamma\beta| = |1 - B^{-e-1}\mu\beta| = |1 - \mu\nu| \leq B^{-t}$, implying that we can indeed without loss of generality work with ν instead of β in what follows

Example: Normalizing the exponent of the divisor

- Let us work in base $B = 10$ for convenience
- Suppose that $\beta = 86295076320 = 8.6295076320 \cdot 10^{10}$ and $t = 6$
- We have $\nu = 0.86295076320 = 86295076320 \cdot 10^{-11}$
- Suppose the near-inverse algorithm outputs $\mu = 1.1588146$ as the near-inverse of ν
- We thus have $\gamma = 1.1588146 \cdot 10^{-11}$
- We can also verify that $|1 - \gamma\beta| = |1 - \mu\nu| = 5.652269728 \cdot 10^{-8} \leq 10^{-6}$

A Newton iteration with truncation (1/2)

- Suppose we have available a $(t+g)$-digit $\mu \in \mathbb{Q}_B$ with $|1-\mu\nu| \leq B^{-t}$
- Here $g \in \mathbb{Z}_{\geq 0}$ is a constant (number of **guard digits**) whose value will be fixed later
- Let $t \in \mathbb{Z}_{\geq 2}$; initially we can assume that $t = 2$
(this needs a preprocessing algorithm; we postpone a discussion)
- We present an $O(M(t))$-time algorithm that computes a $(2t-1+g)$-digit $\hat{\mu} \in \mathbb{Q}_B$ with $|1-\mu\nu| \leq B^{-2t+1}$
- (Iterating this algorithm will produce a desired μ for any ν and t given as input in time $O(M(t))$; we postpone the analysis)

A Newton iteration with truncation (2/2)

- Let us recall that $\hat{\mu} \leftarrow (2 - \mu\nu)\mu$ is the iteration step without truncation
- Recall also that we assume $t \geq 2$ and $|1 - \mu\nu| \leq B^{-t}$ with $B^{-1} \leq \nu < 1$; furthermore, $\mu \in \mathbb{Q}_B$ has $t + g$ digits
- We conclude that $1 - B^{-t} \leq \mu \leq B(1 + B^{-t})$ and thus $(1 - B^{-t})B^{-1} \leq \mu\nu_{\underline{2t-1+g}} < B(1 + B^{-t})$
- Let us study the following iteration step with two truncation operations:

$$\hat{\mu} \leftarrow ((2 - \mu\nu_{\underline{2t-1+g}})\mu)_{\underline{2t-1+g}} \tag{25}$$

- Apart from the truncation operations, the arithmetic in (25) is exact and no intermediate result uses more than $3 + (t + g) + (2t - 1 + g) + (t + g) = 4t + 3g + 2 = O(t)$ digits in base B
- Thus we can compute $\hat{\mu}$ from μ in time $O(M(t))$ as desired

Example: Newton iteration

- Let us work in base $B = 10$
- Suppose we are given as input $t = 32$ and $v = 0.17143811808770734696384501779846951999 2294775$
- From the initialization algorithm (discussed later) we obtain the initial value $\mu = 5.834$
- Applying Newton iteration with truncation (25) with $g = 6$, we observe:

t	g	μ	$v_{\underline{2t-1+g}}$
2	6	5.8340000	0.171438118
3	6	5.83300833	0.17143811808
5	6	5.8330085000	0.171438118087707
9	6	5.83300849982735	0.17143811808770734696384
17	6	5.8330084998273388632428	0.1714381180877073469638450177984695 19992
33	6	5.83300849982733886324269185413958594030	

- Disregarding the g guard digits, we observe μ essentially doubles in length at each step

Convergence analysis with truncation (1/3)

