Computing terms of P-finite sequences

Part II

Marc Mezzarobba

MPRI CompAlg, 2025–2026, Lecture 10

Taylor shifts

Exercise. Design a divide-and-conquer algorithm that takes as input $P(x) \in \mathbb{K}[x]$ of degree $< d$ and a point $a \in \mathbb{K}$ and computes the Taylor shift $P(x + a)$ in $O(M(d) \log d)$ ops.

Remark. Can actually be done in $O(M(d))$ ops, at least if $\operatorname{char} \mathbb{K} \geqslant d$.

Taylor shifts

Exercise. Design a divide-and-conquer algorithm that takes as input $P(x) \in \mathbb{K}[x]$ of degree $< d$ and a point $a \in \mathbb{K}$ and computes the Taylor shift $P(x+a)$ in $O(M(d)\log d)$ ops.

Algorithm. (Assuming w.l.o.g. that $d = 2^k$.)

1. Compute $(x+a), (x+a)^2, \ldots, (x+a)^{2^{k-1}}$
2. Compute $P(x+a)$ using the following recursive subroutine.

   Given $Q(x)$ of degree $\delta - 1$:

   a. Write $Q(x) = Q_0(x) + x^{\delta/2} Q_1(x)$ with $\deg Q_1, \deg Q_2 < \delta/2$

   b. Recursively compute $Q_0(x+a)$, $Q_1(x+a)$

   c. Deduce $P(x+a)$

Remark. Can actually be done in $O(M(d))$ ops, at least if $\operatorname{char} \mathbb{K} \geqslant d$.

Taylor shifts

Exercise. Design a divide-and-conquer algorithm that takes as input $P(x) \in \mathbb{K}[x]$ of degree $< d$ and a point $a \in \mathbb{K}$ and computes the Taylor shift $P(x+a)$ in $O(M(d)\log d)$ ops.

Algorithm. (Assuming w.l.o.g. that $d = 2^k$.)

1. Compute $(x+a), (x+a)^2, \dots, (x+a)^{2^{k-1}}$ $\qquad O(M(d))$
2. Compute $P(x+a)$ using the following recursive subroutine.

   Given $Q(x)$ of degree $\delta - 1$:

   a. Write $Q(x) = Q_0(x) + x^{\delta/2} Q_1(x)$ with $\deg Q_1, \deg Q_2 < \delta/2$

   b. Recursively compute $Q_0(x+a)$, $Q_1(x+a)$

   c. Deduce $P(x+a)$

Remark. Can actually be done in $O(M(d))$ ops, at least if $\operatorname{char} \mathbb{K} \geqslant d$.

Taylor shifts

Exercise. Design a divide-and-conquer algorithm that takes as input $P(x) \in \mathbb{K}[x]$ of degree $< d$ and a point $a \in \mathbb{K}$ and computes the Taylor shift $P(x+a)$ in $O(M(d)\log d)$ ops.

Algorithm. (Assuming w.l.o.g. that $d = 2^k$.)

1. Compute $(x+a), (x+a)^2, \dots, (x+a)^{2^{k-1}}$ $O(M(d))$
2. Compute $P(x+a)$ using the following recursive subroutine.

   Given $Q(x)$ of degree $\delta - 1$:

   a. Write $Q(x) = Q_0(x) + x^{\delta/2} Q_1(x)$ with $\deg Q_1, \deg Q_2 < \delta/2$ $O(1)$

   b. Recursively compute $Q_0(x+a)$, $Q_1(x+a)$

   c. Deduce $P(x+a)$

Remark. Can actually be done in $O(M(d))$ ops, at least if $\operatorname{char} \mathbb{K} \geqslant d$.

Taylor shifts

Exercise. Design a divide-and-conquer algorithm that takes as input $P(x) \in \mathbb{K}[x]$ of degree $< d$ and a point $a \in \mathbb{K}$ and computes the Taylor shift $P(x+a)$ in $O(M(d)\log d)$ ops.

Algorithm. (Assuming w.l.o.g. that $d = 2^k$.)

1. Compute $(x+a), (x+a)^2, \dots, (x+a)^{2^{k-1}}$ — $O(M(d))$
2. Compute $P(x+a)$ using the following recursive subroutine.

