Fast algorithms for polynomials and matrices
(A brief introduction to Computer Algebra)
— Part 1 —

Alin Bostan

SpecFun, INRIA

Seminar on Algebraic Systems Theory
April 4, 2013

General framework

Computer algebra = effective mathematics and algebraic complexity

- Effective mathematics: what can we compute?
- their complexity: how fast?

Mathematical Objects

- Main objects
  - polynomials $\mathbb{K}[x]$
  - rational functions $\mathbb{K}(x)$
  - power series $\mathbb{K}[[x]]$
  - matrices $\mathcal{M}_r(\mathbb{K})$
  - polynomial matrices $\mathcal{M}_r(\mathbb{K}[x])$
  - power series matrices $\mathcal{M}_r(\mathbb{K}[[x]])$

  where $\mathbb{K}$ is a field (generally assumed of characteristic 0, or large enough)

- Secondary/auxiliary objects
  - linear recurrences with constant, or polynomial, coefficients $\mathbb{K}[n]\langle S_n\rangle$
  - linear differential equations with polynomial coefficients $\mathbb{K}[x]\langle \partial_x\rangle$

This course

- Aims
  - design and analysis of fast algorithms for various algebraic problems
  - Fast = using asymptotically few operations $(+, \times, \div)$ in the basefield $\mathbb{K}$
  - Holy Grail: quasi-optimal algorithms = (time) complexity almost linear in the input/output size

- Specific algorithms depending on the kind of the input
  - dense (i.e., arbitrary)
  - structured (i.e., special relations between coefficients)
  - sparse (i.e., few elements)

- In this lecture, we focus on dense objects

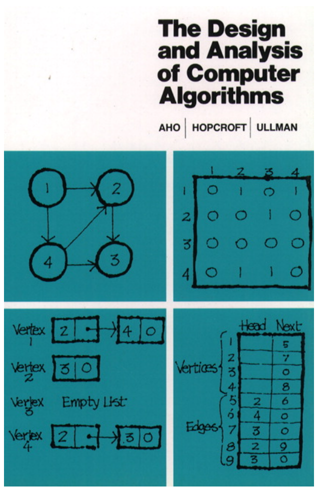
A word about structure and sparsity

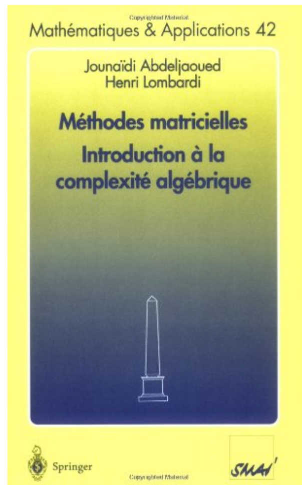
- sparse means
  - for degree n polynomials: $s \ll n$ coefficients
  - for $r \times r$ matrices: $s \ll r^2$ entries

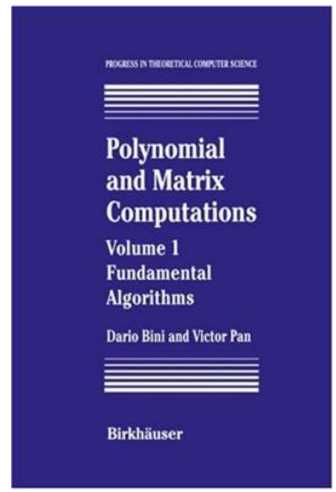
- structured means
  - for $r \times r$ matrices: special form, e.g., Toeplitz, Hankel, Vandermonde, Cauchy, Sylvester, etc) $\longrightarrow$ encoded by $O(r)$ elements
  - for polynomials/power series: satisfying an equation (algebraic or differential) $\longrightarrow$ encoded by degree/order of size $O(1)$

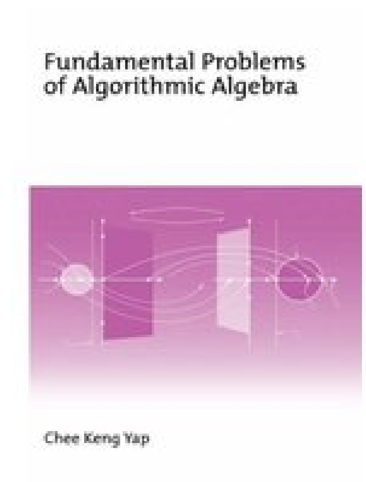
- In this lecture, we focus on dense objects

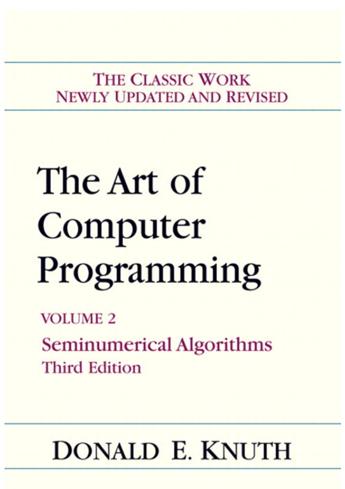
Computer algebra books

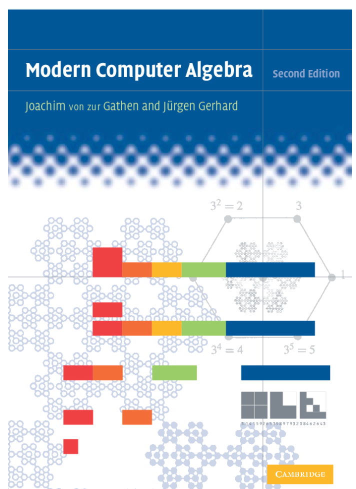


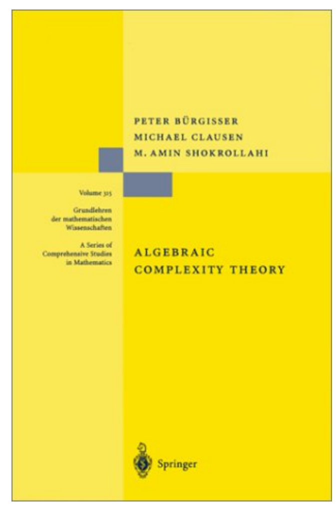


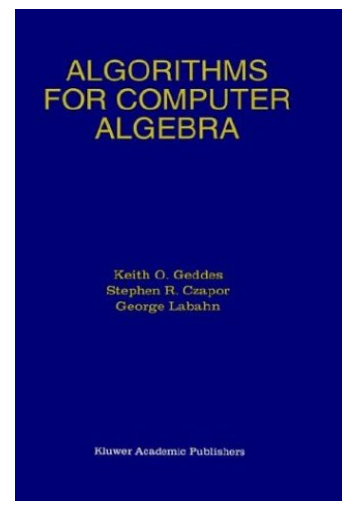












Complexity yardsticks

Important features:

- addition is easy: naive algorithm already optimal
- multiplication is the most basic (non-trivial) problem
- almost all problems can be reduced to multiplication

Are there quasi-optimal algorithms for:

- integer/polynomial/power series multiplication? Yes!
- matrix multiplication? Big open problem!

