Generic reductions for in-place polynomial multiplication

Pascal Giorgi[1] **Bruno Grenet**[1] Daniel S. Roche[2]

ISSAC'19 – Beijing – July 15-18, 2019

[1] LIRMM, Université de Montpellier
[2] CS Department, US Naval Academy

Computer Algebra 101

Multiplication of polynomials: $\mathsf{M}(n)$

Naive:	$O(n^2)$	
Karatsuba:	$O(n^{\log_2 3}) = O(n^{1.585})$	Karatsuba (1962)
Toom-3:	$O(n^{\log_3 5}) = O(n^{1.465})$	Toom (1963), Cook (1966)
FFT-based algorithms:		
	$O(n \log n)$ with $\omega^{2n} = 1$	Cooley, Tukey (1965)
	$O(n \log n \log\log n)$	Cantor, Kaltofen (1991)
	$\ldots$	

Computer Algebra 101

Multiplication of polynomials: $\mathrm{M}(n)$

Naive:	$O(n^2)$	
Karatsuba:	$O(n^{\log_2 3}) = O(n^{1.585})$	Karatsuba (1962)
Toom-3:	$O(n^{\log_3 5}) = O(n^{1.465})$	Toom (1963), Cook (1966)
FFT-based algorithms:		
	$O(n \log n)$ with $\omega^{2n} = 1$	Cooley, Tukey (1965)
	$O(n \log n \log\log n)$	Cantor, Kaltofen (1991)
	...	

Other polynomial operations:

Euclidean division:	$O(\mathrm{M}(n))$
GCD:	$O(\mathrm{M}(n) \log n)$
Evaluation & interpolation:	$O(\mathrm{M}(n) \log n)$
...	

Computer Algebra 101

Multiplication of polynomials: $\mathsf{M}(n)$

Naive:	$O(n^2)$	
Karatsuba:	$O(n^{\log_2 3}) = O(n^{1.585})$	Karatsuba (1962)
Toom-3:	$O(n^{\log_3 5}) = O(n^{1.465})$	Toom (1963), Cook (1966)
FFT-based algorithms:		
	$O(n \log n)$ with $\omega^{2n} = 1$	Cooley, Tukey (1965)
	$O(n \log n \log\log n)$	Cantor, Kaltofen (1991)
	...	

Other polynomial operations:

Euclidean division:	$O(\mathsf{M}(n))$
GCD:	$O(\mathsf{M}(n) \log n)$
Evaluation & interpolation:	$O(\mathsf{M}(n) \log n)$
...	

What about space complexity?

Space-complexity models

Algebraic-RAM Machine

- *Standard* registers of size $O(\log n)$
- *Algebraic* registers containing one coefficient

$\rightarrow$ Count *extra* registers used (not input nor output)

Space-complexity models

Algebraic-RAM Machine

- *Standard* registers of size $O(\log n)$
- *Algebraic* registers containing one coefficient

$\rightarrow$ Count *extra* registers used (not input nor output)

Read-write permissions

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication

Space-complexity models

Algebraic-RAM Machine

- *Standard* registers of size $O(\log n)$
- *Algebraic* registers containing one coefficient

$\rightarrow$ Count *extra* registers used (not input nor output)

Read-write permissions

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication
- Read-write input and output
  - Too permissive in general
  - Variant: inputs must be restored at the end

Space-complexity models

Algebraic-RAM Machine

- *Standard* registers of size $O(\log n)$
- *Algebraic* registers containing one coefficient

$\rightarrow$ Count *extra* registers used (not input nor output)

Read-write permissions

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication
- Read-write input and output
  - Too permissive in general
  - Variant: inputs must be restored at the end
- Read-only input / read-write output
  - *Reasonable* from a programmer's viewpoint

Space-complexity models

Algebraic-RAM Machine

- *Standard* registers of size $O(\log n)$
- *Algebraic* registers containing one coefficient

$\rightarrow$ Count *extra* registers used (not input nor output)

Read-write permissions

- Read-only input / write-only output
  - (Close to) classical complexity theory
  - Lower bound $\Omega(n^2)$ on time $\times$ space for multiplication
- Read-write input and output
  - Too permissive in general
  - Variant: inputs must be restored at the end
- ✓ Read-only input / read-write output
  - *Reasonable* from a programmer's viewpoint

Space complexity of multiplication algorithms

	space	time
Naive algorithm:	$O(1)$	$O(n^2)$

Space complexity of multiplication algorithms

	space	time
Naive algorithm:	$O(1)$	$O(n^2)$
Karatsuba's algorithm:		
Original (1962)	$O(n)$	$\sim 6.5n^{\log 3}$
Thomé (2002)	$n + O(\log n)$	$\sim 7n^{\log 3}$
Roche (2009)	$O(\log n)$	$\sim 10n^{\log 3}$

Space complexity of multiplication algorithms

	space	time
Naive algorithm:	$O(1)$	$O(n^2)$
Karatsuba's algorithm:		
Original (1962)	$O(n)$	$\sim 6.5n^{\log 3}$
Thomé (2002)	$n + O(\log n)$	$\sim 7n^{\log 3}$
Roche (2009)	$O(\log n)$	$\sim 10n^{\log 3}$
Toom-Cook algorithms:		
Toom-3 (1963)	$O(n)$	$\sim \frac{73}{4} n^{\log_3 5}$

Space complexity of multiplication algorithms

	space	time
Naive algorithm:	$O(1)$	$O(n^2)$
Karatsuba's algorithm:		
Original (1962)	$O(n)$	$\sim 6.5n^{\log 3}$
Thomé (2002)	$n + O(\log n)$	$\sim 7n^{\log 3}$
Roche (2009)	$O(\log n)$	$\sim 10n^{\log 3}$
Toom-Cook algorithms:		
Toom-3 (1963)	$O(n)$	$\sim \frac{73}{4} n^{\log_3 5}$
FFT/TFT-based algorithms (given $\omega^{2n} = 1$)**:**		
Original (1965)	$O(n)$	$\sim 9n \log(2n)$
Roche (2009) if $n = 2^k$	$O(1)$	$\sim 11n \log(2n)$
Harvey, Roche (2010)	$O(1)$	$O(n \log(n))$

Our problematic

Can *every* polynomial multiplication algorithm be performed without extra memory?

Our problematic

Can *every* polynomial multiplication algorithm be performed without extra memory?

- Karatsuba? Toom-Cook?
- FFT/TFT without $\omega^{2n} = 1$?
- What about other products (short and middle)?

Our problematic

Can *every* polynomial multiplication algorithm be performed without extra memory?

- Karatsuba? Toom-Cook?
- FFT/TFT without $\omega^{2n} = 1$?
- What about other products (short and middle)?

Results:

- Yes!
- Almost (for other products)

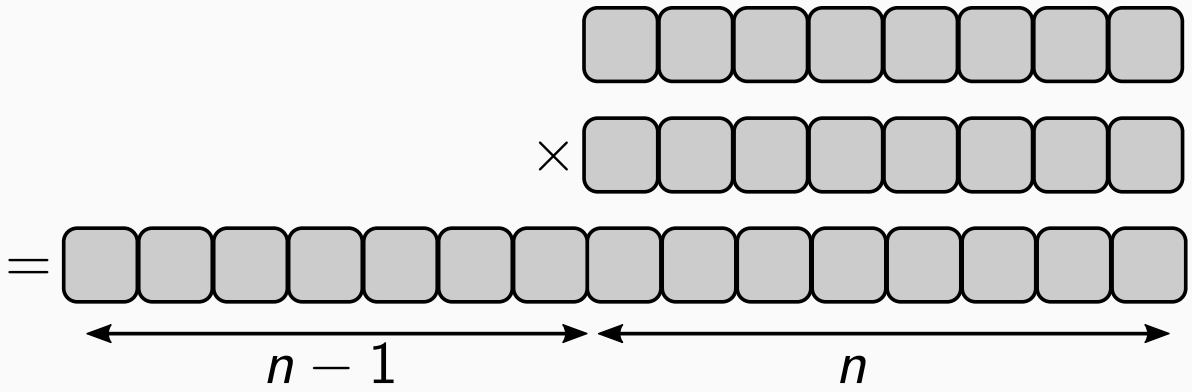
Contents

Space-preserving reductions between products

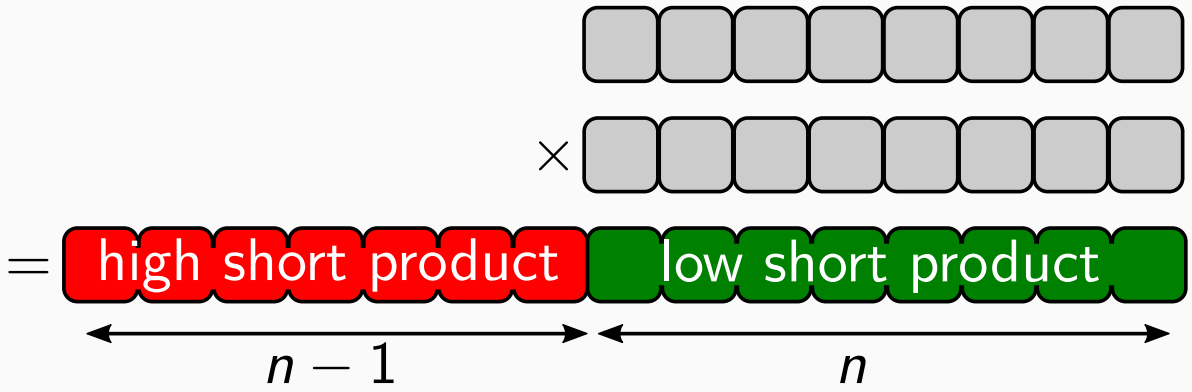
Self-reductions: In-place algorithms from out-of-place algorithms