   Given $Q(x)$ of degree $\delta - 1$:

   a. Write $Q(x) = Q_0(x) + x^{\delta/2} Q_1(x)$ with $\deg Q_1, \deg Q_2 < \delta/2$ — $O(1)$

   b. Recursively compute $Q_0(x+a)$, $Q_1(x+a)$ — $2\,C(\delta/2)$

   c. Deduce $P(x+a)$

Remark. Can actually be done in $O(M(d))$ ops, at least if $\operatorname{char} \mathbb{K} \geqslant d$.

Taylor shifts

Exercise. Design a divide-and-conquer algorithm that takes as input $P(x) \in \mathbb{K}[x]$ of degree $< d$ and a point $a \in \mathbb{K}$ and computes the Taylor shift $P(x+a)$ in $O(M(d)\log d)$ ops.

Algorithm. (Assuming w.l.o.g. that $d = 2^k$.)

1. Compute $(x+a), (x+a)^2, \dots, (x+a)^{2^{k-1}}$ — $O(M(d))$
2. Compute $P(x+a)$ using the following recursive subroutine.

   Given $Q(x)$ of degree $\delta - 1$:

   a. Write $Q(x) = Q_0(x) + x^{\delta/2} Q_1(x)$ with $\deg Q_1, \deg Q_2 < \delta/2$ — $O(1)$

   b. Recursively compute $Q_0(x+a)$, $Q_1(x+a)$ — $2\,C(\delta/2)$

   c. Deduce $P(x+a)$ — $O(M(\delta))$

Remark. Can actually be done in $O(M(d))$ ops, at least if $\operatorname{char} \mathbb{K} \geqslant d$.

Taylor shifts

Exercise. Design a divide-and-conquer algorithm that takes as input $P(x) \in \mathbb{K}[x]$ of degree $< d$ and a point $a \in \mathbb{K}$ and computes the Taylor shift $P(x+a)$ in $O(M(d)\log d)$ ops.

Algorithm. (Assuming w.l.o.g. that $d = 2^k$.)

1. Compute $(x+a), (x+a)^2, \dots, (x+a)^{2^{k-1}}$ — $O(M(d))$
2. Compute $P(x+a)$ using the following recursive subroutine.

   Given $Q(x)$ of degree $\delta - 1$: — $C(\delta) = 2\,C(\delta/2) + O(M(\delta))$

   a. Write $Q(x) = Q_0(x) + x^{\delta/2}\,Q_1(x)$ with $\deg Q_1, \deg Q_2 < \delta/2$ — $O(1)$

   b. Recursively compute $Q_0(x+a)$, $Q_1(x+a)$ — $2\,C(\delta/2)$

   c. Deduce $P(x+a)$ — $O(M(\delta))$

Remark. Can actually be done in $O(M(d))$ ops, at least if $\operatorname{char} \mathbb{K} \geqslant d$.

Taylor shifts

Exercise. Design a divide-and-conquer algorithm that takes as input $P(x) \in \mathbb{K}[x]$ of degree $< d$ and a point $a \in \mathbb{K}$ and computes the Taylor shift $P(x+a)$ in $O(M(d)\log d)$ ops.

Algorithm. (Assuming w.l.o.g. that $d = 2^k$.)

1. Compute $(x+a), (x+a)^2, \dots, (x+a)^{2^{k-1}}$ — $O(M(d))$
2. Compute $P(x+a)$ using the following recursive subroutine. — $O(M(d)\log(d))$

   Given $Q(x)$ of degree $\delta - 1$: — $C(\delta) = 2\,C(\delta/2) + O(M(\delta))$

   a. Write $Q(x) = Q_0(x) + x^{\delta/2} Q_1(x)$ with $\deg Q_1, \deg Q_2 < \delta/2$ — $O(1)$

   b. Recursively compute $Q_0(x+a)$, $Q_1(x+a)$ — $2\,C(\delta/2)$

   c. Deduce $P(x+a)$ — $O(M(\delta))$

Remark. Can actually be done in $O(M(d))$ ops, at least if $\operatorname{char} \mathbb{K} \geqslant d$.

Situation

$$b_s(n)\,u_{n+s} + \cdots + b_1(n)\,u_{n+1} + b_0(n)\,u_n = 0 \qquad \text{(Rec)}$$

		C-finite (lecture 4)	P-finite
first N terms	arithmetic	$O(N)$	$O(N)$
	bit	$O(N^2)$	$\tilde{O}(\mathbf{N}^2)$
Nth term	arithmetic	$O(\log N)$	$\tilde{O}(\sqrt{\mathbf{N}})$
	bit	$O(M(N))$	$\tilde{O}(\mathbf{N})$

bit sizes	
	Nth term: $O(N \log N)$
	N terms: $O(N^2 \log N)$

Assumption: Nonsingular recurrences ($b_s(n) \neq 0$ for all $n \in \mathbb{N}$).