- Let us first introduce parameters δ_1 and δ_2 to quantify the inaccuracy introduced by truncation
- Let $v_{\underline{2t-1+g}} = v + \delta_1$
- Since $B^{-1} \leq v < 1$, we can take $|\delta_1| \leq B^{-1-(2t-1+g)+1} = B^{1-2t-g}$ by (24)
- Let $((2 - \mu v_{\underline{2t-1+g}})\mu)_{\underline{2t-1+g}} = (2 - \mu v_{\underline{2t-1+g}})\mu + \delta_2$
- Since $(2 - \mu v_{\underline{2t-1+g}})\mu \leq 4B \leq B^3$ and $(2 - \mu v_{\underline{2t-1+g}})\mu \geq (2 - \mu(v + \delta_1))(1 - B^{-t}) \geq (1 - B^{-t} - \mu\delta_1)(1 - B^{-t}) \geq (1 - B^{-t} - B(1 + B^{-t})B^{1-2t-g})(1 - B^{-t}) > 0$, we can take $|\delta_2| \leq B^{3-(2t-1+g)+1} = B^{5-2t-g}$ by (24)
- In particular, we can control δ_1 and δ_2 by selection of the constant $g \in \mathbb{Z}_{\geq 0}$
- Let us now proceed to analyze the aggregate convergence ...

Convergence analysis with truncation (2/3)

- Let $\mu\nu = 1 + \delta$ with $|\delta| \leq B^{-t}$
- We have

$$
\begin{aligned}
\hat{\mu}\nu &= ((2 - \mu\nu_{\underline{2t-1+g}})\mu)_{\underline{2t-1+g}}\nu \\
&= ((2 - \mu\nu_{\underline{2t-1+g}})\mu + \delta_2)\nu \\
&= ((2 - \mu(\nu + \delta_1))\mu + \delta_2)\nu \\
&= (2 - \mu\nu)\mu\nu - \mu^2\nu\delta_1 + \nu\delta_2 \\
&= (1 - \delta)(1 + \delta) - \mu^2\nu\delta_1 + \nu\delta_2 \\
&= 1 - \delta^2 - \mu^2\nu\delta_1 + \nu\delta_2
\end{aligned}
$$

Convergence analysis with truncation (3/3)

- Let us recall that $B^{-1} \leq \nu < 1$, $1 - B^{-t} \leq \mu\nu \leq 1 + B^{-t}$, and $\mu \leq B(1 + B^{-t})$
- Furthermore, we recall that $|\delta| \leq B^{-t}$, $|\delta_1| \leq B^{1-2t-g}$, and $|\delta_2| \leq B^{5-2t-g}$
- Thus, also recalling that $B \geq 2$ and $t \geq 2$, we have

$$
\begin{aligned}
|\hat{\mu}\nu - 1| &\leq \delta^2 + \mu^2\nu|\delta_1| + \nu|\delta_2| \\
&\leq \delta^2 + B^2(1 + B^{-t})^2|\delta_1| + |\delta_2| \\
&\leq \delta^2 + B^2(1 + 2B^{-t} + B^{-2t})|\delta_1| + |\delta_2| \\
&\leq \delta^2 + B^3|\delta_1| + |\delta_2| \\
&\leq B^{-2t} + B^{4-2t-g} + B^{5-2t-g} \\
&\leq B^{-2t} + B^{6-2t-g} \\
&\leq B^{-2t+1}
\end{aligned}
$$

where in the last inequality we have used the assumption that $g \geq 6$

Running time (1/2)

- Recall that one step of iteration takes an input μ with $t + g$ digits and produces an output $\hat{\mu}$ of $2t - 1 + g$ digits with $|1 - \mu v| \leq B^{-2t+1}$
- Consider the map $\psi(t) = 2t - 1$
- Starting with the base case $\psi^0(t) = t$, an easy induction shows that the map ψ iterated $k = 0, 1, \ldots$ times yields the map $\psi^k(t) = 2^k t - 2^k + 1$
- At start, we can assume that the initial value to the Newton iteration has $t + g$ digits with $t = 2$ and $g = 6$
- (Indeed, the initialization algorithm will run in time $O(1)$; this will be discussed later)
- Thus, after k steps of iteration, the approximate inverse μ has $\psi^k(t) + g = 2^k t - 2^k + 1 + g$ digits and $|1 - \mu v| \leq B^{-\psi^k(t)}$
- Substituting $t = 2$ and $g = 6$, we obtain that after k steps of iteration μ has $2^k \leq 2^{k+1} - 2^k + 6 \leq 2^{k+4}$ digits and $|1 - \mu v| \leq B^{-\psi^k(2)} \leq B^{-2^k}$