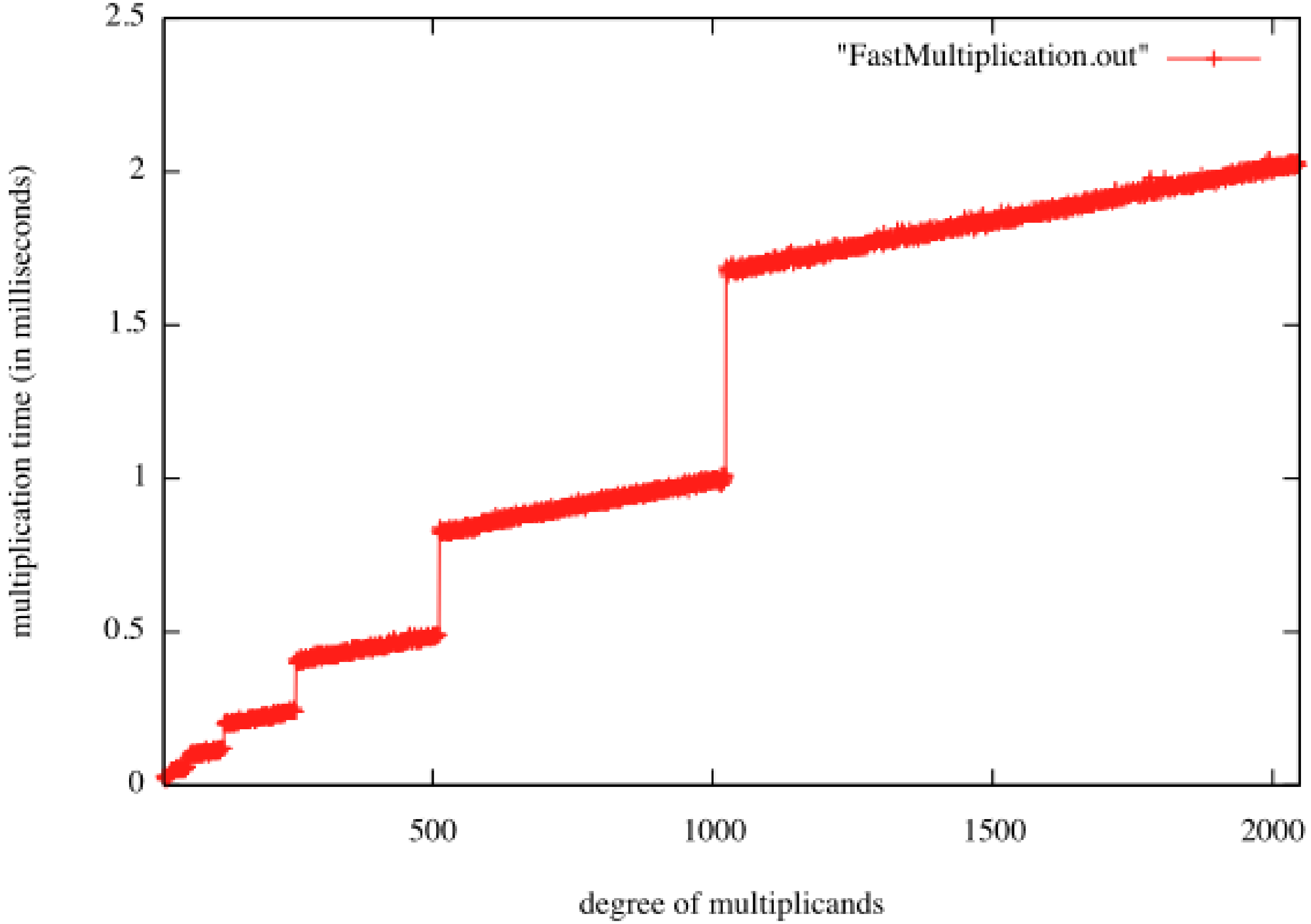
Complexity yardsticks

$$
\begin{aligned}
\mathsf{M}(n) &= \text{complexity of polynomial multiplication in } \mathbb{K}[x]_{<n}\\
&= O(n^2) \text{ by the naive algorithm}\\
&= O\big(n^{1.58}\big) \text{ by Karatsuba's algorithm}\\
&= O\big(n^{\log_\alpha (2\alpha-1)}\big) \text{ by the Toom-Cook algorithm } (\alpha \geq 3)\\
&= O\big(n \log n \operatorname{loglog} n\big) \text{ by the Schönhage-Strassen algorithm}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{MM}(r) &= \text{complexity of matrix product in } \mathcal{M}_r(\mathbb{K})\\
&= O(r^3) \text{ by the naive algorithm}\\
&= O(r^{2.81}) \text{ by Strassen's algorithm}\\
&= O(r^{2.38}) \text{ by the Coppersmith-Winograd algorithm}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{MM}(r,n) &= \text{complexity of polynomial matrix product in } \mathcal{M}_r(\mathbb{K}[x]_{<n})\\
&= O(r^3\, \mathsf{M}(n)) \text{ by the naive algorithm}\\
&= O(\mathsf{MM}(r)\, n \log(n) + r^2 n \log n \operatorname{loglog} n) \text{ by the Cantor-Kaltofen algo}\\
&= O(\mathsf{MM}(r)\, n + r^2\, \mathsf{M}(n)) \text{ by the B-Schost algorithm}
\end{aligned}
$$

Fast polynomial multiplication in practice



Practical complexity of Magma's multiplication in $\mathbb{F}_p[x]$, for $p = 29 \times 2^{57} + 1$.

What can be computed in 1 minute with a CA system*

polynomial product† in degree 14,000,000 (>1 year with schoolbook)

product of two integers with 500,000,000 binary digits

factorial of $N = 20,000,000$ (output of 140,000,000 digits)

gcd of two polynomials of degree 600,000

resultant of two polynomials of degree 40,000

factorization of a univariate polynomial of degree 4,000

factorization of a bivariate polynomial of total degree 500

resultant of two bivariate polynomials of total degree 100 (output 10,000)

product/sum of two algebraic numbers of degree 450 (output 200,000)

determinant (char. polynomial) of a matrix with 4,500 (2,000) rows

determinant of an integer matrix with 32-bit entries and 700 rows

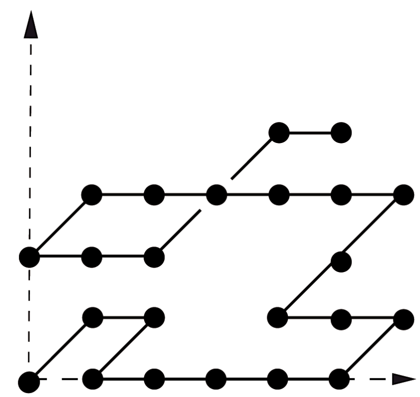
---

*on a PC, (Intel Xeon X5160, 3GHz processor, with 8GB RAM), running Magma V2.16-7