Space-preserving reductions between products

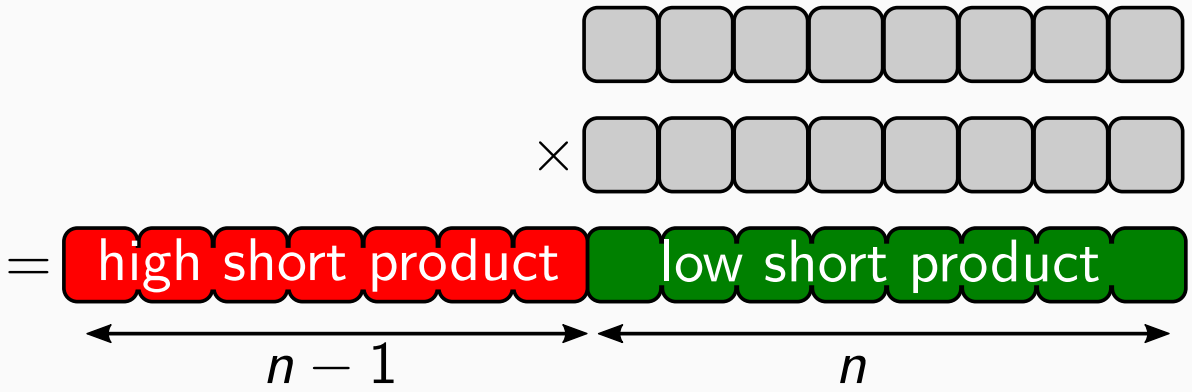
Short product



Short product



Short product

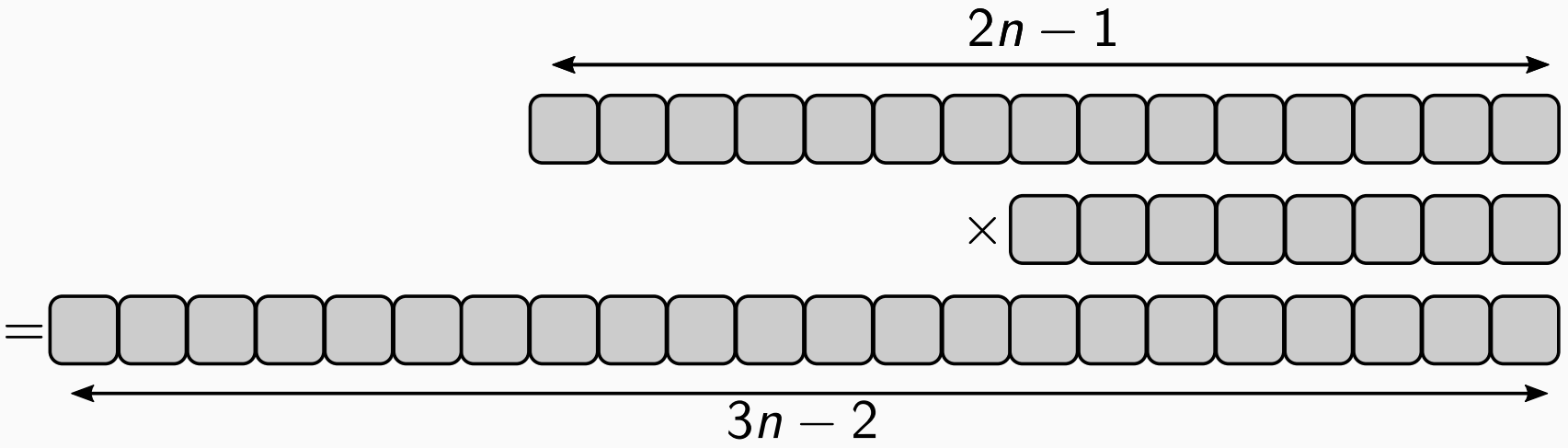


Formal definition

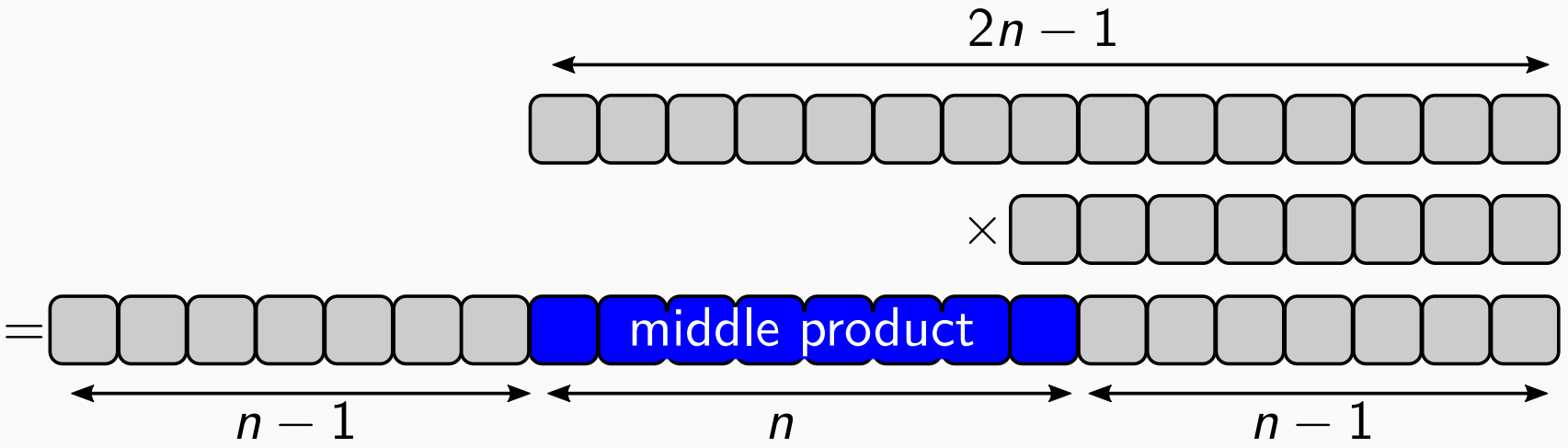
- $\mathrm{SP}_{\mathrm{lo}}(f, g) = f \cdot g \mod X^n$
- $\mathrm{SP}_{\mathrm{hi}}(f, g) = f \cdot g \operatorname{div} X^n$

Example of use
Product of truncated power series

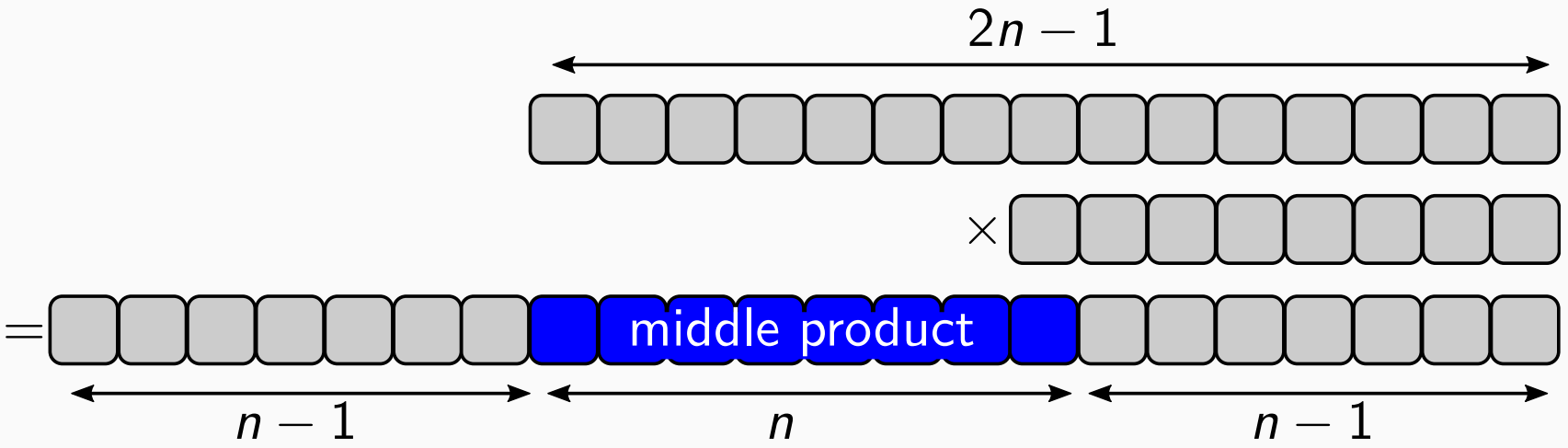
Middle product



Middle product



Middle product



Formal definition

$\mathrm{MP}(f, g) = (f \cdot g \text{ div } X^{n-1}) \mod X^n$

Example of use

Newton iteration (division, square root, ...)

Relative difficulties of products

Notations

- FP: *full* (standard) product
- $\mathrm{FP}(n)$, $\mathrm{SP}_{\mathsf{lo}}(n)$, $\mathrm{SP}_{\mathsf{hi}}(n)$, $\mathrm{MP}(n)$: complexity for size-n inputs

Relative difficulties of products

Notations

- FP: *full* (standard) product
- $\mathsf{FP}(n)$, $\mathsf{SP}_{\mathsf{lo}}(n)$, $\mathsf{SP}_{\mathsf{hi}}(n)$, $\mathsf{MP}(n)$: complexity for size-n inputs

Classical results (no space restriction)

- $\mathsf{SP}_{\mathsf{lo}}(n) \leq \mathsf{FP}(n)$ and $\mathsf{SP}_{\mathsf{hi}}(n) \leq \mathsf{FP}(n-1)$
- $\mathsf{FP}(n) \leq \mathsf{SP}_{\mathsf{lo}}(n) + \mathsf{SP}_{\mathsf{hi}}(n)$
- $\mathsf{MP}(n) = \mathsf{FP}(n) + n - 1$ (transposition principle)

Relative difficulties of products

Notations

- FP: *full* (standard) product
- $\mathsf{FP}(n)$, $\mathsf{SP}_{\mathsf{lo}}(n)$, $\mathsf{SP}_{\mathsf{hi}}(n)$, $\mathsf{MP}(n)$: complexity for size-n inputs

Classical results (no space restriction)

- $\mathsf{SP}_{\mathsf{lo}}(n) \leq \mathsf{FP}(n)$ and $\mathsf{SP}_{\mathsf{hi}}(n) \leq \mathsf{FP}(n-1)$
- $\mathsf{FP}(n) \leq \mathsf{SP}_{\mathsf{lo}}(n) + \mathsf{SP}_{\mathsf{hi}}(n)$
- $\mathsf{MP}(n) = \mathsf{FP}(n) + n - 1$ (transposition principle)