Running time (2/2)

- Let us recall that after k steps of Newton iteration we have at most 2^{k+4} digits in μ, and $|1-\mu\nu| \leq B^{-2^k}$
- Thus, for a $t \in \mathbb{Z}_{\geq 1}$ given as input, to obtain an approximate inverse μ with $|1-\mu\nu| \leq B^{-t}$, it suffices to run $k = \lceil \log_2 t \rceil$ steps
- We observe that arithmetic during step k works with intermediate results that have at most $4 \cdot 2^{k+6} + 3 \cdot 6 + 2 = O(2^k)$ digits
- Furthermore, since the multiplication time $M(d)$ **grows at most polynomially** in d; that is, for all constants $C \geq 1$ there exists a constant $C' \geq 1$ such that for all $d = 1, 2, \ldots$ we have $M(Cd) \leq C'M(d)$, the running time of step k is $O(M(2^k))$
- By Lemma 5, the total running time to produce approximate inverse μ with $|1-\mu\nu| \leq B^{-t}$ is $O(M(2^{\lceil \log_2 t \rceil + 1}))$, which is $O(M(t))$

Summary—fast integer division (1/2)

- Let integers $\alpha, \beta \in \mathbb{Z}_{\geq 1}$ be given as input in base B

1. Normalize β to $\nu \in \mathbb{Q}_B$ with $B^{-1} \leq \nu < 1$ by adjusting the exponent
2. From ν determine a $(2+g)$-digit initial approximation $\mu \in \mathbb{Q}_B$ with $|1 - \mu\nu| \leq B^{-2}$ (this will be discussed in what follows)
3. Run the Newton iteration with truncation (25) until we have a $(t+g)$-digit approximation $\mu \in \mathbb{Q}_B$ with $|1 - \mu\nu| \leq B^{-t}$ for t large enough
4. Adjust the exponent of μ to obtain $\gamma \in \mathbb{Q}_B$ with $|1 - \gamma\beta| \leq B^{-t}$
5. Recover the quotient $\eta = \alpha \operatorname{quo} \beta$ using the approximate quotient $\tilde{\eta} = \alpha\gamma$
6. Compute the remainder $\rho = \alpha \operatorname{rem} \beta = \alpha - \eta\beta$

- (We leave the details of quotient recovery for the exercises)

Summary—fast integer division (2/2)

- The present algorithm runs in $O(M(n))$ time for two at-most-n-digit integers $\alpha, \beta \in \mathbb{Z}_{\geq 1}$ in base B given as input, $B = O(1)$
- However, the algorithm has not been optimized for practical performance (for example, for a specific choice of B such as $B = 2^{64}$)
- Considerable further work would be needed to optimize for a practical implementation (cf. Brent and Zimmermann [4] for a starting point)

Example: Fast integer division (1/2)

- Let us work in base $B = 10$
- Suppose the given input is

$$\alpha = 1866830377857904687585481026334265282048899060517697915942019834534476682181$$

$$\beta = 1714381180877073469638450177984695199922947 75$$

1. Normalizing β, we obtain

$$\nu = 0.171438118087707346963845017798469519992294775$$

2. The initialization algorithm for $t = 2$ gives $\mu = 5.834$
3. Running Newton iteration with $g = 6$ and $t = 32$ gives

$$\mu = 5.833008499827338863242691854139585940 30$$

4. Adjusting the exponent gives

$$\gamma = 5.83300849982733886324269185413958594030 \cdot 10^{-45}$$

Example: Fast integer division (2/2)

5. The approximate quotient is thus

$$\tilde{\eta} = \alpha\gamma = 1088923746178104077970193438116 6.79300360663554935084051345 \quad \backslash\backslash$$
$$18273843794550524803513473907034720060209940246591397943$$

   from which we recover the quotient

$$\eta = 10889237461781040779701934381166$$

6. Finally we compute the remainder

$$\rho = \alpha - \eta\beta = 13595104275066478629269766068561155659647453 1$$

Extra: Initial approximation (1/3)