1 Baby steps, giant steps

A baby steps-giant steps algorithm for N!

[Strassen 1976]

$$N! = \underbrace{\underbrace{1 \cdot 2 \cdots \ell}_{} \cdot \underbrace{(\ell+1)\,(\ell+2) \cdots (2\,\ell)}_{} \cdots \underbrace{(\ell^2-\ell+1)\,(\ell^2-\ell+2) \cdots \ell^2}_{}}_{N^{1/2} \text{ blocks of size } N^{1/2}} \qquad \ell = N^{1/2}$$

Algorithm. *Input*: N *Output*: $N!$

1. Let $\ell = \lfloor N^{1/2} \rfloor$
2. Baby steps:
   a. Compute $F = (x+1)\,(x+2) \cdots (x+\ell)$ $O(M(\ell) \log \ell)$
3. Giant steps:
   a. Compute $P_0 = F(0), P_1 = F(\ell), P_2 = F(2\,\ell), \ldots, P_{\ell-1} = F((\ell-1)\,\ell)$ by multipoint evaluation $O(M(\ell) \log \ell)$
   b. Return $P_0\,P_1 \cdots P_{\ell-1} \cdot (\ell^2+1) \cdots (N-1)\,N$ $O(\ell)$

Total $O(M(N^{1/2}) \log N)$ ops

Generalization to P-recursive sequences

[Chudnovsky & Chudnovsky 1987]

Write the recurrence in matrix form, pull out the denominator:

$$\begin{pmatrix} u_{n+1} \\ \vdots \\ u_{n+s-1} \\ u_{n+s} \end{pmatrix} = \frac{1}{b_s(n)} \underbrace{\begin{pmatrix} & b_s(n) & & \\ & & \ddots & \\ & & & b_s(n) \\ -b_0(n) & -b_1(n) & \cdots & -b_{s-1}(n) \end{pmatrix}}_{B(n)} \underbrace{\begin{pmatrix} u_n \\ \vdots \\ u_{n+s-2} \\ u_{n+s-1} \end{pmatrix}}_{U_n}$$

Then

$$U_N = \frac{1}{b_s(N-1)\cdots b_s(1)\, b_s(0)}\, B(N-1)\cdots B(1)\, B(0)\, U_0$$

$B(n) =$ matrix of polynomials of degree $< d$

Exercise. Adapt the previous algorithm to the product $B(N-1)\cdots B(0)$

Fast polynomial matrix "factorial"

Algorithm. *Input*: $B \in \mathbb{K}[n]^{s \times s}$ of deg $< d$, $N \in \mathbb{N}$ *Output*: $B(N-1) \cdots B(1)\, B(0)$

1. Write $N = \ell\, m$ with $\ell =$ and $m =$ (assumed exact for simplicity)
2. Baby steps:
   a. Compute $B(X+1), \ldots, B(X+\ell-1)$
   b. Compute $F(X) = B(X+\ell-1) \cdots B(X+1)\, B(X)$
3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m-1)\,\ell)$ simultaneously
   b. Deduce and return the product $F((m-1)\,\ell) \cdots F(\ell)\, F(0)$

Fast polynomial matrix "factorial"

Algorithm. *Input*: $B \in \mathbb{K}[n]^{s \times s}$ of deg $< d$, $N \in \mathbb{N}$ *Output*: $B(N-1) \cdots B(1)\, B(0)$

1. Write $N = \ell\, m$ with $\ell =$ and $m =$ (assumed exact for simplicity)
2. Baby steps:
   a. Compute $B(X+1), \ldots, B(X+\ell-1)$
   b. Compute $F(X) = B(X+\ell-1) \cdots B(X+1)\, B(X)$
3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m-1)\,\ell)$ simultaneously
   b. Deduce and return the product $F((m-1)\,\ell) \cdots F(\ell)\, F(0)$

Fast polynomial matrix "factorial"

Algorithm. *Input*: $B \in \mathbb{K}[n]^{s \times s}$ of deg $< d$, $N \in \mathbb{N}$ *Output*: $B(N-1) \cdots B(1)\, B(0)$