†in $\mathbb{K}[x]$, for $\mathbb{K} = \mathbb{F}_{67108879}$

A recent application: Gessel's conjecture

- Gessel walks: walks in $\mathbb{N}^2$ using only steps in $\mathcal{S} = \{\nearrow, \swarrow, \leftarrow, \rightarrow\}$
- $g(i,j,n)$ = number of walks from $(0,0)$ to (i,j) with n steps in $\mathcal{S}$



Question: Nature of the generating function

$$G(x,y,t) = \sum_{i,j,n=0}^{\infty} g(i,j,n)\, x^i y^j t^n \in \mathbb{Q}[[x,y,t]]$$

▶ Computer algebra conjectures and proves:

Theorem [B. & Kauers 2010] $G(x,y,t)$ is an algebraic function† and

$$G(1,1,t) = \frac{1}{2t} \cdot {}_2F_1\left(\begin{matrix} -1/12 & 1/4 \\ & 2/3 \end{matrix} \,\middle|\, -\frac{64t(4t+1)^2}{(4t-1)^4}\right) - \frac{1}{2t}.$$

▶ No human proof yet.

---

† Minimal polynomial $P(x,y,t,G(x,y,t)) = 0$ has $> 10^{11}$ monomials; $\approx$30Gb (!)

Mathematical Objects

- Main objects
  - polynomials $\mathbb{K}[x]$
  - rational functions $\mathbb{K}(x)$
  - power series $\mathbb{K}[[x]]$
  - matrices $\mathcal{M}_r(\mathbb{K})$
  - polynomial matrices $\mathcal{M}_r(\mathbb{K}[x])$
  - power series matrices $\mathcal{M}_r(\mathbb{K}[[x]])$

  where $\mathbb{K}$ is a field (generally assumed of characteristic 0, or large enough)

- Secondary/auxiliary objects
  - linear recurrences with constant, or polynomial, coefficients $\mathbb{K}[n]\langle S_n \rangle$
  - linear differential equations with polynomial coefficients $\mathbb{K}[x]\langle \partial_x \rangle$

Typical problems

- On all objects
  - sum, product
  - inversion, division
- On power series
  - logarithm, exponential
  - composition
  - Padé and Hermite-Padé approximation
- On polynomials
  - (multipoint) evaluation, interpolation
  - (extended) greatest commun divisor, resultant
  - shift
  - composed sum and product
- On matrices
  - system solving
  - determinant, characteristic polynomial

Typical problems, and their complexities

- Polynomials, power series and matrices
  - product $\mathsf{M}(n)$, $\mathsf{MM}(r)$
  - division/inversion $O(\mathsf{M}(n))$, $O(\mathsf{MM}(r))$
- On power series
  - logarithm, exponential $O(\mathsf{M}(n))$
  - composition $O(\sqrt{n \log n}\, \mathsf{M}(n))$
  - Padé approximation $O(\mathsf{M}(n) \log n)$
- On polynomials
  - (multipoint) evaluation, interpolation $O(\mathsf{M}(n) \log n)$
  - extended gcd, resultant $O(\mathsf{M}(n) \log n)$
  - shift $O(\mathsf{M}(n))$
  - composed sum and product $O(\mathsf{M}(n))$
- On matrices
  - system solving, determinant $O(\mathsf{MM}(r))$
  - characteristic / minimal polynomial $O(\mathsf{MM}(r))$

Typical problems, and the algorithms' designers

- Polynomials, power series and matrices
  - product
  - division/inversion — Sieveking-Kung 1972, Strassen 1969, 1973
- On power series
  - logarithm, exponential — Brent 1975
  - composition — Brent-Kung 1978
  - Padé approximation — Brent-Gustavson-Yun 1980
- On polynomials
  - (multipoint) evaluation, interpolation — Borodin-Moenck 1974
  - extended gcd, resultant — Knuth-Schönhage 1971, Schwartz 1980
  - shift — Aho-Steiglitz-Ullman 1975
  - composed sum and product — B-Flajolet-Salvy-Schost 2006
- On matrices
  - system solving, determinant — Strassen 1969
  - characteristic polynomial / minimal polynomial — Keller-Gehrig 1985

Typical problems, and their complexities

- On power series matrices
  - product $\mathsf{MM}(r, n)$
  - inversion $O(\mathsf{MM}(r, n))$
  - quasi-exponential (sol. of $Y' = AY$) $O(\mathsf{MM}(r, n))$
- On power series
  - Hermite-Padé approximation of r series $O(\mathsf{MM}(r, n) \log n)$
- On polynomial matrices
  - product $\mathsf{MM}(r, n)$
  - system solving $O(\mathsf{MM}(r, n) \log n)$
  - determinant $O(\mathsf{MM}(r, n) \log^2(n))$
  - inversion $\tilde{O}(r^3\, n)$, if $r = 2^k$
  - characteristic / minimal polynomial $\tilde{O}(r^{2.6972}\, n)$

Typical problems, and the algorithms' designers

- On power series matrices
  - product
  - inversion — Schulz 1933
  - quasi-exponential — B-Chyzak-Ollivier-Salvy-Schost-Sedoglavic 2007
- On power series
  - Hermite-Padé approximation — Beckermann-Labahn 1994
- On polynomial matrices
  - product
  - system solving — Storjohann 2002
  - determinant — Storjohann 2002
  - inversion — Jeannerod-Villard 2005
  - characteristic / minimal polynomial — Kaltofen-Villard 2004

Other problems, and their complexities

- On structured (D-finite, algebraic) power series
  - sum, product, Hadamard product — $O(n)$
  - inversion — $O(\mathsf{M}(n))$, $O(n)$
- On structured matrices
  - Toeplitz-like: system solving, determinant — $O(\mathsf{M}(r)\log r)$
  - Vandermonde-like: system solving, determinant — $O(\mathsf{M}(r)\log^2(r))$
  - Cauchy-like: system solving, determinant — $O(\mathsf{M}(r)\log^2(r))$
- On sparse matrices
  - system solving — $O(r^2)$
  - determinant — $O(r^2)$
  - rank — $O(r^2)$
  - minimal polynomial — $O(r^2)$

Other problems, and their complexities

- On structured (D-finite, algebraic) power series
  - sum, product, Hadamard product — folklore, but not sufficiently known!
  - inversion
- On structured matrices
  - Toeplitz-like: system solving, determinant — Bitmead-Anderson-Morf 1980
  - Vandermonde-like: system solving, determinant — Pan 1990
  - Cauchy-like: system solving, determinant — Pan 2000
- On sparse matrices
  - system solving — Wiedemann 1986
  - determinant — Wiedemann 1986
  - rank — Kaltofen-Saunders 1991
  - minimal polynomial — Wiedemann 1986

Algorithmic paradigms

Given a problem, how to find an efficient algorithm for its solution?