- To complete the algorithm design, we still need an initial value for the Newton iteration
- In precise terms, given $t \in \mathbb{Z}_{\geq 1}$ and $v \in \mathbb{Q}_B$ with $B^{-1} \leq v < 1$ as input, we need a $(t + g)$-digit $\mu \in \mathbb{Q}_B$ with $|1 - \mu v| \leq B^{-t}$ for some fixed constant $g \in \mathbb{Z}_{\geq 1}$
- The initial approximation needs only constant values of t and g;
  for example, already $t = 2$ suffices to initialize our Newton iteration
- Accordingly, we need not be particularly efficient with the initialization
  (though a practical implementation would carefully optimize this step too)
- For illustration, let us reduce initialization to integer division (which can be solved, for example, with the classical integer division algorithm since t and g are constants)

Extra: Initial approximation (2/3)

- Let $t \in \mathbb{Z}_{\geq 1}$ and $v \in \mathbb{Q}_B$ with $B^{-1} \leq v < 1$ be given as input
- Let a, ℓ, k be parameters whose values we fix in what follows

1. Set $\alpha = B^a$ and $\beta = (B^{k+1}v)_{\underline{\ell}}$ with $1 \leq \ell \leq k+1$ and $a \geq 0$ so that $\alpha, \beta \in \mathbb{Z}$
2. Run classical integer division to obtain $\eta, \rho \in \mathbb{Z}_{\geq 0}$ with $\alpha = \eta\beta + \rho$ and $0 \leq \rho \leq \beta - 1$
3. Return the initial approximation $\mu = B^{-a+k+1}\eta$

- Let us now analyze the accuracy of μ and the number of digits in μ
- Since $1 \leq \ell \leq k+1$ and $B^{-1} \leq v < 1$, we have that $\beta = (B^{k+1}v)_{\underline{\ell}} = B^{k+1}v_{\underline{\ell}}$
- Let $v_{\underline{\ell}} = v + \delta$ and observe that $|\delta| \leq B^{-\ell}$ by $B^{-1} \leq v < 1$ and (24)
- Since $B^{-1} \leq v < 1$, we have $B^k \leq \beta \leq B^{k+1} - 1$

Extra: Initial approximation (3/3)

- Recall that $\alpha = B^a$, $\beta = B^{k+1}(\nu + \delta)$ with $|\delta| \le B^{-\ell}$, and $\mu = B^{-a+k+1}\eta$
- Multiply both sides of $\alpha = \eta\beta + \rho$ by B^{-a} to conclude that $1 = \mu(\nu + \delta) + B^{-a}\rho$
- Recalling that $0 \le \rho \le \beta - 1$ and that $B^k \le \beta \le B^{k+1}$, we conclude that $|1 - \mu\nu| \le \delta\mu + B^{-a+k+1}$
- We have $0 \le \mu \le B^{-a+k+1}\alpha/\beta \le B$, implying that $|1 - \mu\nu| \le B^{-\ell+1} + B^{-a+k+1}$
- Now set $a = 2t + 3$, $k = t + 1$, and $\ell = t + 2$
- Since $B \ge 2$ we conclude that $|1 - \mu\nu| \le B^{-t-1} + B^{-t-1} \le B^{-t}$
- Since $\alpha = B^a$ and $B^k \le \beta \le B^{k+1} - 1$, we have $B^{a-k-1} \le \alpha/\beta \le B^{a-k}$, and thus $\eta = \lfloor \alpha/\beta \rfloor$ (and hence $\mu = B^{-a+k+1}\eta$) has at most $a - k + 1 = t + 3$ digits
- Accordingly we can take $g = 3$ to complete the initial approximation algorithm; using classical division, this algorithm runs in $O(t^2)$ time for $B = O(1)$, but we only apply it for inputs of size $t = O(1)$, such as $t = 2$ to initialize our Newton iteration

Extra: Initial approximation with a look-up table

- Recall that we assume that the base B is a constant
- Since constant t and g suffice, we observe that the parameters $a = 2t + 3$, $k = t + 1$, and $\ell = t + 2$ are also constants
- Since $\alpha = B^a$ is a constant, $\beta = B^{k+1} v_{\underline{\ell}} = B^{t+2} v_{\underline{t+2}}$ suffices to determine the initial approximation μ
- Since $v_{\underline{t+2}}$ has $t + 2$ digits, the first of which is nonzero, we can prepare a **look-up table** with $(B - 1)B^{t+1}$ entries for use in initialization
- That is, using the $t + 2$ most significant digits of v as an index, we consult the look-up table for a valid initialization μ (which has at most $t + 3$ digits)
- For example, when $B = 2$ and $t = 2$, it suffices to have a look-up table with $(2 - 1)2^3 = 8$ entries, where each entry has at most 5 digits (that is, bits, since $B = 2$)