1. Write $N = \ell\, m$ with $\ell =$ and $m =$ (assumed exact for simplicity)
2. Baby steps:
   a. Compute $B(X+1), \ldots, B(X+\ell-1)$
   b. Compute $F(X) = B(X+\ell-1) \cdots B(X+1)\, B(X)$ $O(M(\ell\, d) \log(\ell)\, s^\theta)$
3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m-1)\,\ell)$ simultaneously
   b. Deduce and return the product $F((m-1)\,\ell) \cdots F(\ell)\, F(0)$

$\deg F(X) < \ell\, d$

Fast polynomial matrix "factorial"

Algorithm. *Input*: $B \in \mathbb{K}[n]^{s \times s}$ of deg $< d$, $N \in \mathbb{N}$ *Output*: $B(N-1) \cdots B(1)\, B(0)$

1. Write $N = \ell\, m$ with $\ell =$ and $m =$ (assumed exact for simplicity)
2. Baby steps:
   a. Compute $B(X+1), \ldots, B(X+\ell-1)$
   b. Compute $F(X) = B(X+\ell-1) \cdots B(X+1)\, B(X)$ $O(M(\ell\, d) \log(\ell)\, s^\theta)$
3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m-1)\,\ell)$ simultaneously $O(M(m) \log(m)\, s^\theta)$
   b. Deduce and return the product $F((m-1)\,\ell) \cdots F(\ell)\, F(0)$

$\deg F(X) < \ell\, d \qquad \ell\, d \leqslant m$

Fast polynomial matrix "factorial"

Algorithm. *Input*: $B \in \mathbb{K}[n]^{s \times s}$ of deg $< d$, $N \in \mathbb{N}$ *Output*: $B(N-1) \cdots B(1)\, B(0)$

1. Write $N = \ell\, m$ with $\ell = (N/d)^{1/2}$ and $m = (N\, d)^{1/2}$ (assumed exact for simplicity)
2. Baby steps:
   a. Compute $B(X+1), \ldots, B(X+\ell-1)$
   b. Compute $F(X) = B(X+\ell-1) \cdots B(X+1)\, B(X)$ $O(M(\ell\, d) \log(\ell)\, s^\theta)$
3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m-1)\,\ell)$ simultaneously $O(M(m) \log(m)\, s^\theta)$
   b. Deduce and return the product $F((m-1)\,\ell) \cdots F(\ell)\, F(0)$

$\deg F(X) < \ell\, d \qquad \ell\, d \leqslant m$

Fast polynomial matrix "factorial"

Algorithm. *Input*: $B \in \mathbb{K}[n]^{s \times s}$ of deg $< d$, $N \in \mathbb{N}$ *Output*: $B(N-1) \cdots B(1)\, B(0)$

1. Write $N = \ell\, m$ with $\ell = (N/d)^{1/2}$ and $m = (N\, d)^{1/2}$ (assumed exact for simplicity)
2. Baby steps:
   a. Compute $B(X+1), \ldots, B(X+\ell-1)$
   b. Compute $F(X) = B(X+\ell-1) \cdots B(X+1)\, B(X)$ $O(M(\ell\, d) \log(\ell)\, s^\theta)$
3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m-1)\,\ell)$ simultaneously $O(M(m) \log(m)\, s^\theta)$
   b. Deduce and return the product $F((m-1)\,\ell) \cdots F(\ell)\, F(0)$ $O(m\, s^\theta)$

$\deg F(X) < \ell\, d \qquad \ell\, d \leqslant m$

Fast polynomial matrix "factorial"

Algorithm. *Input*: $B \in \mathbb{K}[n]^{s \times s}$ of deg $< d$, $N \in \mathbb{N}$ *Output*: $B(N-1) \cdots B(1)\, B(0)$

1. Write $N = \ell\, m$ with $\ell = (N/d)^{1/2}$ and $m = (N\,d)^{1/2}$ (assumed exact for simplicity)
2. Baby steps:
   a. Compute $B(X+1), \ldots, B(X+\ell-1)$ $O(\ell\, M(d) \log(d)\, s^2)$
   b. Compute $F(X) = B(X+\ell-1) \cdots B(X+1)\, B(X)$ $O(M(\ell\, d) \log(\ell)\, s^\theta)$
3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m-1)\,\ell)$ simultaneously $O(M(m) \log(m)\, s^\theta)$
   b. Deduce and return the product $F((m-1)\,\ell) \cdots F(\ell)\, F(0)$ $O(m\, s^\theta)$