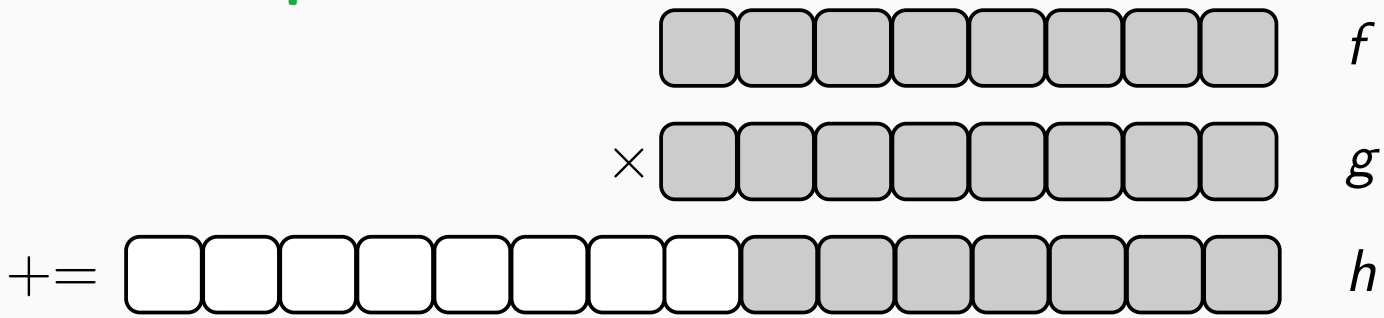
Space-restricted settings

✗ Reductions unusable

✓ Reduction still valid

Relative difficulties in space-restricted settings

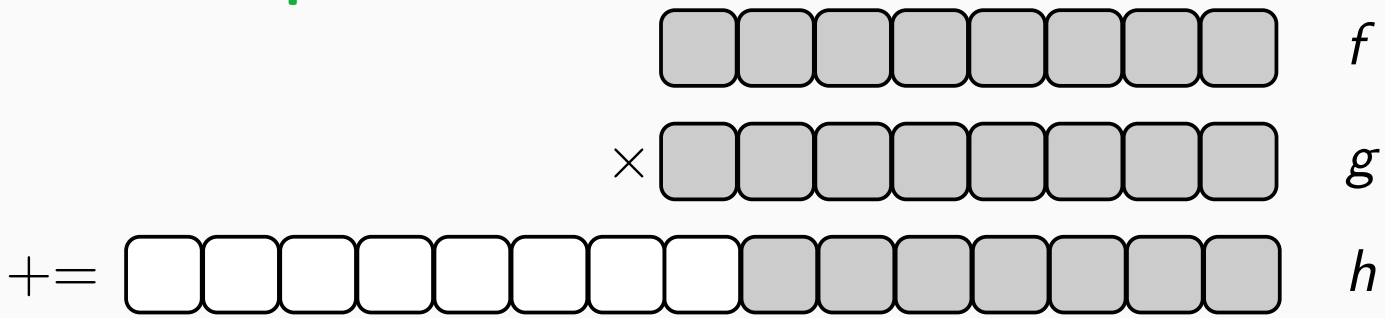
Half-additive full product FP^+



Add $f \cdot g$ to h where $\deg(h) < \deg(f), \deg(g)$

Relative difficulties in space-restricted settings

Half-additive full product FP^+



Add $f \cdot g$ to h where $\deg(h) < \deg(f), \deg(g)$

Theorem ("$\mathsf{SP} \equiv \mathsf{FP}^+$")
The following reductions do not increase space:

- $\mathsf{FP}^+(n) \leq \mathsf{SP}_{\mathsf{lo}}(n) + \mathsf{SP}_{\mathsf{hi}}(n) + n - 1$
- $\mathsf{SP}_{\mathsf{lo}}(n) \leq 3\mathsf{FP}^+(\lceil n/2 \rceil)$

Summary of relative difficulties

Without space restrictions

$$\mathsf{SP} \equiv \mathsf{FP}^{(+)} \equiv \mathsf{MP}$$

In space-restricted settings

$$\mathsf{FP} \leq \begin{pmatrix} \mathsf{SP} \\ \| \\ \mathsf{FP}^{+} \end{pmatrix} \leq \mathsf{MP}$$

Summary of relative difficulties

Without space restrictions

$$\mathsf{SP} \equiv \mathsf{FP}^{(+)} \equiv \mathsf{MP}$$

In space-restricted settings

$$\mathsf{FP} \leq \begin{pmatrix} \mathsf{SP} \\ \| \\ \mathsf{FP}^{+} \end{pmatrix} \leq \mathsf{MP}$$

Converse directions

- $\mathsf{SP}(n) \leq \mathsf{FP}(n)$?
  - problem with output size : n v. $2n-1$
  - without space restrictions: is $\mathsf{SP}(n) \leq \mathsf{FP}(n/2)$?
- $\mathsf{FP} \equiv \mathsf{MP}$?
  - partial result: $\mathsf{MP}(n) \leq \mathsf{FP}(n) \times \log(n)$
  - related: space-preserving transposition? Kaltofen (2000)

Self-reductions: In-place algorithms from out-of-place algorithms

Framework

In-place algorithms parametrized by out-of-place algorithms

- Assumption: Out-of-place alg. uses cn extra space
- Constant c known to the in-place algorithm

Framework

In-place algorithms parametrized by out-of-place algorithms

- Assumption: Out-of-place alg. uses cn extra space
- Constant c known to the in-place algorithm

Goal

- Space complexity: $O(1)$
- Time complexity: closest to the out-of-place algorithm

Framework

In-place algorithms parametrized by out-of-place algorithms

- Assumption: Out-of-place alg. uses cn extra space
- Constant c known to the in-place algorithm

Goal

- Space complexity: $O(1)$
- Time complexity: closest to the out-of-place algorithm

Techniques

- Oracle calls in smaller size
- *Fake* padding of inputs (*cf.* strides in lin. alg.)
- **Tail** recursive call (avoid $O(\log n)$ stack)

Framework

In-place algorithms parametrized by out-of-place algorithms

- Assumption: Out-of-place alg. uses cn extra space
- Constant c known to the in-place algorithm

Goal

- Space complexity: $O(1)$
- Time complexity: closest to the out-of-place algorithm

Techniques

- Oracle calls in smaller size
- *Fake* padding of inputs (*cf.* strides in lin. alg.)
- **Tail** recursive call (avoid $O(\log n)$ stack)

Similar approach for matrix mul.: Boyer, Dumas, Pernet, Zhou (2009)

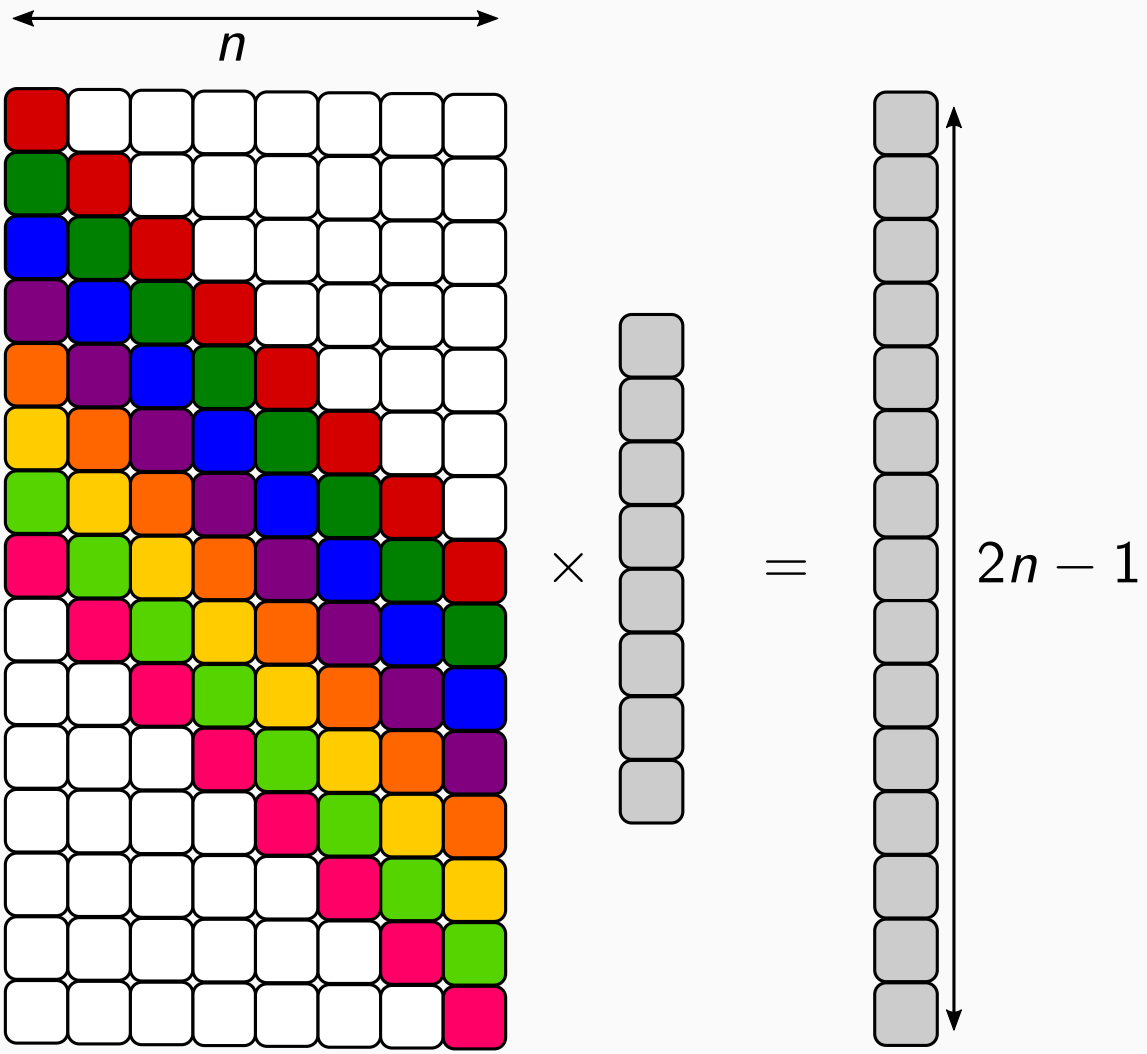
Our results

Theorem

- In-place (half-additive) full product in time $(2c + 7)\mathrm{M}(n)$
- In-place short product in time $(2c + 5)\mathrm{M}(n)$
- In-place middle product in time $O(\mathrm{M}(n) \log n)$
  (or $O(\mathrm{M}(n))$ if $\mathrm{M}(n) = \Omega(n^{1+\delta})$)