Five paradigms for algorithmic design

- divide and conquer (DAC)
- decrease and conquer (dac)
- baby steps / giant steps (BS-GS)
- change of representation (CR)
- Tellegen's transposition principle (Tellegen)

Algorithmic paradigms, and main techniques

Given a problem, how to find an efficient algorithm for its solution?

Five paradigms for algorithmic design

- divide and conquer
- decrease and conquer
  - binary powering
  - Newton iteration
  - Keller-Gehrig iteration
- baby steps / giant steps
- change of representation
  - evaluation-interpolation
  - expansion-reconstruction
- Tellegen's transposition principle

Divide and conquer

Idea: recursively break down a problem into two or more similar subproblems, solve them, and combine their solutions

Origin: unknown, probably very ancient.
Modern form: merge sort algorithm — von Neumann 1945

Our main examples:

- Karatsuba algorithm — polynomial multiplication
- Strassen algorithm — matrix product
- Strassen algorithm — matrix inversion
- Borodin-Moenck algorithm — polynomial evaluation-interpolation
- Beckermann-Labahn algorithm — Hermite-Padé approximation
- Bitmead-Anderson-Morf algorithm — solving Toeplitz-like linear systems
- Lehmer-Knuth-Schönhage-Moenck-Strassen algorithm — extended gcd

Decrease and conquer

Idea: reduce each problem to only one similar subproblem of half size

Origin: probably Pingala's Hindu classic Chandah-sutra, 200 BC

Modern form: binary search algorithm — Mauchly 1946

Our main examples:

- binary powering — exponentiation in rings
- modular exponentiation — exponentiation in quotient rings
  - N-th term of a recurrence with constant coefficients
- Newton iteration — power series root-finding
  - polynomial division
  - composed sum and product
  - polynomial shift
- Kehler-Gehrig algorithm — Krylov sequence computation
- Storjohann's high order lifting algorithm — polynomial matrices
- B-Schost algorithm — interpolation on geometric sequences

Baby steps / giant steps

Idea: reduce a problem of size N to two similar subproblem of size $\sqrt{N}$

Origin: computational number theory, ≈ 1960

Modern form: discrete logarithm problem — Shanks 1969

Our main examples:

- Paterson-Stockmeyer 1973 — polynomial evaluation in an algebra
- Strassen 1976 — deterministic integer factorization
- Brent-Kung 1978 — composition of power series
- Chudnovsky-Chudnovsky 1987 — N-th term of a P-recursive sequence
  - point counting on hyperelliptic curves
  - polynomial solutions of linear differential equations
  - p-curvature of linear differential operators
- Shoup 1995 — power projection $[\ell(1), \ell(u), \ldots, \ell(u^{N-1})]$

Change of representation

Idea: represent objects in a different way, mathematically equivalent, but better suited for the algorithmic treatment

Origin: unknown, probably Sun Zi ≈ 300 (Chinese remainder theorem)
Modern form: the Czech number system — Svoboda-Valach 1955

Our main examples: One can represent

- a polynomial by
  - the list of its coefficients
  - the values it takes at sufficiently many points — easy $\times$
  - its Newton sums (= powersums of roots) — easy $\otimes, \oplus$
- a rational fraction by
  - the coefficient lists of its denominator and numerator
  - its values at sufficiently many points
  - its Taylor series expansion

Tellegen's transposition principle

Idea: to solve a linear problem, find an algorithm for its dual, and transpose it

Origin: electrical network theory: Tellegen, Bordewijk, $\approx$ 1950

Modern form: transposition of algorithms, complexity version — Fiduccia 1972

Our main examples:

- improve algorithms by constant factors
  - Hanrot-Quercia-Zimmermann 2002 — middle product for polynomials
  - B-Lecerf-Schost 2003 — multipoint evaluation and interpolation
- prove computational equivalence between problems
  - B-Schost 2004 — multipoint evaluation $\Leftrightarrow$ interpolation
- discover new algorithms
  - B-Salvy-Schost 2008 — base conversions
- understand (connections between) existing algorithms
  - DFT: decimation in time vs. decimation in frequency
  - Strassen's polynomial division vs. Shoup's extension of recurrences

The Master Theorem

Suppose that the complexity $\mathsf{C}(n)$ of an algorithm satisfies

$$\mathsf{C}(n) \le s \cdot \mathsf{C}\left(\frac{n}{2}\right) + \mathsf{T}(n),$$

where the function T is such that $q\mathsf{T}(n) \le \mathsf{T}(2n)$. Then, for $n \to \infty$

$$\mathsf{C}(n) \;=\; \begin{cases} O(\mathsf{T}(n)), & \text{if } s < q, \\ O(\mathsf{T}(n)\log n), & \text{if } s = q, \\ O\left(\mathsf{T}(n)\, n^{\log \frac{s}{q}}\right), & \text{if } s > q. \end{cases}$$

Proof:

$$\begin{aligned} \mathsf{C}(n) &\le \mathsf{T}(n) + s \cdot \mathsf{C}\left(\frac{n}{2}\right) \\ &\le \mathsf{T}(n) + s \cdot \mathsf{T}\left(\frac{n}{2}\right) + \cdots + s^{k-1} \cdot \mathsf{T}\left(\frac{n}{2^{k-1}}\right) + s^k \cdot \mathsf{C}\left(\frac{n}{2^k}\right) \\ &\le \mathsf{T}(n) \cdot \left(1 + \frac{s}{q} + \cdots + \left(\frac{s}{q}\right)^{\log(n)-1}\right) + s^{\log n} \cdot \mathsf{C}(1) \end{aligned}$$

The Master Theorem, main consequences

Corollary — DFT / Karatsuba

$$\mathsf{C}(n) \le s \cdot \mathsf{C}\left(\frac{n}{2}\right) + O(n) \quad \Longrightarrow \quad \mathsf{C}(n) = \begin{cases} O(n \log n), & \text{if } s = 2 \\ O(n^{\log s}), & \text{if } s \ge 3 \end{cases}$$

Corollary — Newton / evaluation-interpolation

$$\mathsf{C}(n) \le s \cdot \mathsf{C}\left(\frac{n}{2}\right) + O(\mathsf{M}(n)) \quad \Longrightarrow \quad \mathsf{C}(n) = \begin{cases} O(\mathsf{M}(n)), & \text{if } s = 1 \\ O(\mathsf{M}(n) \log n), & \text{if } s = 2 \end{cases}$$

Corollary — Strassen's matrix product

$$\mathsf{C}(n) \le s \cdot \mathsf{C}\left(\frac{n}{2}\right) + O(n^2), \quad (s \ge 5) \quad \Longrightarrow \quad \mathsf{C}(n) = O(n^{\log s})$$

Corollary — Strassen's matrix inversion

$$\mathsf{C}(n) \le s \cdot \mathsf{C}\left(\frac{n}{2}\right) + O(\mathsf{MM}(n)), \quad (s \le 3) \quad \Longrightarrow \quad \mathsf{C}(n) = O(\mathsf{MM}(n))$$