Example: Initial approximation

- Let us work in base $B = 10$
- Suppose the given input is $t = 2$ together with

$$\nu = 0.171438118087707346963845017798469519992294775$$

- Following the initialization algorithm, we set

$$\begin{aligned} \alpha &= 10000000 \\ \beta &= 1714 \end{aligned}$$

  and thus obtain the quotient $\eta = \lfloor \alpha / \beta \rfloor = 5834$ and hence the initial value $\mu = 5.834$

- In particular, we use only $t + 2 = 4$ first digits of ν to obtain μ

Example: Look-up table for initialization

- For $B = 2$ and $t = 2$, we obtain the following look-up table for initializing the Newton iteration so that $|1 - \mu v| \leq B^{-t} = 1/4$:

$v_{\underline{4}}$	μ
0.1000	10
0.1001	1.11
0.1010	1.1
0.1011	1.011
0.1100	1.01
0.1101	1.001
0.1110	1.001
0.1111	1

- In particular, we use only the first $t + 2 = 4$ digits of v to obtain μ

Key content recalled

- **Division** (**quotient** and **remainder**) for integers and polynomials
- Fast division by **reduction** to fast multiplication
- Integer division via **approximation** of the multiplicative inverse of the divisor
- The radix-point representation and approximation of rational numbers
- **Newton iteration**
- Newton iteration for the multiplicative inverse of the divisor
- **Convergence analysis** for Newton iteration
- Polynomial division via **reversal**
- Newton iteration for the inverse of the reverse of the divisor

Goal for fast polynomial division

- Let R be a ring
- Let $a, b \in R[x]$ with b monic and $d \geq \deg a \geq \deg b$ for some $d \in \mathbb{Z}_{\geq 0}$
- We want an algorithm that computes the quotient q and the remainder r in the division of a by b in $O(M(d))$ operations in R
- Here $M(d) = O(d \log d)$ or $M(d) = O(d \log d \log\log d)$ depending on R

First reduction recalled: the quotient suffices

- Division (viewed from 36,000ft, see earlier slides for details):

$$\text{Given } a, b \text{ we need to compute } q, r \text{ such that } a = qb + r$$

- Observation:
  It suffices to compute q since then we can recover $r = a - qb$ by fast multiplication

Reversal to recover the quotient

- For a polynomial

$$f = \varphi_0 + \varphi_1 x + \varphi_2 x^2 + \ldots + \varphi_n x^n$$

of degree at most $n \in \mathbb{Z}_{\geq 0}$, the n-**reversal** of f is the polynomial

$$\operatorname{rev}_n f = \varphi_n + \varphi_{n-1} x + \varphi_{n-2} x^2 + \ldots + \varphi_0 x^n$$

- For the quotient-and-remainder identity $a = qb + r$ with $\deg a = n \geq m = \deg b$ and $\deg r \leq m - 1$, we observe (exercise) that the reversal operator satisfies

$$\operatorname{rev}_n a = (\operatorname{rev}_{n-m} q)(\operatorname{rev}_m b) + x^{n-m+1} \operatorname{rev}_{m-1} r$$

- In particular, working in the factor ring relative to the ideal $\langle x^{n-m+1} \rangle$,

$$\operatorname{rev}_n a \equiv (\operatorname{rev}_{n-m} q)(\operatorname{rev}_m b) \pmod{x^{n-m+1}}$$

- We can thus compute the quotient q by computing the multiplicative inverse of $\operatorname{rev}_m b$ modulo x^{n-m+1} (we will show this inverse exists because b is monic), multiplying by $\operatorname{rev}_n a$, and $(n - m)$-reversing the result to obtain q

Example: Reversal (1/2)