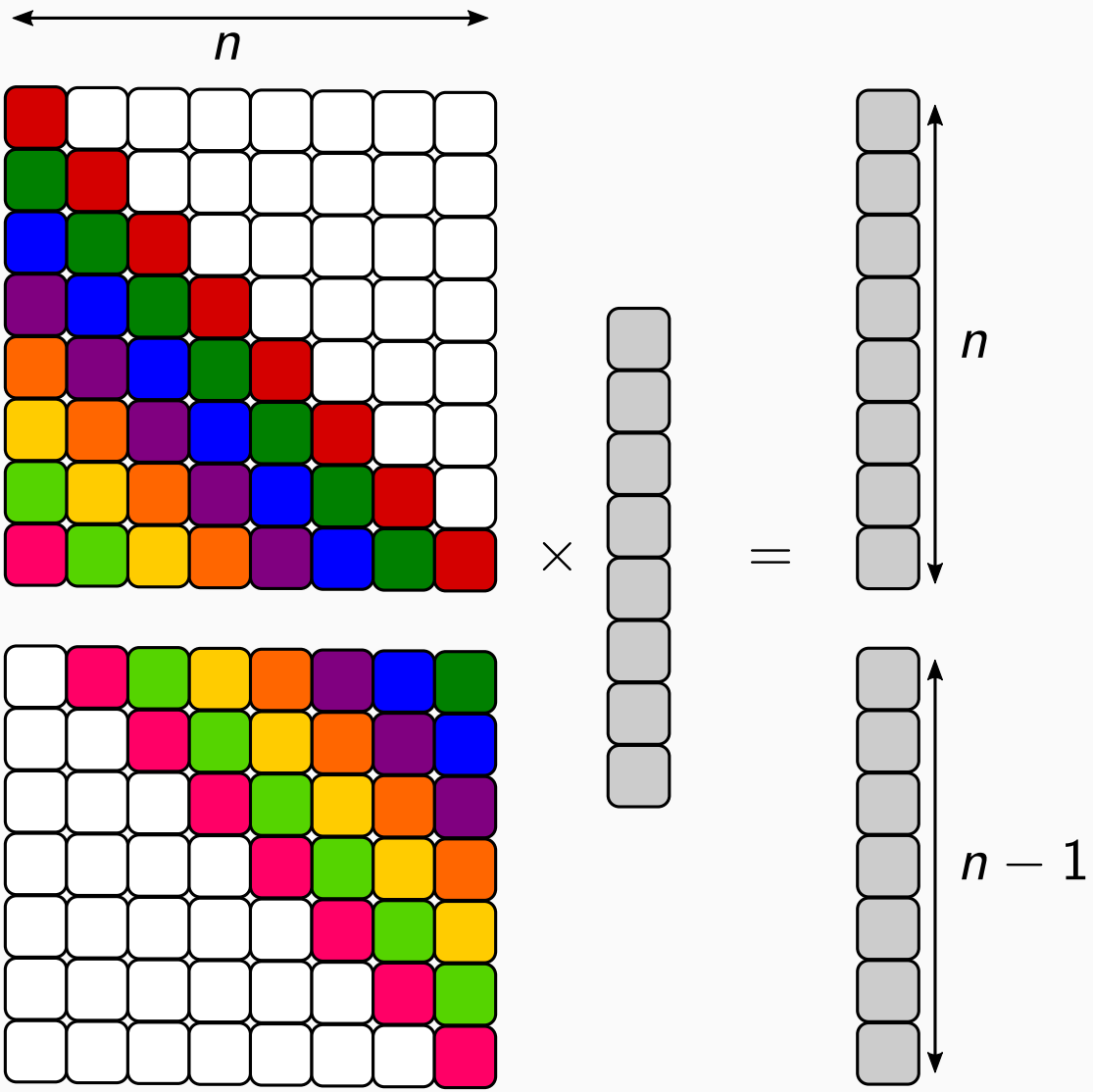
Multiplications as linear maps

Full product:



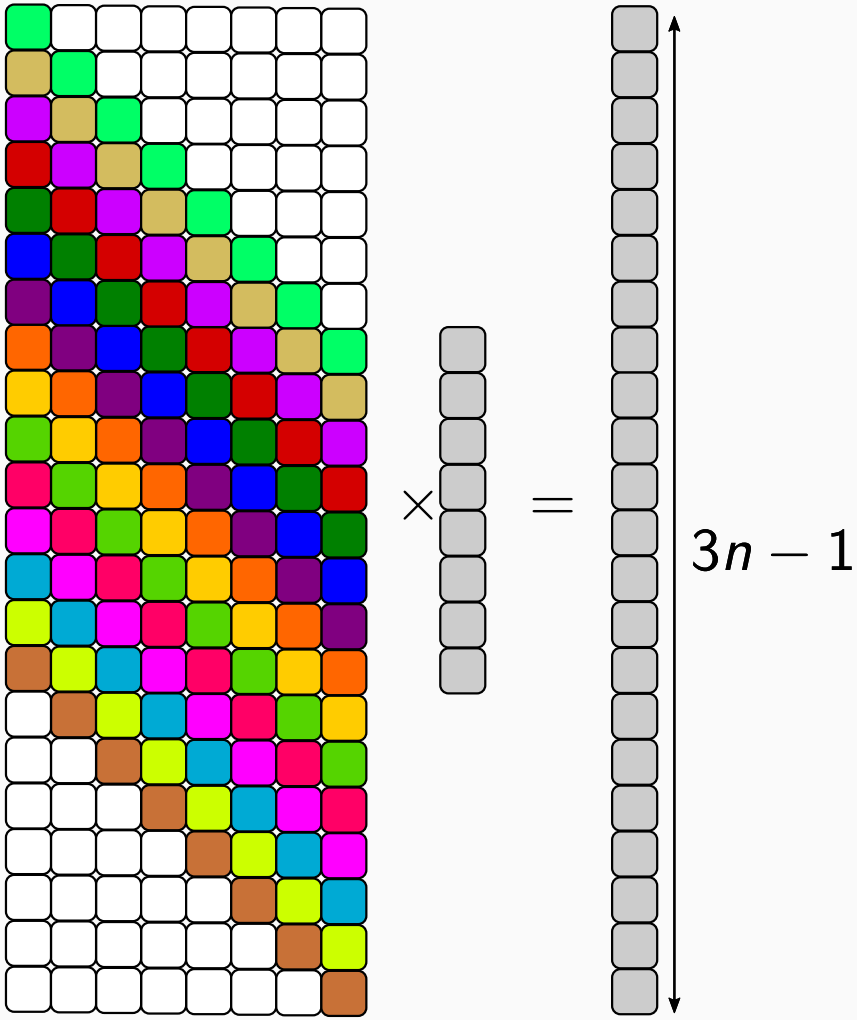
Multiplications as linear maps

Short products:



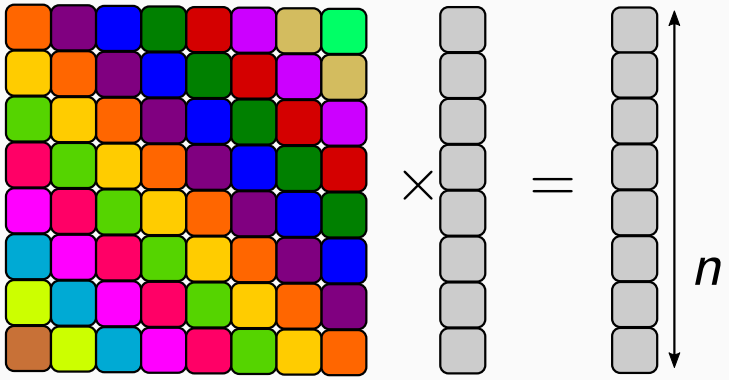
Multiplications as linear maps

Middle product:



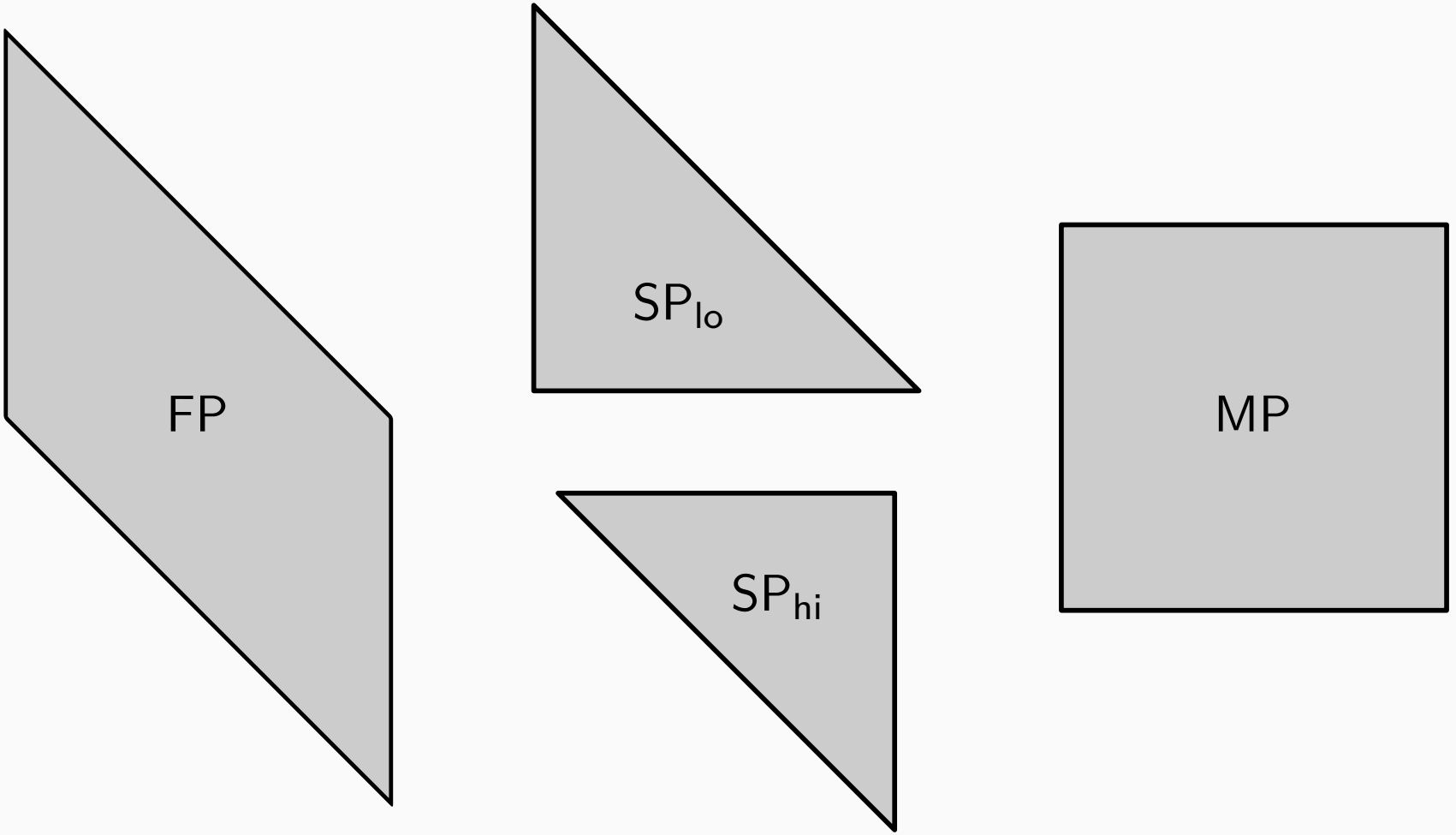
Multiplications as linear maps

Middle product:



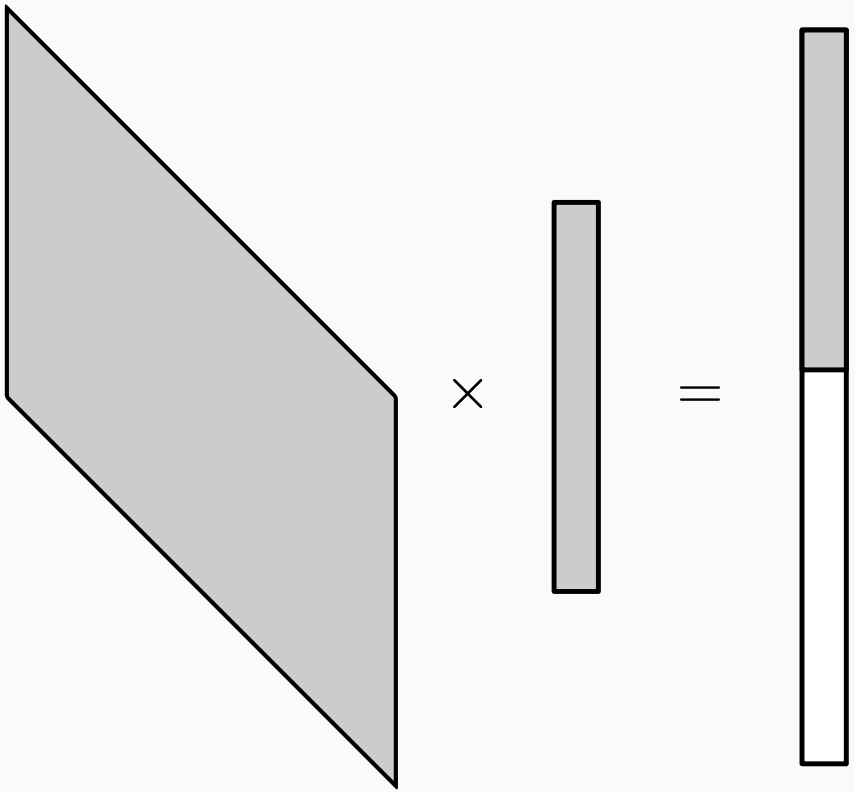
Multiplications as linear maps

Graphical representation:

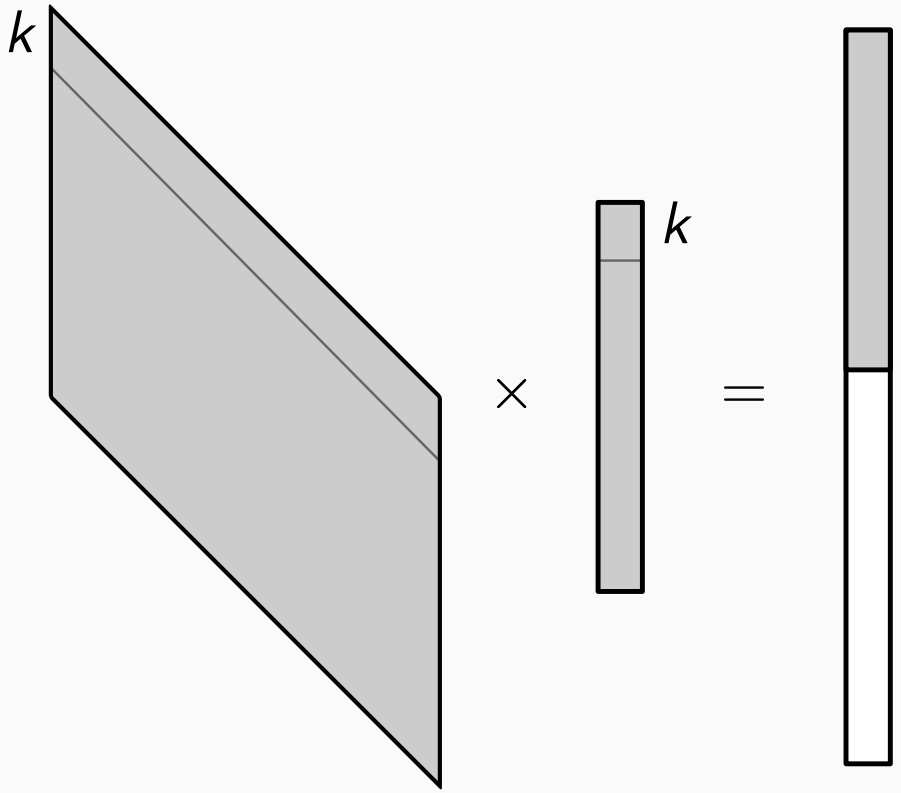


Full product Short products Middle product

In-place FP^+ from out-of-place FP

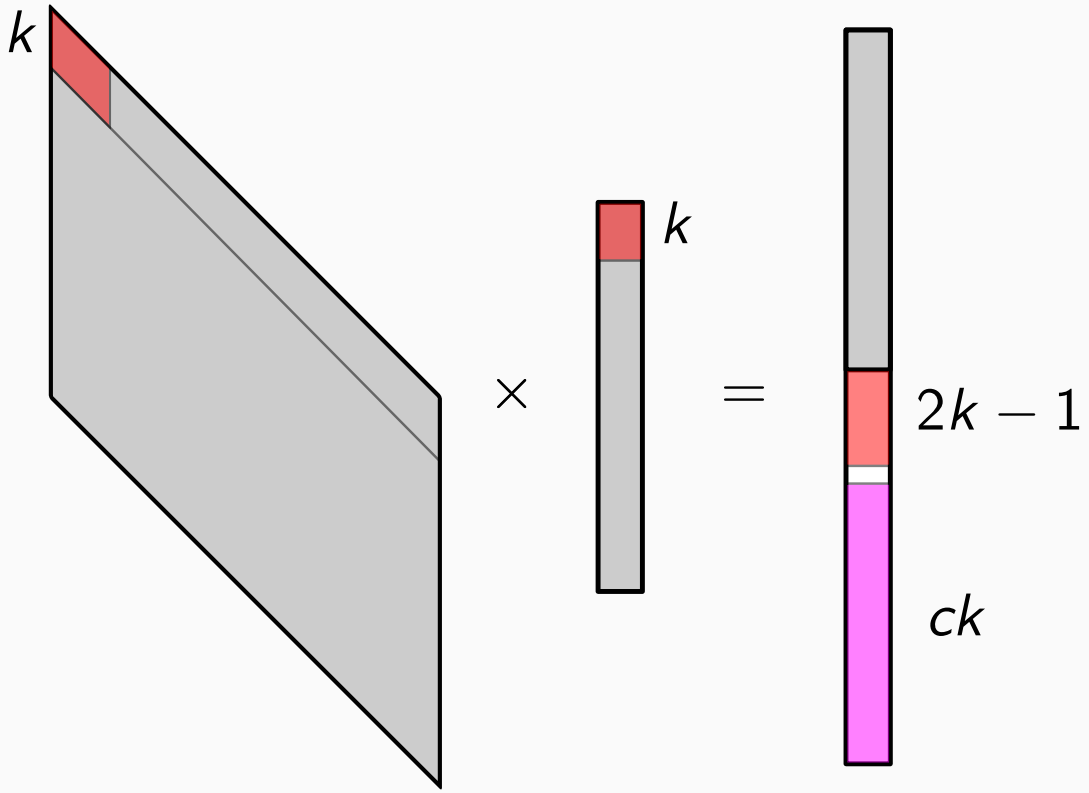


In-place FP$^+$ from out-of-place FP



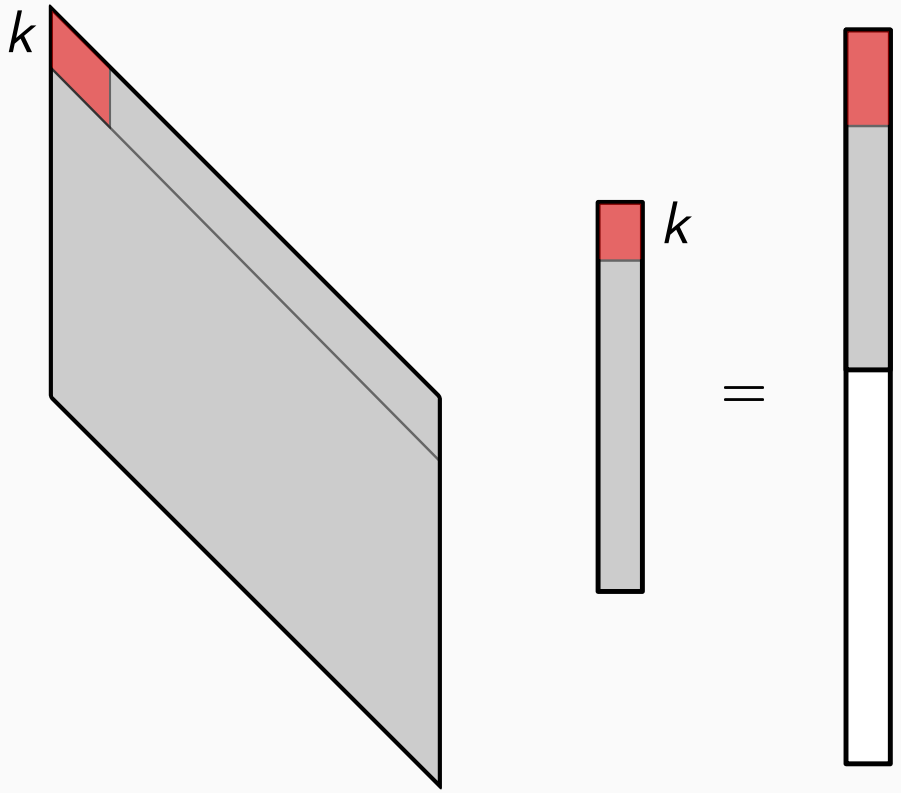
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP



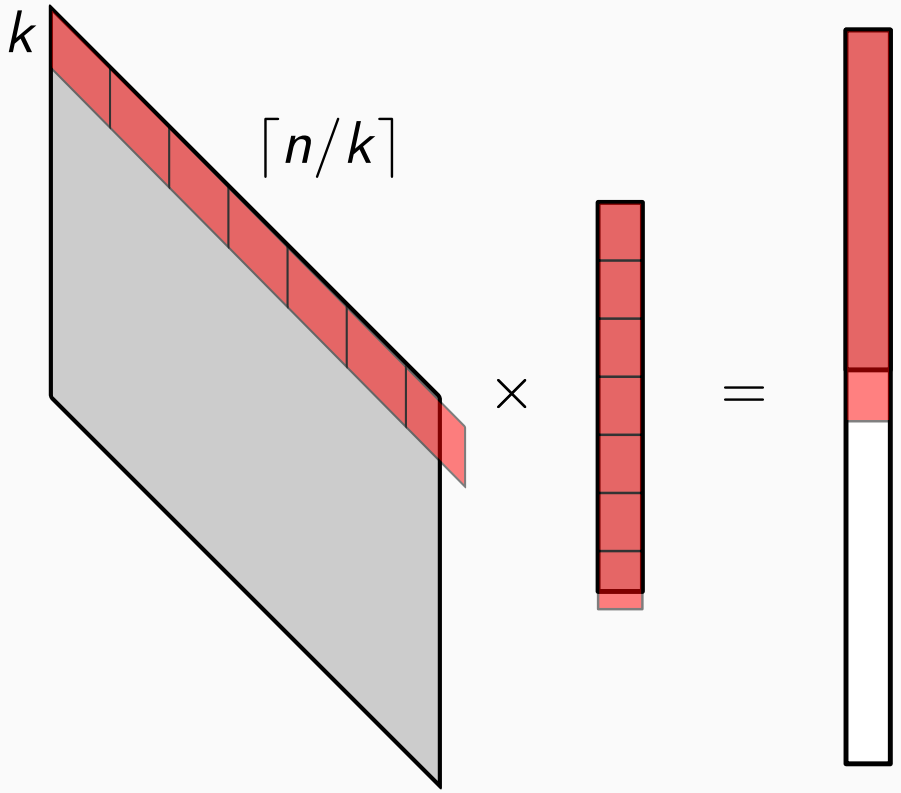
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP



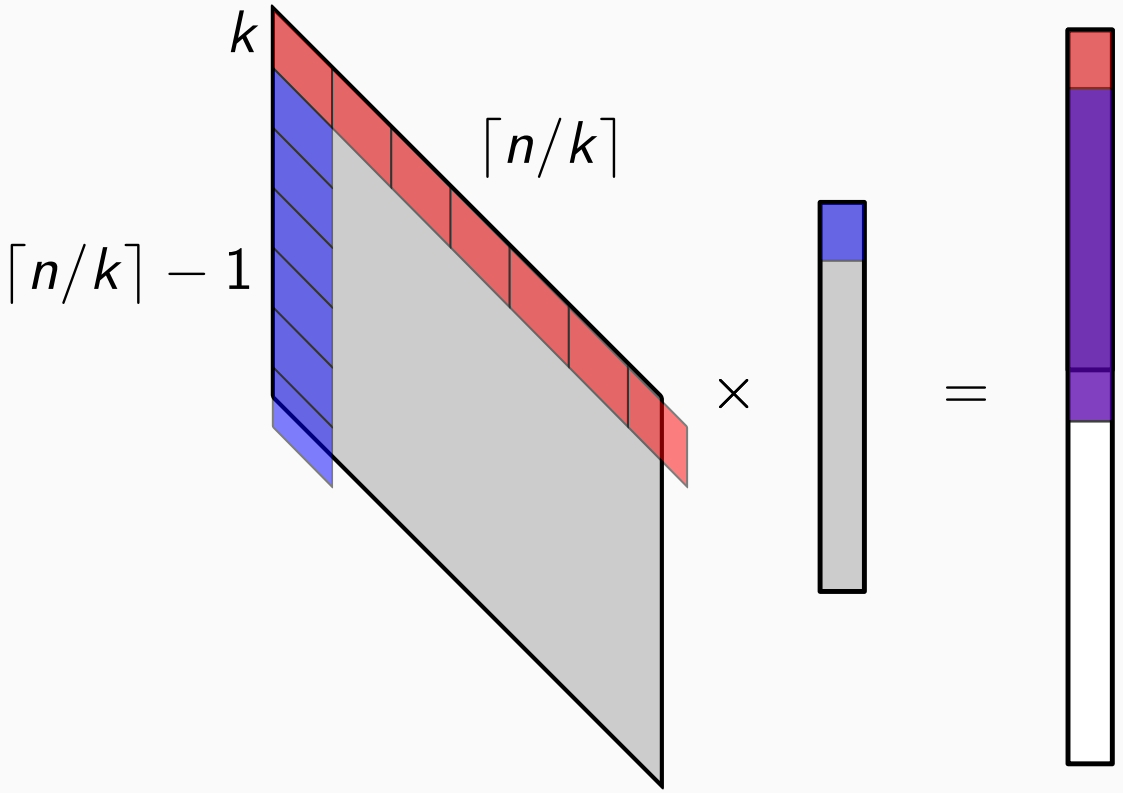
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP



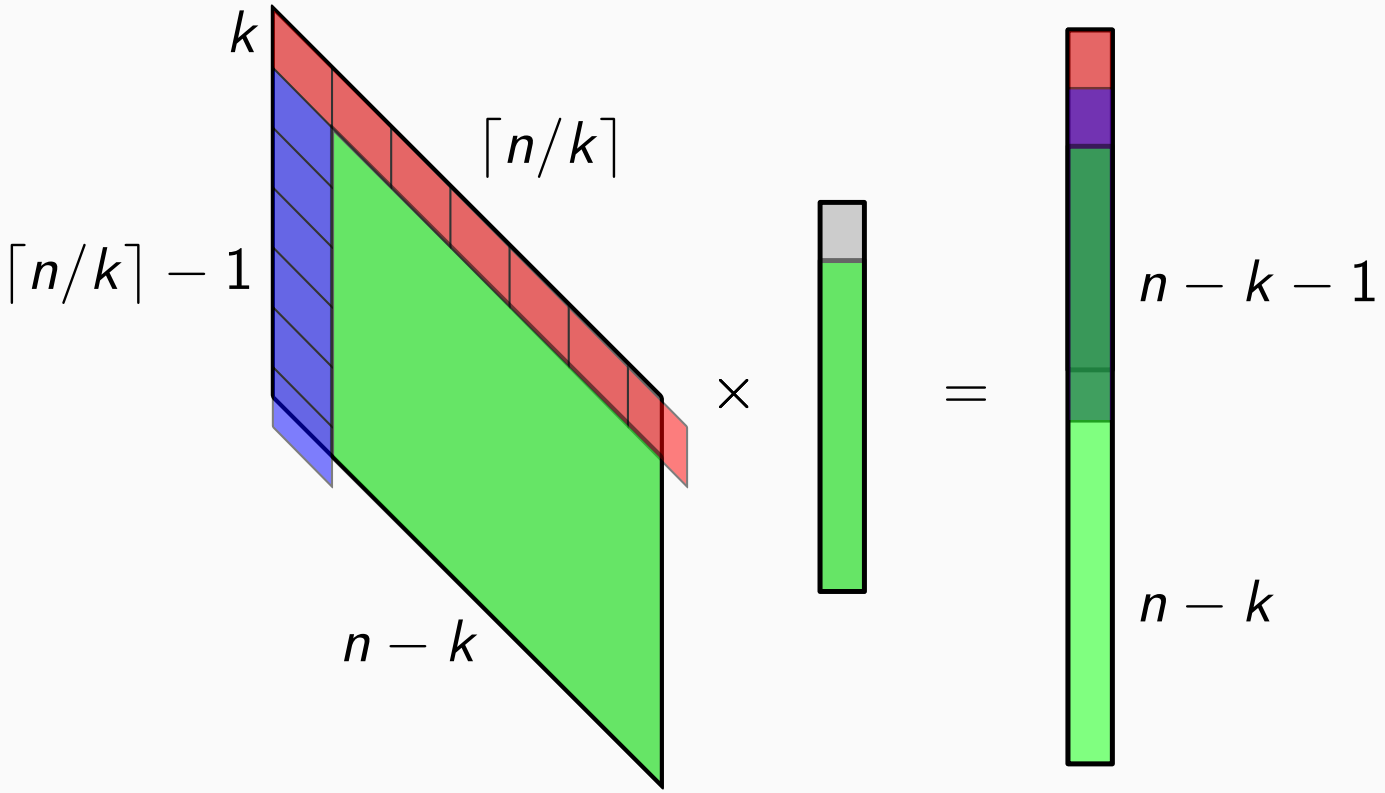
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP



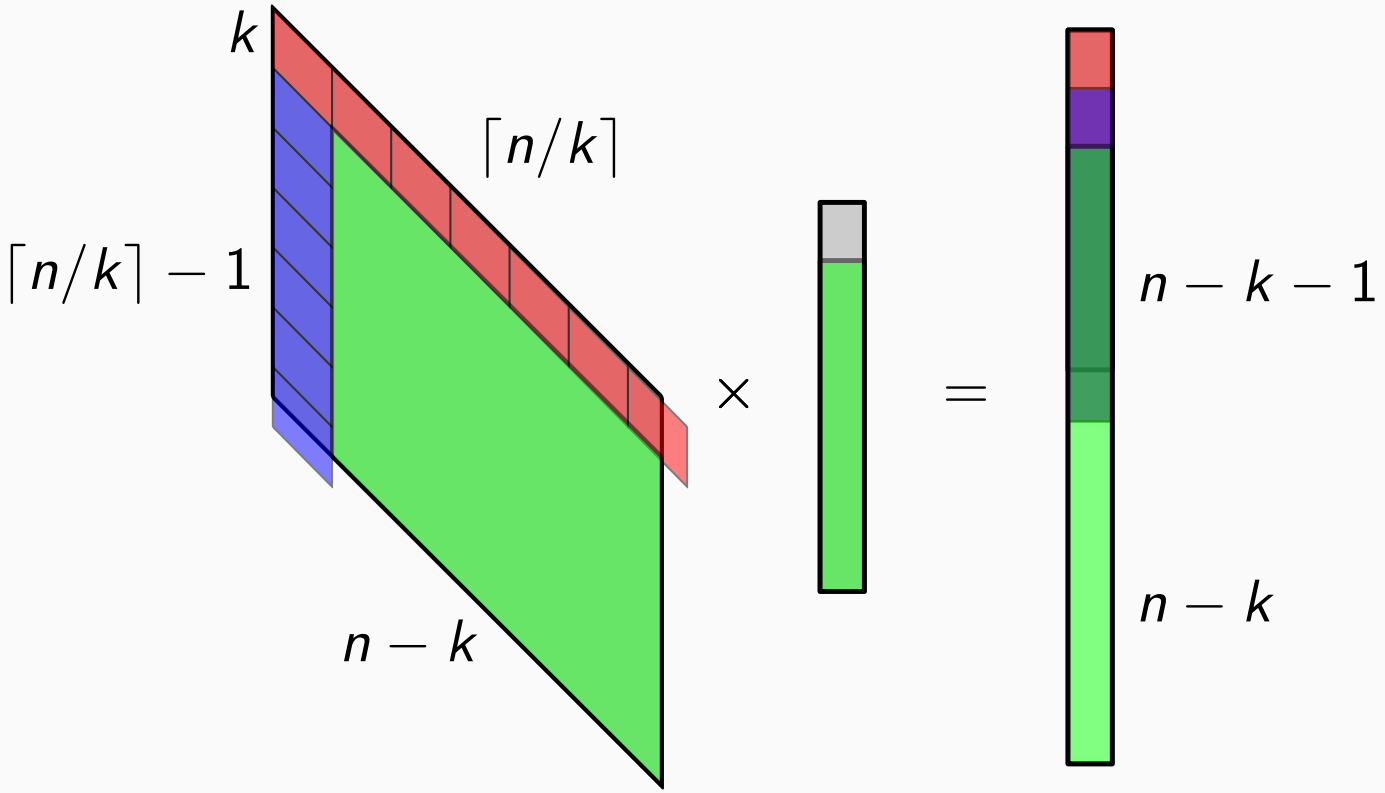
$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