Divide and conquer

Karatsuba's algorithm

Gauss's trick ($\approx$1800) The product of two complex numbers can be computed using only 3 real multiplications

$$(ai+b)(ci+d) = (ad+bc)i + (bd-ac) = ((a+b)(c+d) - bd - ac)i + (bd-ac)$$

Kolmogorov (1956) n^2 conjecture: n^2 ops. are needed to multiply n-digit integers

Karatsuba (1960) disproof of the Kolmogorov conjecture

$\longrightarrow$ first DAC algorithm in Computer algebra; it combines Gauss's trick (on polynomials) with the power of recursion

$$(ax^{n/2}+b)(cx^{n/2}+d) = acx^n + ((a+b)(c+d) - bd - ac)x^{n/2} + bd$$

Master Theorem: $\mathsf{K}(n) = 3 \cdot \mathsf{K}(n/2) + O(n) \implies \mathsf{K}(n) = O(n^{\log(3)}) = O(n^{1.59})$

The idea behind the trick

Let $f = ax + b$, $g = cx + d$. Compute $h = fg$ by evaluation-interpolation:

Evaluation:

$$\begin{array}{rcl} b & = & f(0) \\ a+b & = & f(1) \\ a & = & f(\infty) \end{array} \qquad \begin{array}{rcl} d & = & g(0) \\ c+d & = & g(1) \\ c & = & g(\infty) \end{array}$$

Multiplication:

$$\begin{array}{rcl} h(0) & = & f(0) \cdot g(0) \\ h(1) & = & f(1) \cdot g(1) \\ h(\infty) & = & f(\infty) \cdot g(\infty) \end{array}$$

Interpolation:

$$h = h(0) + (h(1) - h(0) - h(\infty))\, x + h(\infty)\, x^2$$

Toom's algorithm

Let

$$f = f_0 + f_1 x + f_2 x^2, \quad g = g_0 + g_1 x + g_2 x^2$$

and

$$h = fg = h_0 + h_1 x + h_2 x^2 + h_3 x^3 + h_4 x^4.$$

To get h, do again:

- evaluation,
- multiplication,
- interpolation.

Now, 5 values are needed, because h has 5 unknown coefficients:

- $0, 1, -1, 2, \infty$ other choices are possible
- would not work with coefficients in $\mathbb{F}_2$.

The evaluation / interpolation phase

Evaluation:

$$
\begin{aligned}
f(0) &= f_0 & g(0) &= g_0 \\
f(1) &= f_0 + f_1 + f_2 & g(1) &= g_0 + g_1 + g_2 \\
f(-1) &= f_0 - f_1 + f_2 & g(-1) &= g_0 - g_1 + g_2 \\
f(2) &= f_0 + 2f_1 + 4f_2 & g(2) &= g_0 + 2g_1 + 4g_2 \\
f(\infty) &= f_2 & g(\infty) &= g_2
\end{aligned}
$$

Multiplication:

$$h(0) = f(0)g(0), \quad \ldots, \quad h(\infty) = f(\infty)g(\infty)$$

Interpolation: recover h from its values.

$\Longrightarrow$ one can multiply degree-2 polynomials using 5 products instead of 9

Master Theorem: $\mathsf{T}(n) = 5 \cdot \mathsf{T}(n/3) + O(n) \implies \mathsf{T}(n) = O(n^{\log_3(5)}) = O(n^{1.47})$

Generalization of Toom

Let

$$f = f_0 + f_1 x + \cdots + f_{\alpha-1} x^{\alpha-1}, \quad g = g_0 + g_1 x + \cdots + g_{\alpha-1} x^{\alpha-1}$$

and

$$h = fg = h_0 + h_1 x + \cdots + h_{2\alpha-2} x^{2\alpha-2}.$$

Analysis: at each step,

- divide n by α; — number of terms in f, g
- and perform $2\alpha - 1$ recursive calls; — number of terms in h
- the extra operations count is ℓn, for some ℓ.

Master theorem:

$$\mathsf{T}(n) = O(n^{\log_\alpha(2\alpha-1)}).$$

Examples:

$$\alpha = 100 \implies O(n^{1.15}), \quad \alpha = 1000 \implies O(n^{1.1}), \quad \alpha = 10000 \implies O(n^{1.07})$$

Discrete Fourier Transform (Gentleman-Sande 1966, decimation-in-frequency)

Problem: Given $n = 2^k$, $f \in \mathbb{K}[x]_{<n}$, and $\omega \in \mathbb{K}$ a primitive n-th root of unity, compute $(f(1), f(\omega), \ldots, f(\omega^{n-1}))$

Idea: $\omega = n$-th primitive root of 1 $\Longrightarrow \omega^2 = \frac{n}{2}$-th primitive root of 1, and

$$r_0(x) = f(x) \bmod x^{n/2} - 1 \quad \Longrightarrow \quad f(\omega^{2j}) = r_0\left((\omega^2)^j\right)$$

$$r_1(x) = f(x) \bmod x^{n/2} + 1 \quad \Longrightarrow \quad f(\omega^{2j+1}) = r_1(\omega^{2j+1}) = r_1(\omega x)\Big|_{x=(\omega^2)^j}$$

Moreover, $O(n)$ ops. are enough to get $r_0(x), r_1(x), r_1(\omega x)$ from $f(x)$

Complexity: $\quad \mathsf{F}(n) = 2 \cdot \mathsf{F}(n/2) + O(n) \quad \Longrightarrow \quad \mathsf{F}(n) = O(n \log n)$

Discrete Fourier Transform (Cooley-Tukey 1965, decimation-in-time)

Problem: Given $n = 2^k$, $f \in \mathbb{K}[x]_{<n}$, and $\omega \in \mathbb{K}$ a primitive n-th root of unity, compute $(f(1), f(\omega), \ldots, f(\omega^{n-1}))$

Idea: Write $f = f_{\text{even}}(x^2) + x f_{\text{odd}}(x^2), \quad \text{with } \deg(f_{\text{even}}), \deg(f_{\text{odd}}) < n/2.$

Then $f(\omega^j) = f_{\text{even}}(\omega^{2j}) + \omega^j f_{\text{odd}}(\omega^{2j})$, and $(\omega^{2j})_{0 \le j < n} = \frac{n}{2}$-roots of 1.

Complexity: $\mathsf{F}(n) = 2 \cdot \mathsf{F}(n/2) + O(n) \implies \mathsf{F}(n) = O(n \log n)$

Inverse DFT

Problem: Given $n = 2^k$, $v_0, \dots, v_{n-1} \in \mathbb{K}$ and $\omega \in \mathbb{K}$ a primitive n-th root of unity, compute $f \in \mathbb{K}[x]_{<n}$ such that $f(1) = v_0, \dots, f(\omega^{n-1}) = v_{n-1}$

- $V_\omega \cdot V_{\omega^{-1}} = n \cdot I_n \;\rightarrow\;$ performing the inverse DFT in size n amounts to:
  - performing a DFT at
  $$\frac{1}{1}, \;\; \frac{1}{\omega}, \;\; \dots, \;\; \frac{1}{\omega^{n-1}}$$
  - dividing the results by n.
- this new DFT is the same as before:
  $$\frac{1}{\omega^i} = \omega^{n-i},$$
  so the outputs are just shuffled.