- Suppose that in $\mathbb{Z}_5[x]$ we have

$$a = 3 + 3x + x^2 + 2x^3 + x^4 + 4x^6 + x^7 + 3x^8 + 4x^9 + 3x^{10} + x^{11} + x^{12}$$
$$b = 2 + x + x^2 + 3x^3 + 3x^4 + 3x^5 + x^6$$

  with $n = \deg a = 12$ and $m = \deg b = 6$; we also observe that b is monic

- We have $a = qb + r$ and $0 \leq \deg r \leq \deg b - 1$ for

$$q = 3 + 3x + 3x^2 + 4x^3 + x^4 + 3x^5 + x^6$$
$$r = 2 + 4x + 4x^2 + 4x^3 + 4x^4 + 2x^5$$

- Taking reverses, we have

$$\operatorname{rev}_n a = 1 + x + 3x^2 + 4x^3 + 3x^4 + x^5 + 4x^6 + x^8 + 2x^9 + x^{10} + 3x^{11} + 3x^{12}$$
$$\operatorname{rev}_m b = 1 + 3x + 3x^2 + 3x^3 + x^4 + x^5 + 2x^6$$
$$\operatorname{rev}_{n-m} q = 1 + 3x + x^2 + 4x^3 + 3x^4 + 3x^5 + 3x^6$$
$$\operatorname{rev}_{m-1} r = 2 + 4x + 4x^2 + 4x^3 + 4x^4 + 2x^5$$

Example: Reversal (2/2)

- Recalling that

$$
\begin{aligned}
\operatorname{rev}_n a &= 1 + x + 3x^2 + 4x^3 + 3x^4 + x^5 + 4x^6 + x^8 + 2x^9 + x^{10} + 3x^{11} + 3x^{12} \\
\operatorname{rev}_m b &= 1 + 3x + 3x^2 + 3x^3 + x^4 + x^5 + 2x^6 \\
\operatorname{rev}_{n-m} q &= 1 + 3x + x^2 + 4x^3 + 3x^4 + 3x^5 + 3x^6 \\
\operatorname{rev}_{m-1} r &= 2 + 4x + 4x^2 + 4x^3 + 4x^4 + 2x^5
\end{aligned}
$$

  with $n = 12$ and $m = 5$, we can now verify the reversed division equality

$$\operatorname{rev}_n a = (\operatorname{rev}_{n-m} q)(\operatorname{rev}_m b) + x^{n-m-1} \operatorname{rev}_{m-1} r$$

- Indeed,

$$
\begin{aligned}
\operatorname{rev}_n a &= 1 + x + 3x^2 + 4x^3 + 3x^4 + x^5 + 4x^6 + x^8 + 2x^9 + x^{10} + 3x^{11} + 3x^{12} \\
(\operatorname{rev}_{n-m} q)(\operatorname{rev}_m b) &= 1 + x + 3x^2 + 4x^3 + 3x^4 + x^5 + 4x^6 + 3x^7 + 2x^8 + 3x^9 + 2x^{10} + 4x^{11} + x^{12} \\
x^{n-m-1} r &= 2x^7 + 4x^8 + 4x^9 + 4x^{10} + 4x^{11} + 2x^{12}
\end{aligned}
$$

The inverse modulo x^d by reduction to fast multiplication

- Let $g = \sum_j \psi_j x^j \in R[x]$ with $\psi_0 = 1$ be given as input
- We set up a Newton iteration that doubles d at every step
- Assume inductively that $f \in R[x]$ satisfies $fg \equiv 1 \pmod{x^{2^k}}$ for $k \in \mathbb{Z}_{\geq 0}$
- To set up the base case $k = 0$, take $f = 1$ and observe that the assumption holds
- Compute $\hat{f} \equiv (2 - fg)f \pmod{x^{2^{k+1}}}$ using fast multiplication, truncating both g and $\hat{f}$ using the substitution $x^{2^{k+1}} = 0$
- Since the assumption holds for f with parameter value k, there exists a $h \in R[x]$ with $fg = 1 + x^{2^k} h$
- We observe that $\hat{f}g \equiv (2 - fg)fg \equiv (1 - x^{2^k} h)(1 + x^{2^k} h) \equiv 1 \pmod{x^{2^{k+1}}}$ and thus the assumption holds for $\hat{f}$ with parameter value $k + 1$
- The cost of step k is $O(M(2^k))$ since M grows at most polynomially; by Lemma 5 the total cost is $O(M(d))$ operations in R