In-place FP$^+$ from out-of-place FP

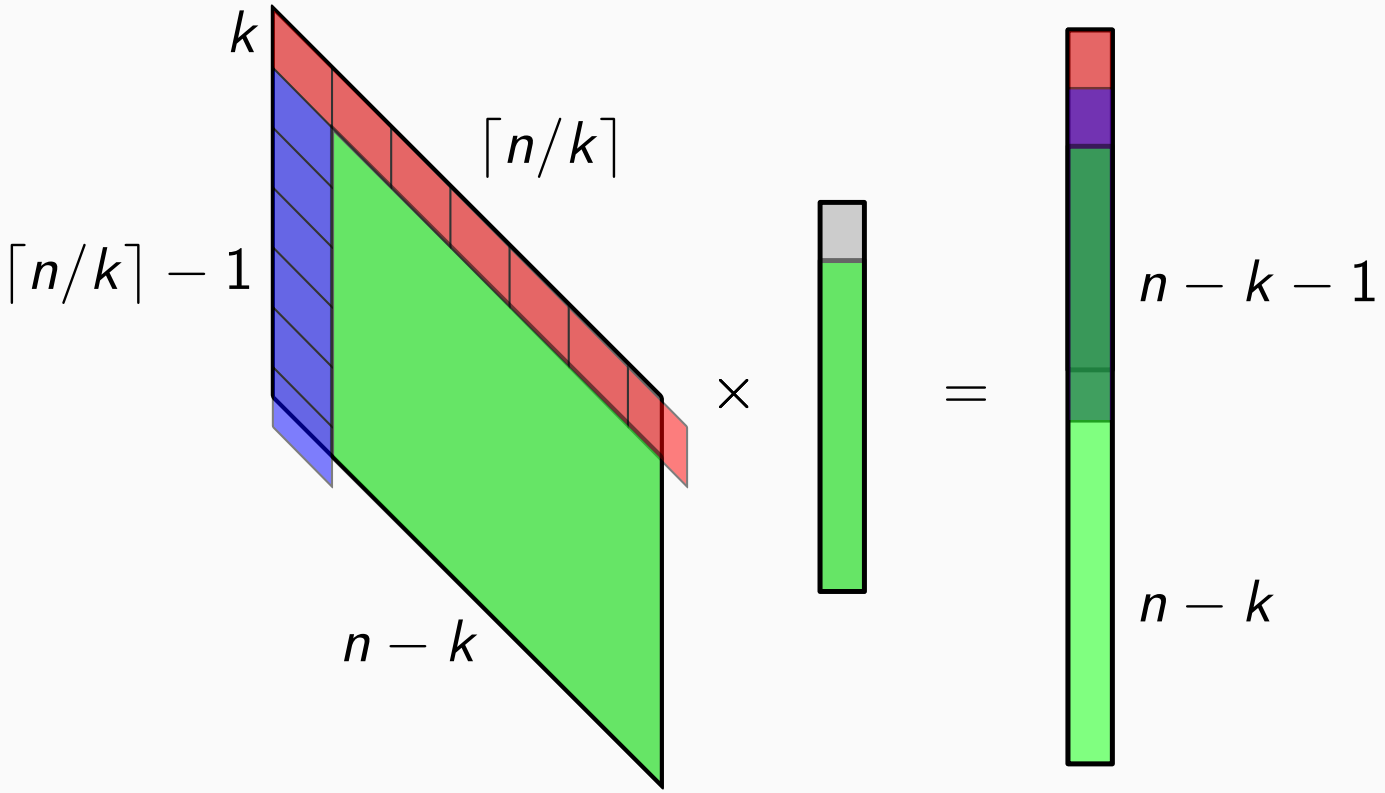


$$(f_0 + X^k \hat{f}) \cdot (g_0 + X^k \hat{g}) = f_0 g_0 + X^k (f_0 \hat{g} + \hat{f} g_0) + X^{2k} \hat{f} \hat{g}$$

Analysis

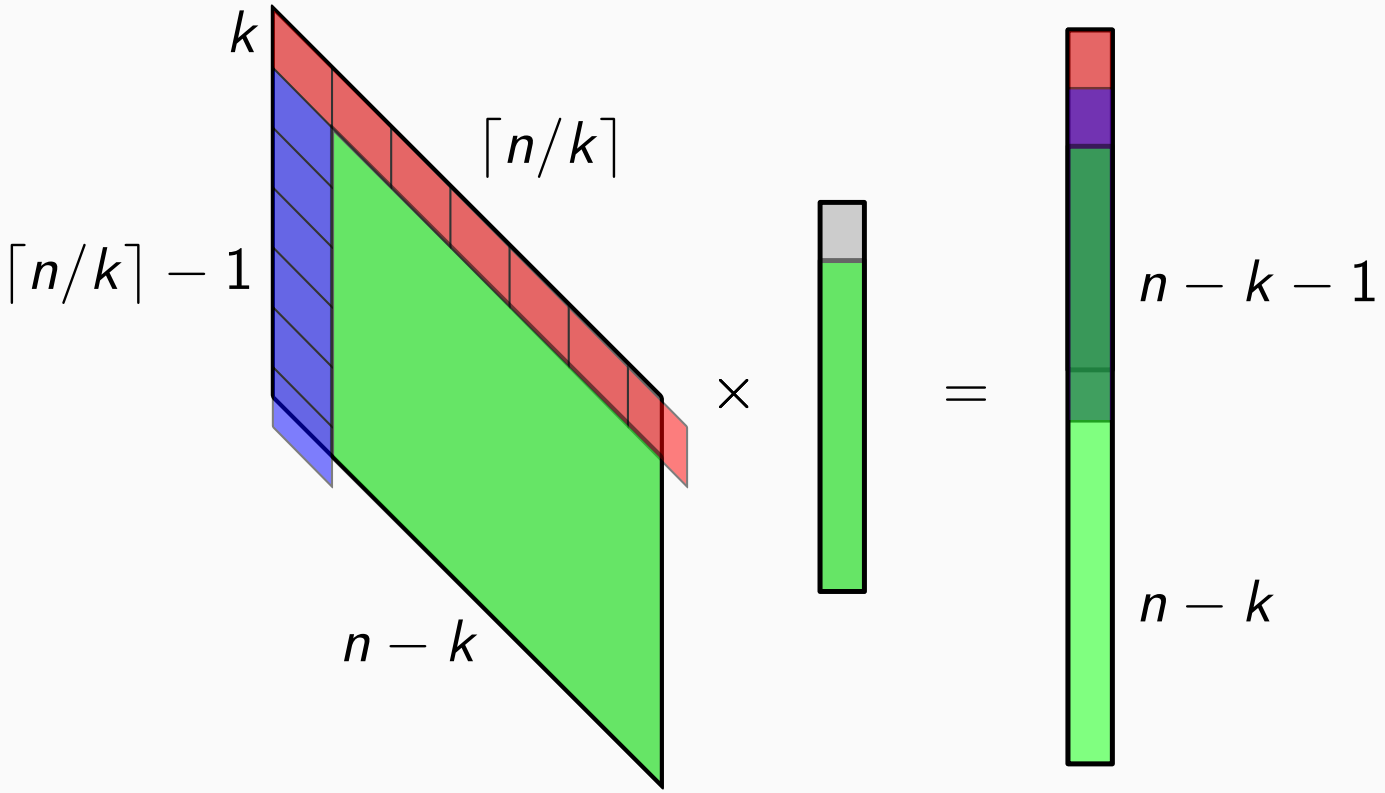


Analysis



- $ck + 2k - 1 \leq n - k \implies k \leq \frac{n+1}{c+3}$
- $T(n) = (2\lceil n/k \rceil - 1)(\mathrm{M}(k) + 2k - 1) + T(n - k)$

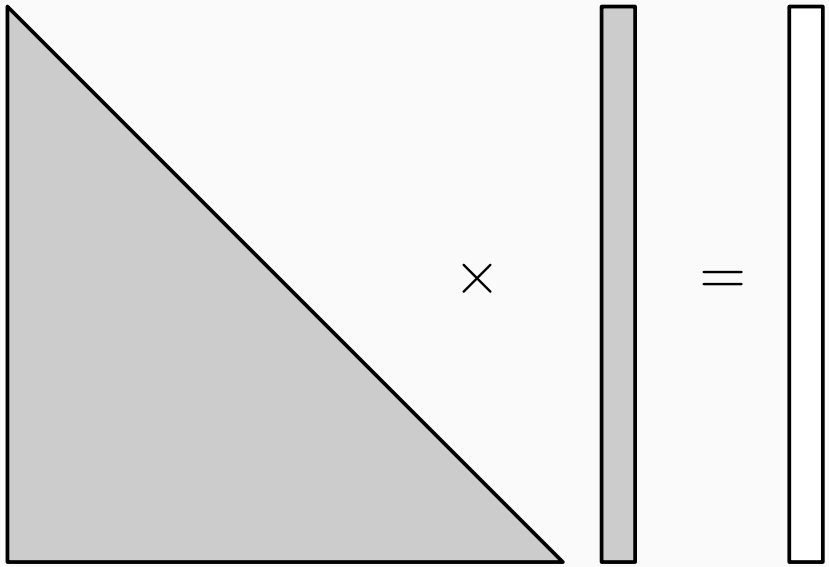
Analysis



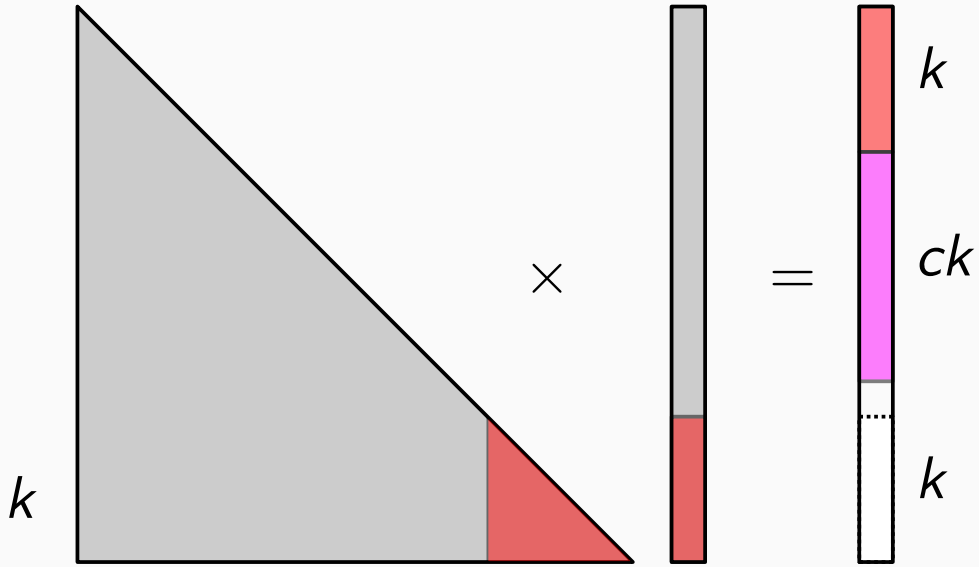
- $ck + 2k - 1 \leq n - k \implies k \leq \frac{n+1}{c+3}$
- $T(n) = (2\lceil n/k \rceil - 1)(\mathrm{M}(k) + 2k - 1) + T(n - k)$