Consequence: the cost of the inverse DFT is $O(n\log(n))$

FFT polynomial multiplication

Suppose the basefield $\mathbb{K}$ contains enough roots of unity

To multiply two polynomials f, g in $\mathbb{K}[x]$, of degrees $< n$:

- find $N = 2^k$ such that $h = fg$ has degree less than N — $N \leq 4n$
- compute DFT(f, N) and DFT(g, N) — $O(N \log(N))$
- multiply the values to get DFT(h, N) — $O(N)$
- recover h by inverse DFT — $O(N \log(N))$

Cost: $O(N \log(N)) = O(n \log(n))$

General case: Create artificial roots of unity — $O(n \log(n) \log \log n)$

Strassen's matrix multiplication algorithm

Same idea as for Karatsuba's algorithm: trick in low size + recursion

Additional difficulty: Formulas should be non-commutative

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x & y \\ z & t \end{bmatrix} \iff \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix} \times \begin{bmatrix} x \\ z \\ y \\ t \end{bmatrix}$$

Crucial remark: If $\varepsilon \in \{0, 1\}$ and $\alpha \in \mathbb{K}$, then 1 multiplication suffices for $E \cdot v$, where v is a vector, and E is a matrix of one of the following types:

$$\begin{bmatrix} \alpha & \alpha \\ \varepsilon\alpha & \varepsilon\alpha \end{bmatrix}, \begin{bmatrix} \alpha & -\alpha \\ \varepsilon\alpha & -\varepsilon\alpha \end{bmatrix}, \begin{bmatrix} \alpha & \varepsilon\alpha \\ -\alpha & -\varepsilon\alpha \end{bmatrix}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

$$M - \underbrace{\begin{bmatrix} a & a & & \\ a & a & & \\ & & & \\ & & & \end{bmatrix}}_{E_1} - \underbrace{\begin{bmatrix} & & & \\ & & & \\ & & d & d \\ & & d & d \end{bmatrix}}_{E_2} = \begin{bmatrix} & b-a & & \\ c-a & d-a & & \\ & & a-d & b-d \\ & & c-d & \end{bmatrix}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

$$M - E_1 - E_2 = \underbrace{\begin{bmatrix} & & & \\ & d-a & a-d & \\ & d-a & a-d & \\ & & & \end{bmatrix}}_{E_3} - \begin{bmatrix} & b-a & & \\ c-a & & d-a & \\ & a-d & & b-d \\ & & c-d & \end{bmatrix}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

$$M - E_1 - E_2 - E_3 = \begin{bmatrix} & b-a & & \\ & & & \\ & a-d & & b-d \\ & & & \end{bmatrix} + \begin{bmatrix} & & & \\ c-a & & d-a & \\ & & & \\ & & c-d & \end{bmatrix}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

$$M - E_1 - E_2 - E_3 = \underbrace{\begin{bmatrix} & b-a & & \\ & & & \\ & (b-d)-(b-a) & & b-d \\ & & & \end{bmatrix}}_{E_4 \;+\; E_5} + \underbrace{\begin{bmatrix} & & & \\ c-a & & (c-a)-(c-d) & \\ & & & \\ & & c-d & \end{bmatrix}}_{E_6 \;+\; E_7}$$

Strassen's matrix multiplication algorithm

Problem: Write

$$M = \begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

as a sum of less than 8 elementary matrices.

Conclusion

$$M = E_1 + E_2 + E_3 + E_4 + E_5 + E_6 + E_7$$

$\Longrightarrow$ one can multiply 2×2 matrices using 7 products instead of 8

Master Theorem:

$\mathsf{MM}(r) = 7 \cdot \mathsf{MM}(r/2) + O(r^2) \quad \Longrightarrow \quad \mathsf{MM}(r) = O(r^{\log_2(7)}) = O(r^{2.81})$

Inversion of dense matrices

[Strassen, 1969]

To invert a dense matrix $A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \in \mathcal{M}_r(\mathbb{K})$:

1. Invert $A_{1,1}$ (recursively)
2. Compute the Schur complement $\Delta := A_{2,2} - A_{2,1}A_{1,1}^{-1}A_{1,2}$
3. Invert Δ (recursively)
4. Recover the inverse of A as

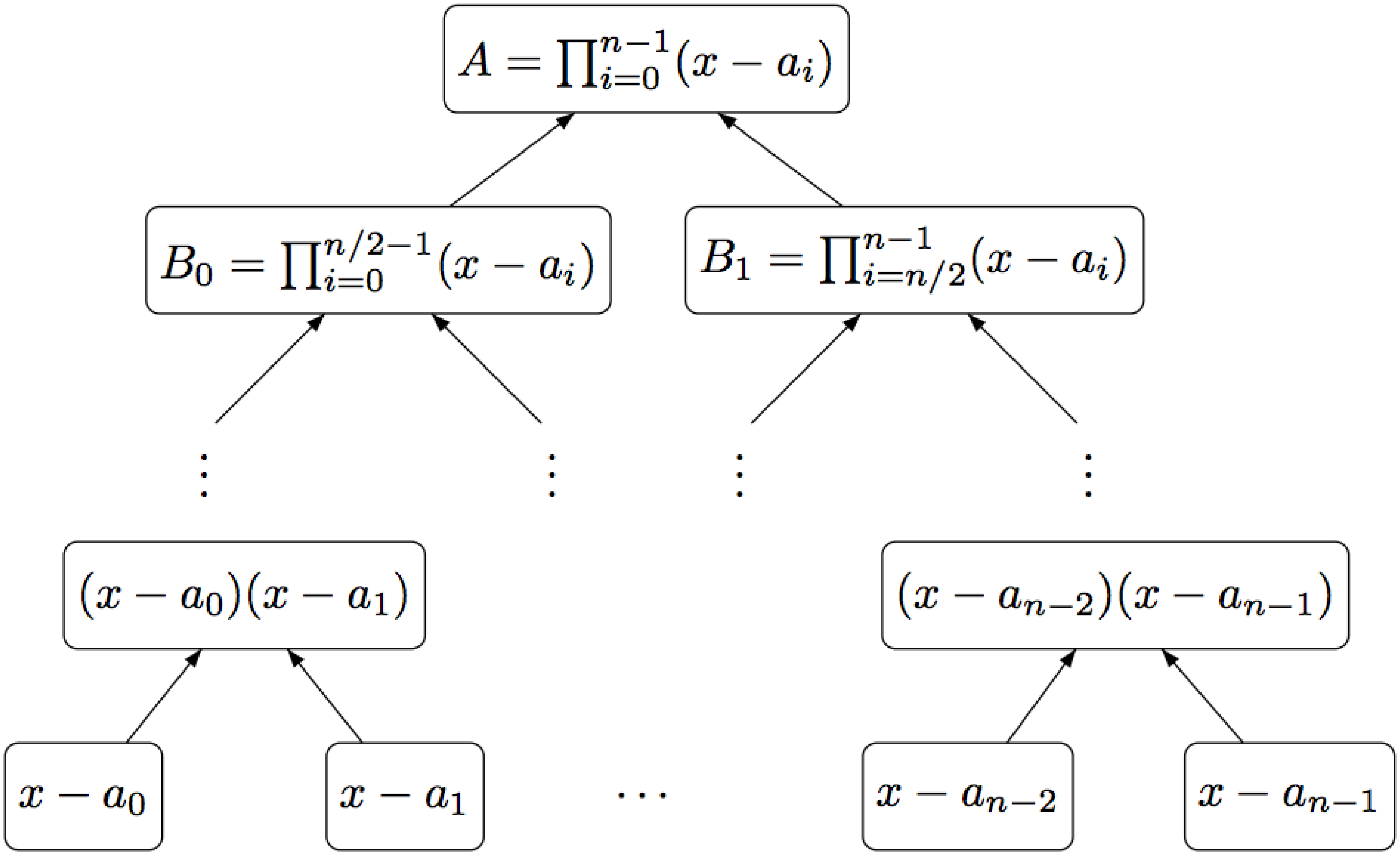
$$A^{-1} = \begin{bmatrix} I & -A_{1,1}^{-1}A_{1,2} \\ & I \end{bmatrix} \times \begin{bmatrix} A_{1,1}^{-1} & \\ & \Delta^{-1} \end{bmatrix} \times \begin{bmatrix} I & \\ -A_{2,1}A_{1,1}^{-1} & I \end{bmatrix}$$