$\deg F(X) < \ell\, d \qquad \ell\, d \leqslant m$

naïvely step 2a takes $O(\ell\, d^2\, s^2)$ ops

Fast polynomial matrix "factorial"

Algorithm. *Input*: $B \in \mathbb{K}[n]^{s \times s}$ of deg $< d$, $N \in \mathbb{N}$ *Output*: $B(N-1) \cdots B(1)\, B(0)$

1. Write $N = \ell\, m$ with $\ell = (N/d)^{1/2}$ and $m = (N\, d)^{1/2}$ (assumed exact for simplicity)
2. Baby steps:
   a. Compute $B(X+1), \ldots, B(X+\ell-1)$ $O(\ell\, M(d) \log(d)\, s^2)$
   b. Compute $F(X) = B(X+\ell-1) \cdots B(X+1)\, B(X)$ $O(M(\ell\, d) \log(\ell)\, s^\theta)$
3. Giant steps:
   a. Compute $F(0), F(\ell), \ldots, F((m-1)\,\ell)$ simultaneously $O(M(m) \log(m)\, s^\theta)$
   b. Deduce and return the product $F((m-1)\,\ell) \cdots F(\ell)\, F(0)$ $O(m\, s^\theta)$

$\deg F(X) < \ell\, d \qquad \ell\, d \leqslant m$

Total $O\Big(M(m) \log(m)\, s^\theta\Big)$

naïvely step 2a takes $O(\ell\, d^2\, s^2)$ ops

Nth term of a P-recursive sequence

Algorithm. *Notation as before.*

1. Compute $B(N-1)\cdots B(1)\,B(0)$ by the previous algorithm — $O(M(m)\log(m)\,s^\theta)$
2. Compute $b_s(N-1)\cdots b_s(1)\,b_s(0)$ by the previous algorithm — $O(M(m)\log(m))$
3. Divide, return — $O(s^2)$

Theorem. Let $(u^{(0)},\dots,u^{(s-1)})$ be the basis of solutions s.t. $u_i^{(j)}=\delta_{i,j}$ of a nonsingular recurrence of order s and degree $<d$. One can compute the matrix $\left(u_{N+i}^{(j)}\right)_{i,j}\in\mathbb{K}^{s\times s}$ in

$$O\Big(M(\sqrt{N\,d})\log(N\,d)\,s^\theta\Big)\quad\text{ops.}$$

Corollary. One can compute the Nth term of any fixed P-recursive sequence in $O(M(\sqrt{N})\log N)$ ops.

2 Binary splitting

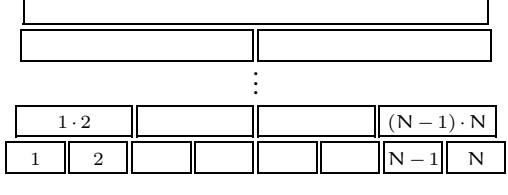
Computing N! in quasi-linear time

Algorithm. Use a product tree. That is, split the product as

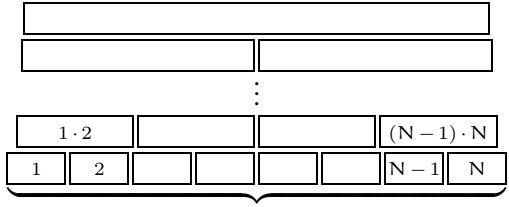
$$N! = \underbrace{1 \cdot 2 \cdots m}_{P(0,m)} \cdot \underbrace{(m+1) \cdots N}_{P(m,N)}, \qquad m = \lfloor N/2 \rfloor,$$

and compute subproducts recursively as

$$P(\ell, h) = P\left(\left\lfloor \frac{\ell + h}{2} \right\rfloor, h\right) P\left(\ell, \left\lfloor \frac{\ell + h}{2} \right\rfloor\right)$$