Example: Iterating for the inverse modulo x^d

- Let $g = 1 + 3x + 3x^2 + 3x^3 + x^4 + x^5 + 2x^6 \in \mathbb{Z}_5[x]$
- Let us compute the multiplicative inverse of g modulo x^d for $d = 7$
- The least integer k for which $2^k \geq d$ is $k = 3$, so we need three rounds of Newton iteration
- Truncating g and $\hat{f}$ by setting $x^{2^{k+1}} = 0$ and iterating, we have

k	f	g
0	1	$1 + 3x$
1	$1 + 2x$	$1 + 3x + 3x^2 + 3x^3$
2	$1 + 2x + x^2 + 3x^3$	$1 + 3x + 3x^2 + 3x^3 + x^4 + x^5 + 2x^6$
3	$1 + 2x + x^2 + 3x^3 + x^4 + 2x^5 + 2x^6 + 2x^7$	

- Thus, the multiplicative inverse of g modulo x^d is

$$1 + 2x + x^2 + 3x^3 + x^4 + 2x^5 + 2x^6$$

Example: Division with reversal and Newton iteration

- Suppose that in $\mathbb{Z}_5[x]$ we have

$$a = 3 + 3x + x^2 + 2x^3 + x^4 + 4x^6 + x^7 + 3x^8 + 4x^9 + 3x^{10} + x^{11} + x^{12}$$
$$b = 2 + x + x^2 + 3x^3 + 3x^4 + 3x^5 + x^6$$

  with $n = \deg a = 12$ and $m = \deg b = 6$; we also observe that b is monic

- Reverse a and b to obtain

$$\operatorname{rev}_n a = 1 + x + 3x^2 + 4x^3 + 3x^4 + x^5 + 4x^6 + x^8 + 2x^9 + x^{10} + 3x^{11} + 3x^{12}$$
$$\operatorname{rev}_m b = 1 + 3x + 3x^2 + 3x^3 + x^4 + x^5 + 2x^6$$

- Iterate for the inverse f of $\operatorname{rev}_m b$ modulo x^{n-m+1} to obtain

$$f = 1 + 2x + x^2 + 3x^3 + x^4 + 2x^5 + 2x^6$$

- Compute $f \operatorname{rev}_n a$, truncate with $x^{n-m+1} = 0$, and $(n-m)$-reverse the result to obtain the quotient $q = 3 + 3x + 3x^2 + 4x^3 + x^4 + 3x^5 + x^6$
- Compute the remainder $r = a - qb = 2 + 4x + 4x^2 + 4x^3 + 4x^4 + 2x^5$

Summary—fast polynomial division

- Let R be a ring
- Let $a, b \in R[x]$ with b monic and $d \geq \deg a \geq \deg b$ for some $d \in \mathbb{Z}_{\geq 0}$
- We have an algorithm that computes the quotient q and the remainder r in the division of a by b in $O(M(d))$ operations in R
  1. Let $n = \deg a$ and $m = \deg b$
  2. m-reverse b and compute the multiplicative inverse of $\mathrm{rev}_m\, b$ modulo x^{n-m+1} using Newton iteration, multiply by the result by $\mathrm{rev}_n\, a$ modulo x^{n-m+1}, and $(n-m)$-reverse the result to obtain the quotient q
  3. Compute remainder r by $r = a - qb$
- Here $M(d) = O(d \log d)$ or $M(d) = O(d \log d \log \log d)$ depending on R

Recap of key content for Lecture 3

- **Division** (**quotient** and **remainder**) for integers and polynomials
- Fast division by **reduction** to fast multiplication
- Integer division via **approximation** of the multiplicative inverse of the divisor
- The radix-point representation and approximation of rational numbers
- **Newton iteration**
- Newton iteration for the multiplicative inverse of the divisor
- **Convergence analysis** for Newton iteration
- Polynomial division via **reversal**
- Newton iteration for the inverse of the reverse of the divisor

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