Master Theorem: $\mathsf{C}(r) = 2 \cdot \mathsf{C}\left(\frac{r}{2}\right) + O(\mathsf{MM}(r)) \quad \Longrightarrow \quad \mathsf{C}(r) = O(\mathsf{MM}(r))$

Corollary: inversion A^{-1} and system solving $A^{-1}b$ in time $O(\mathsf{MM}(r))$

Subproduct tree

Problem: Given $a_0, \ldots, a_{n-1} \in \mathbb{K}$, compute $A = \prod_{i=0}^{n-1}(x - a_i)$



Master Theorem: $\mathsf{C}(n) = 2 \cdot \mathsf{C}(n/2) + O(\mathsf{M}(n)) \implies \mathsf{C}(n) = O(\mathsf{M}(n) \log n)$

Fast multipoint evaluation

[Borodin-Moenck, 1974]

Pb: Given $a_0, \ldots, a_{n-1} \in \mathbb{K}$ and $P \in \mathbb{K}[x]_{<n}$, compute $P(a_0), \ldots, P(a_{n-1})$

Naive algorithm: Compute $P(a_i)$ independently $O(n^2)$

Basic idea: Use recursively Bézout's identity $P(a) = P(x) \bmod (x - a)$

Divide and conquer: Same idea as for DFT = evaluation by repeated division

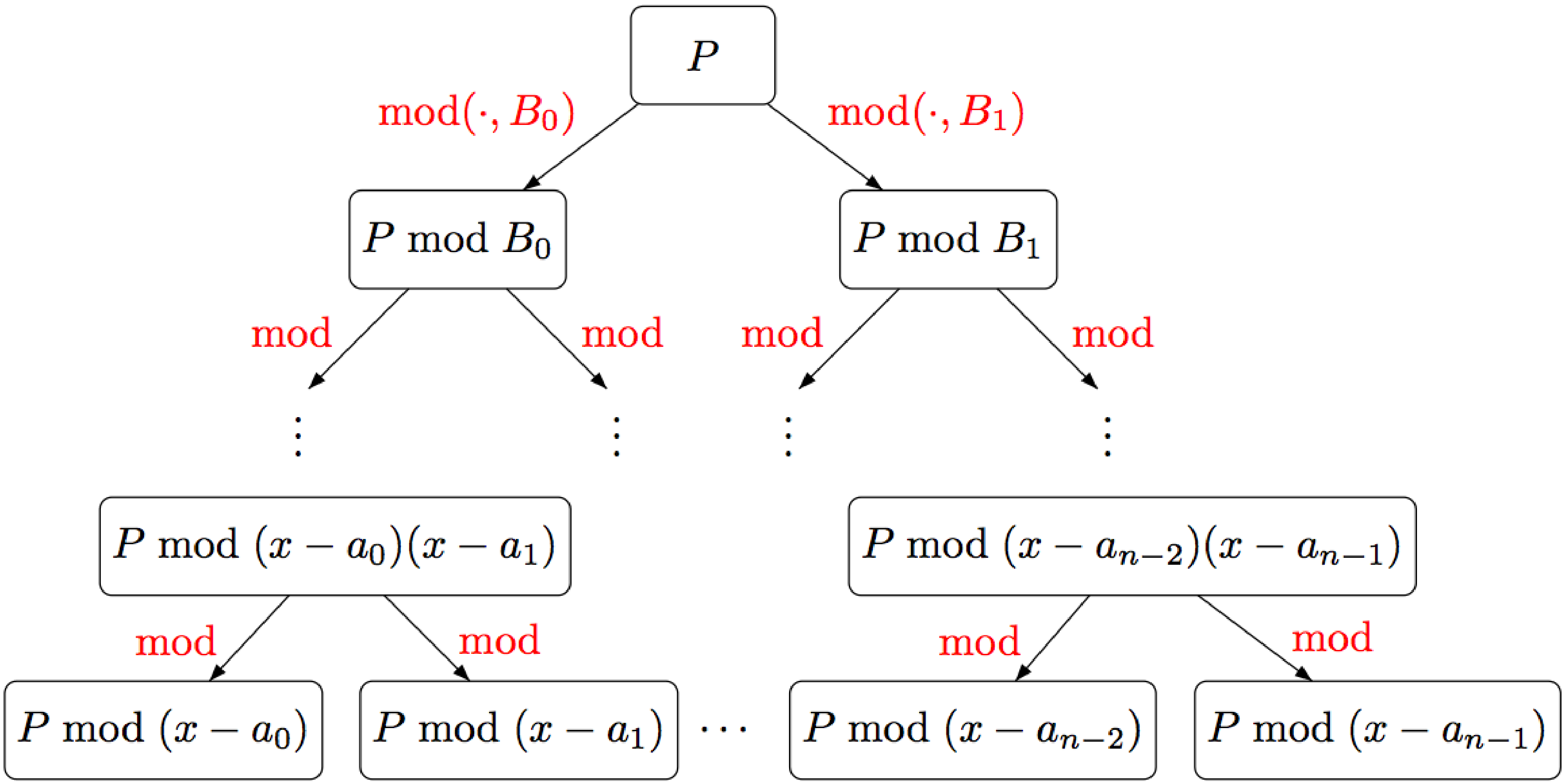
- $P_0 = P \mod (x - a_0) \cdots (x - a_{n/2-1})$
- $P_1 = P \mod (x - a_{n/2}) \cdots (x - a_{n-1})$

$$\Longrightarrow \begin{cases} P_0(a_0) = P(a_0), \quad \ldots, \quad P_0(a_{n/2-1}) = P(a_{n/2-1}) \\ P_1(a_{n/2}) = P(a_{n/2}), \quad \ldots, \quad P_1(a_{n-1}) = P(a_{n-1}) \end{cases}$$