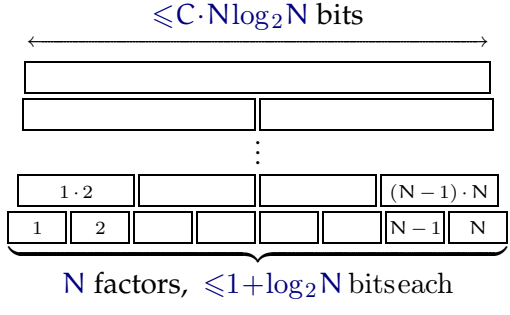


Complexity analysis

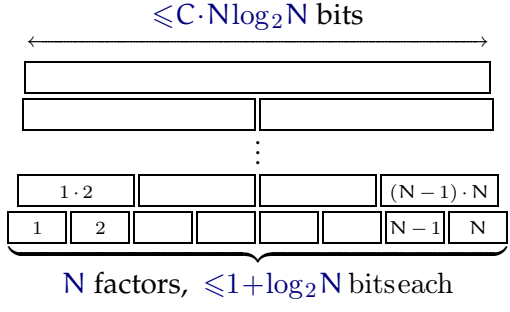


N factors, $\leqslant 1+\log_2 N$ bits each

Complexity analysis

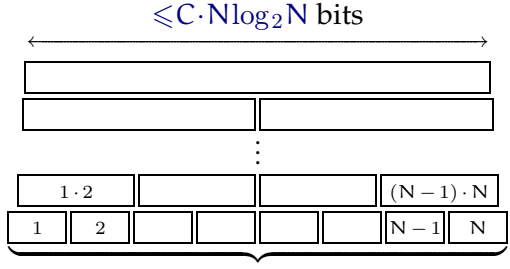


Complexity analysis



$\leqslant \frac{N}{2} M(1 + \log_2 N)$

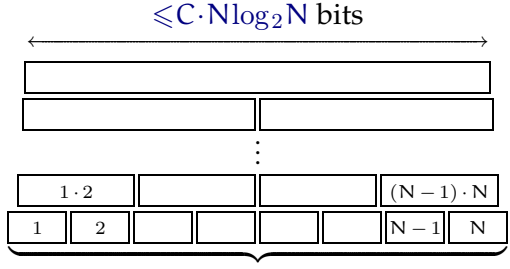
Complexity analysis



$$\text{ל} \leqslant M\left(\frac{N}{2}(1+\log_2 N)\right)$$

$$\text{ל} \leqslant \frac{N}{2} M(1+\log_2 N)$$

Complexity analysis

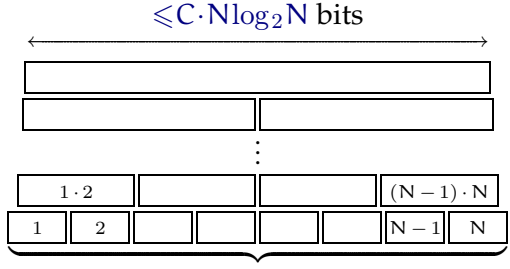


$$\text{ʖ} \leqslant M\left(\frac{N}{2}(1+\log_2 N)\right)$$

$$\text{ʖ} \leqslant \frac{N}{2} M(1+\log_2 N)$$

$$\sum_{i=0}^{\log_2(N)-1} 2^i M\left(\frac{N}{2^{i+1}}(1+\log_2 N)\right)$$

Complexity analysis

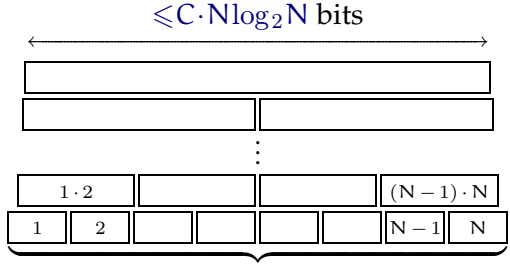


$$\beth \leqslant M\left(\frac{N}{2}(1+\log_2 N)\right)$$

$$\beth \leqslant \frac{N}{2} M(1+\log_2 N)$$

$$\sum_{i=0}^{\log_2(N)-1} 2^i M\left(\frac{N}{2^{i+1}}(1+\log_2 N)\right) \leqslant \frac{1}{2} \sum_{i=0}^{\log_2(N)-1} M(N(1+\log_2 N))$$

Complexity analysis



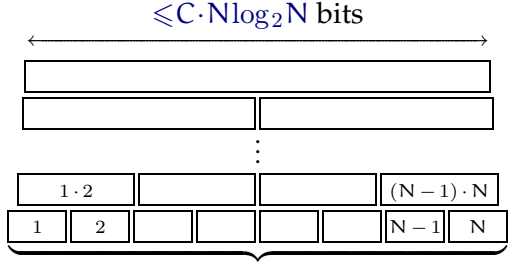
$$\jmath \leqslant M\left(\frac{N}{2}(1+\log_2 N)\right)$$

$$\jmath \leqslant \frac{N}{2} M(1+\log_2 N)$$

$$\sum_{i=0}^{\log_2(N)-1} 2^i M\left(\frac{N}{2^{i+1}}(1+\log_2 N)\right) \leqslant \frac{1}{2} \sum_{i=0}^{\log_2(N)-1} M(N(1+\log_2 N))$$

$$\leqslant \frac{1}{2} M(N(1+\log_2 N)) \log_2 N$$

Complexity analysis

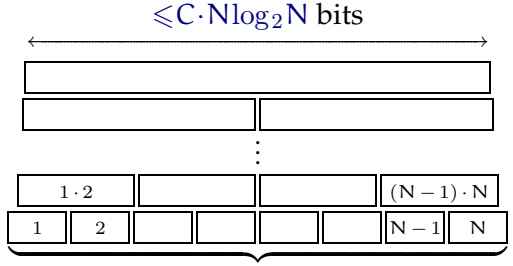


$$\text{ל} \leqslant M\left(\frac{N}{2}(1+\log_2 N)\right)$$

$$\text{ל} \leqslant \frac{N}{2} M(1+\log_2 N)$$

$$\begin{aligned}
\sum_{i=0}^{\log_2(N)-1} 2^i M\left(\frac{N}{2^{i+1}}(1+\log_2 N)\right) &\leqslant \frac{1}{2} \sum_{i=0}^{\log_2(N)-1} M(N(1+\log_2 N)) \\
&\leqslant \frac{1}{2} M(N(1+\log_2 N)) \log_2 N \\
&= O(M(N \log(N)^2))
\end{aligned}$$

Complexity analysis



$$\begin{aligned}\sum_{i=0}^{\log_2(N)-1} 2^i M\left(\frac{N}{2^{i+1}}(1+\log_2 N)\right) &\leqslant \frac{1}{2}\sum_{i=0}^{\log_2(N)-1} M(N(1+\log_2 N)) \\ &\leqslant \frac{1}{2} M(N(1+\log_2 N))\log_2 N \\ &= O(M(N\log(N)^2))\end{aligned}$$

Remark. In the special case of $N!$, there are algorithms of complexity $O(M(N \log N))$.

Nth term of a P-recursive sequence

[Chudnovsky & Chudnovsky 1987]

$$b_s(n)\,u_{n+s} + \cdots + b_1(n)\,u_{n+1} + b_0(n)\,u_n = 0, \quad b_i \in \mathbb{Z}[n]$$

Same idea as before:

write $U_n = (u_n, \ldots, u_{n+s-1})$ and $U_N = \dfrac{1}{b_s(N-1)\cdots b_s(1)\,b_s(0)}\,B(N-1)\cdots B(1)\,B(0)\,U_0$

Algorithm.

1. Compute $B(N-1)\cdots B(1)\,B(0)$ by **binary splitting**
2. Compute $b_s(N-1)\cdots b_s(1)\,b_s(0)$ by **binary splitting**
3. Divide

Theorem. One can compute the Nth term of a sequence $(u_n) \in \mathbb{Q}^{\mathbb{N}}$ given by a recurrence with coefficients in $\mathbb{Z}[n]$ in bit operations.

Nth term of a P-recursive sequence

[Chudnovsky & Chudnovsky 1987]

$$b_s(n)\,u_{n+s} + \cdots + b_1(n)\,u_{n+1} + b_0(n)\,u_n = 0, \quad b_i \in \mathbb{Z}[n]$$

Same idea as before:

write $U_n = (u_n, \ldots, u_{n+s-1})$ and $U_N = \dfrac{1}{b_s(N-1)\cdots b_s(1)\,b_s(0)}\,B(N-1)\cdots B(1)\,B(0)\,U_0$

Algorithm. (costs for fixed recurrence, hides dependency on s and d)

1. Compute $B(N-1)\cdots B(1)\,B(0)$ by **binary splitting** $O(M(N\log N)\log(N))$
2. Compute $b_s(N-1)\cdots b_s(1)\,b_s(0)$ by **binary splitting** $O(M(N\log N)\log(N))$
3. Divide

Theorem. One can compute the Nth term of a sequence $(u_n) \in \mathbb{Q}^{\mathbb{N}}$ given by a recurrence with coefficients in $\mathbb{Z}[n]$ in bit operations.