Fast multipoint evaluation

[Borodin-Moenck, 1974]

Pb: Given $a_0, \dots, a_{n-1} \in \mathbb{K}$ and $P \in \mathbb{K}[x]_{<n}$, compute $P(a_0), \dots, P(a_{n-1})$



Master Theorem: $\mathsf{C}(n) = 2 \cdot \mathsf{C}(n/2) + O(\mathsf{M}(n)) \implies \mathsf{C}(n) = O(\mathsf{M}(n) \log n)$

Fast interpolation

[Borodin-Moenck, 1974]

Problem: Given $a_0, \dots, a_{n-1} \in \mathbb{K}$ mutually distinct, and $v_0, \dots, v_{n-1} \in \mathbb{K}$, compute $P \in \mathbb{K}[x]_{<n}$ such that $P(a_0) = v_0, \dots, P(a_{n-1}) = v_{n-1}$

Naive algorithm: Linear algebra, Vandermonde system $O(\mathsf{MM}(n))$

Lagrange's algorithm: Use $P(x) = \sum_{i=0}^{n-1} v_i \frac{\prod_{j \neq i}(x - a_j)}{\prod_{j \neq i}(a_i - a_j)}$ $O(n^2)$

Fast algorithm: Modified Lagrange formula

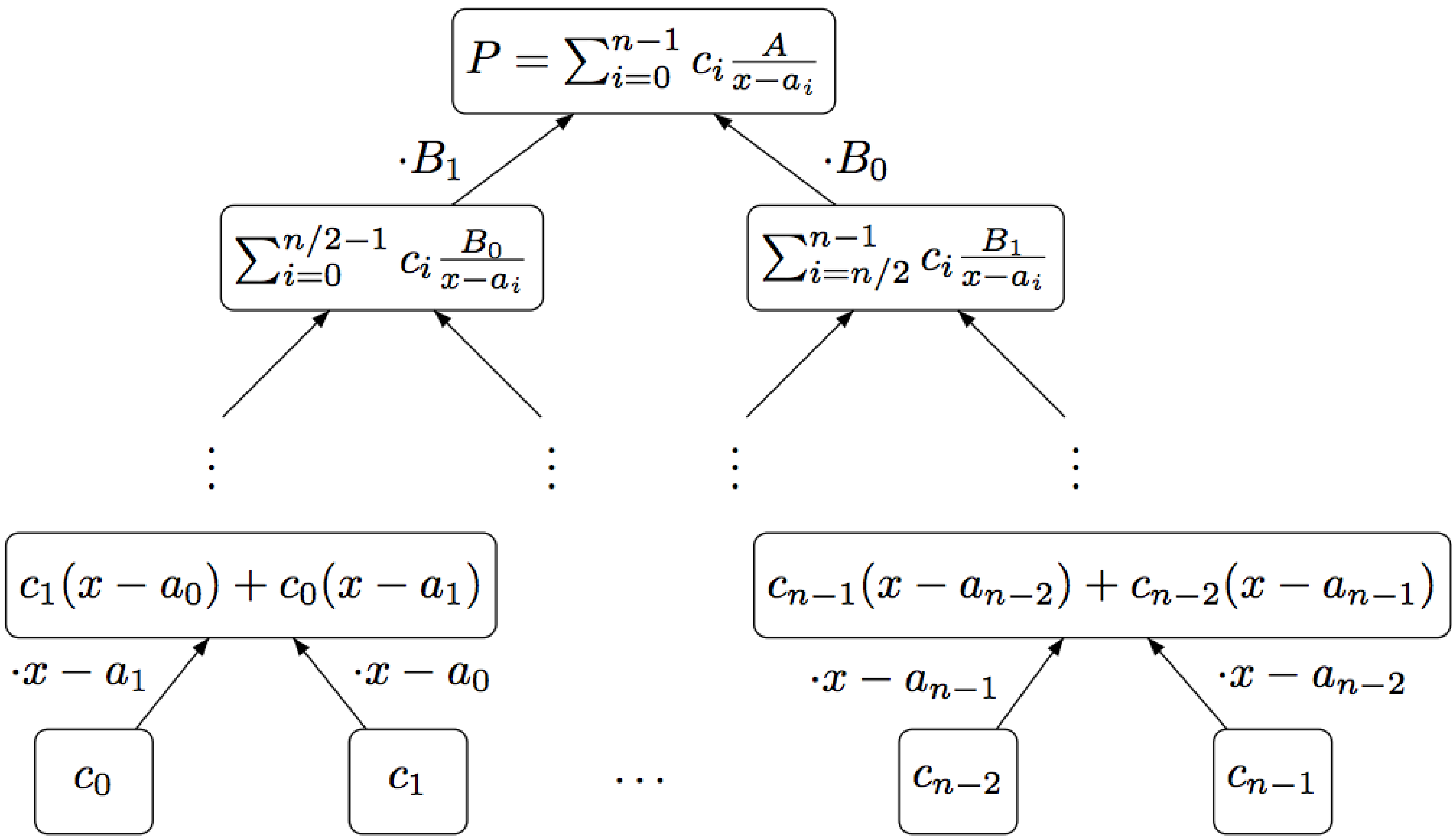
$$P = A(x) \cdot \sum_{i=0}^{n-1} \frac{v_i / A'(a_i)}{x - a_i}$$

- Compute $c_i = v_i / A'(a_i)$ by fast multipoint evaluation $O(\mathsf{M}(n) \log n)$
- Compute $\sum_{i=0}^{n-1} \frac{c_i}{x - a_i}$ by divide and conquer $O(\mathsf{M}(n) \log n)$

Fast interpolation

[Borodin-Moenck, 1974]

Problem: Given $a_0, \ldots, a_{n-1} \in \mathbb{K}$ mutually distinct, and $v_0, \ldots, v_{n-1} \in \mathbb{K}$, compute $P \in \mathbb{K}[x]_{<n}$ such that $P(a_0) = v_0, \ldots, P(a_{n-1}) = v_{n-1}$



Master Theorem: $\mathsf{C}(n) = 2 \cdot \mathsf{C}(n/2) + O(\mathsf{M}(n)) \implies \mathsf{C}(n) = O(\mathsf{M}(n) \log n)$