Nth term of a P-recursive sequence

[Chudnovsky & Chudnovsky 1987]

$$b_s(n)\,u_{n+s} + \cdots + b_1(n)\,u_{n+1} + b_0(n)\,u_n = 0, \quad b_i \in \mathbb{Z}[n]$$

Same idea as before:

write $U_n = (u_n, \ldots, u_{n+s-1})$ and $U_N = \dfrac{1}{b_s(N-1)\cdots b_s(1)\,b_s(0)}\,B(N-1)\cdots B(1)\,B(0)\,U_0$

Algorithm. (costs for fixed recurrence, hides dependency on s and d)

1. Compute $B(N-1)\cdots B(1)\,B(0)$ by **binary splitting** $O(M(N\log N)\log(N))$
2. Compute $b_s(N-1)\cdots b_s(1)\,b_s(0)$ by **binary splitting** $O(M(N\log N)\log(N))$
3. Divide (gcd!) $O(M(N\log N)\log(N))$

Theorem. One can compute the Nth term of a sequence $(u_n) \in \mathbb{Q}^{\mathbb{N}}$ given by a recurrence with coefficients in $\mathbb{Z}[n]$ in $O(M(N\log^2 N))$ bit operations.

Sums of series

If (u_k) is P-finite over $\mathbb{Q}$ and $\xi \in \mathbb{Q}$, then $s_n = \sum_{k=0}^{n-1} u_k \xi^k$ is P-finite. (why?)

Sums of series

If (u_k) is P-finite over $\mathbb{Q}$ and $\xi \in \mathbb{Q}$, then $s_n = \sum_{k=0}^{n-1} u_k \xi^k$ is P-finite. (why?)

Example. $e = \exp(1)$ with error $\leqslant 2^{-t}$ in $O\big(M(t \log t)\big)$ bit operations

$$e = \sum_{k=0}^{n-1} \frac{1}{k!} + \underbrace{\frac{1}{n!} \sum_{k=0}^{\infty} \frac{1}{(n+1)\cdots(n+k)}}_{\leqslant e/n!}, \qquad \frac{e}{n!} \leqslant 2^{-t} \text{ for } n = \frac{t + o(t)}{\log_2 t}$$

Cost of the binary splitting method: $O\left(M\left(\frac{t}{\log_2 t} \log\left(\frac{t}{\log_2 t}\right)^2\right)\right) = O(M(t \log t))$

Sums of series

If (u_k) is P-finite over $\mathbb{Q}$ and $\xi \in \mathbb{Q}$, then $s_n = \sum_{k=0}^{n-1} u_k \xi^k$ is P-finite. (why?)

Example. $e = \exp(1)$ with error $\leqslant 2^{-t}$ in $O\big(M(t \log t)\big)$ bit operations

$$e = \sum_{k=0}^{n-1} \frac{1}{k!} + \underbrace{\frac{1}{n!} \sum_{k=0}^{\infty} \frac{1}{(n+1)\cdots(n+k)}}_{\leqslant e/n!}, \qquad \frac{e}{n!} \leqslant 2^{-t} \text{ for } n = \frac{t + o(t)}{\log_2 t}$$

Cost of the binary splitting method: $O\left(M\left(\frac{t}{\log_2 t} \log\left(\frac{t}{\log_2 t}\right)^2\right)\right) = O(M(t \log t))$

Example. π in $O(M(t \log^2 t))$ bit operations, small $O(\cdot)$ constant

$$\frac{1}{\pi} = \frac{12}{c^{3/2}} \sum_{n=0}^{\infty} (-1)^n \frac{(6n)!}{(3n)!\, n!^3} \frac{(an+b)}{c^{3n}}, \qquad \begin{cases} a = 545140134 \\ b = 13591409 \\ c = 640320 \end{cases}$$

[Chudnovsky² 1987]

1 hypergeometric series, 1 square root, 1 division

Used in record computations — although another algo. yields t digits of π in only $O(M(t) \log t)$ bit ops

[Salamin 1976, Brent 1978]

Another application

[Flajolet & Salvy 1997]

Exercise. Compute the coefficient of x^{2N} in

$$f(x) = (1+x)^{2N}(1+x+x^2)^N$$

for $N = 10^6$.

Another application

[Flajolet & Salvy 1997]

Exercise. Compute the coefficient of x^{2N} in

$$f(x) = (1+x)^{2N}(1+x+x^2)^N$$

for $N = 10^6$.

Let $f(x) = (1+x)^{2N}(1+x+x^2)^N$. One has

$$\frac{f'(x)}{f(x)} = 2N\frac{1}{1+x} + N\frac{2x+1}{1+x+x^2}.$$

Convert ODE to recurrence, use binary splitting.