$$T(n) \leq (2c + 7)\mathrm{M}(n) + o(\mathrm{M}(n))$$

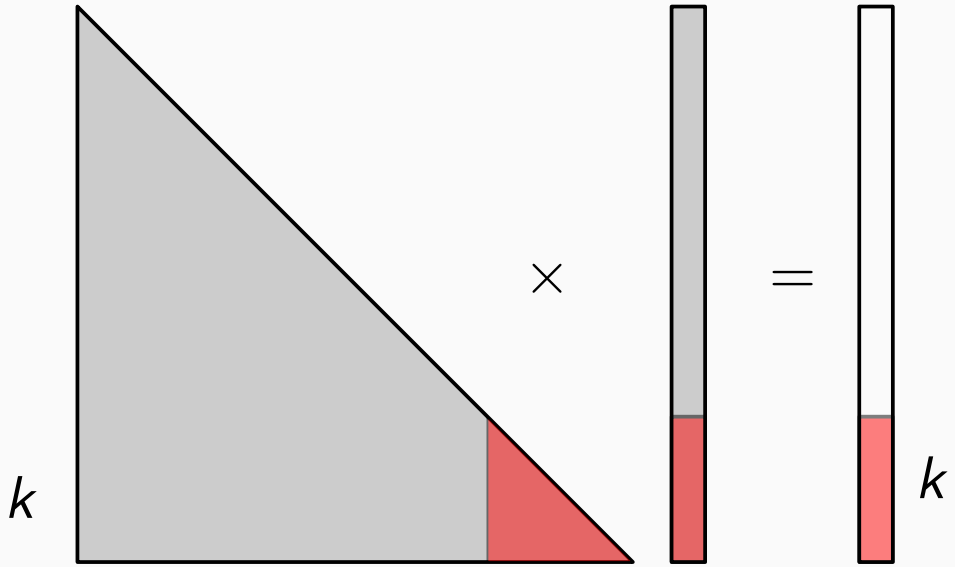
In-place short product



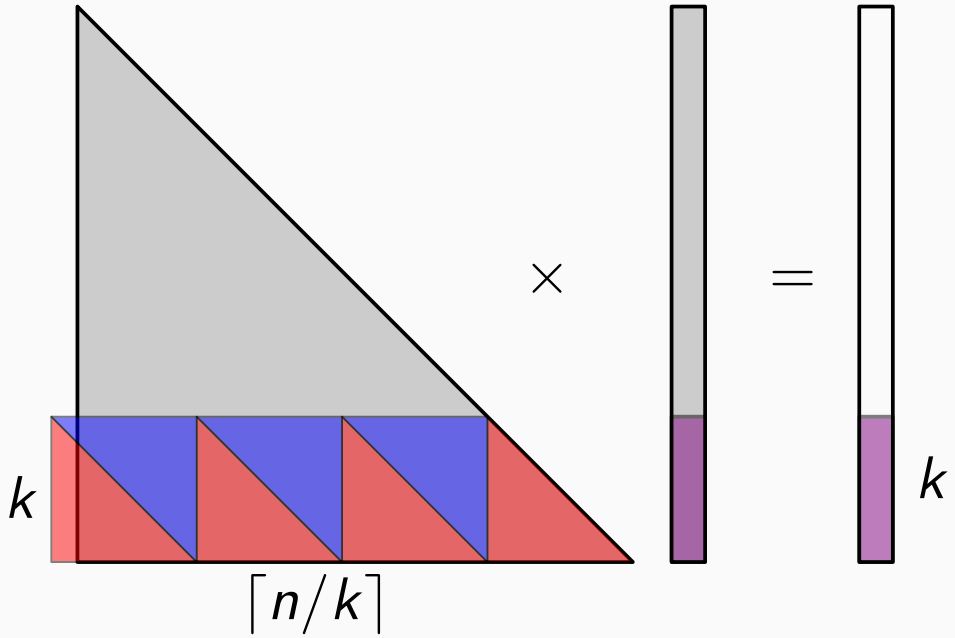
In-place short product



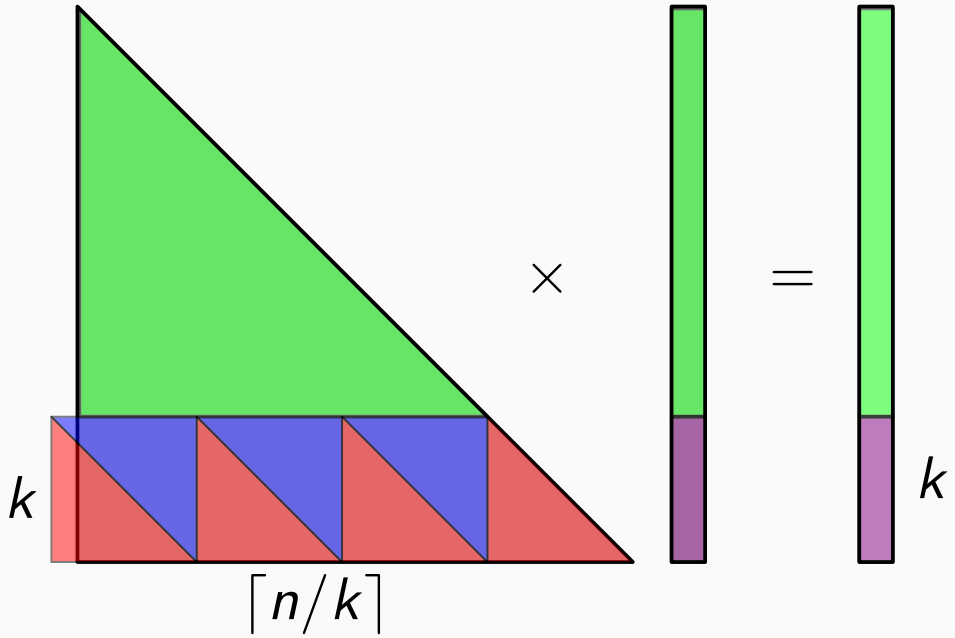
In-place short product



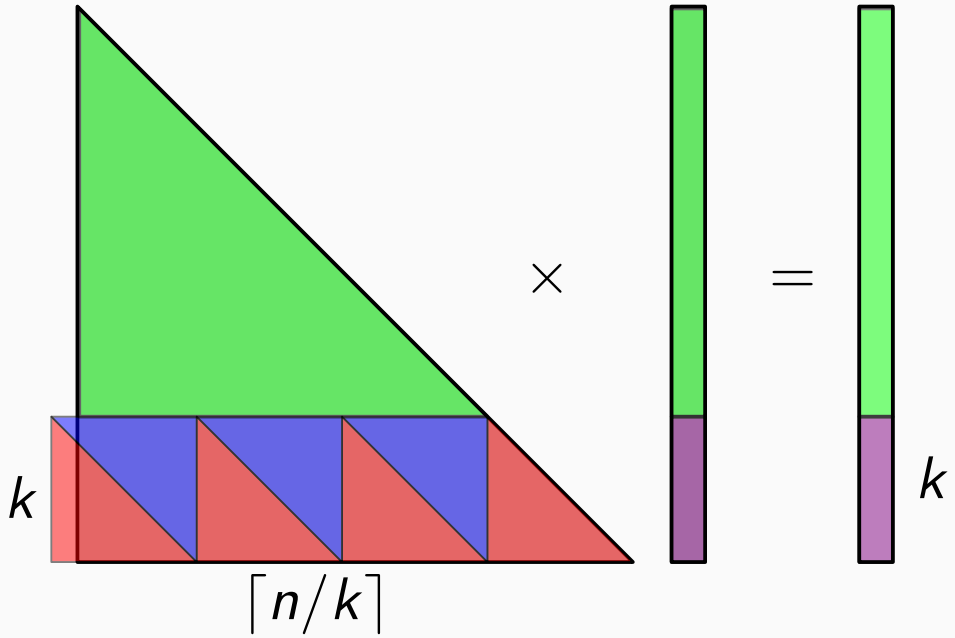
In-place short product



In-place short product

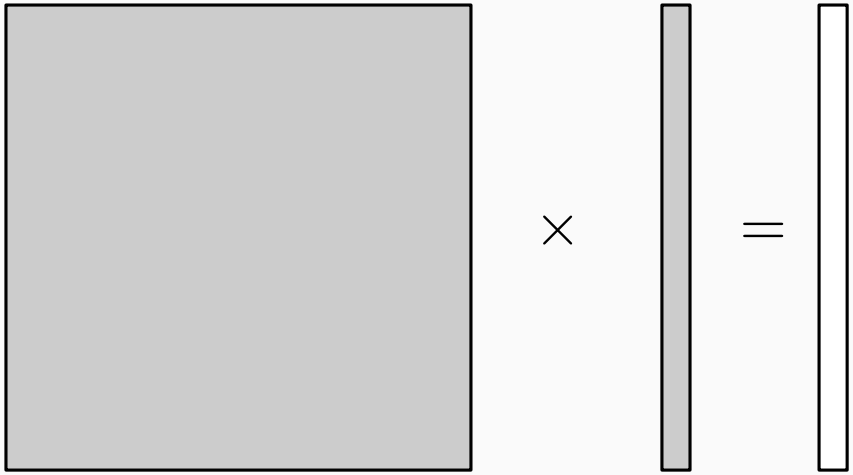


In-place short product

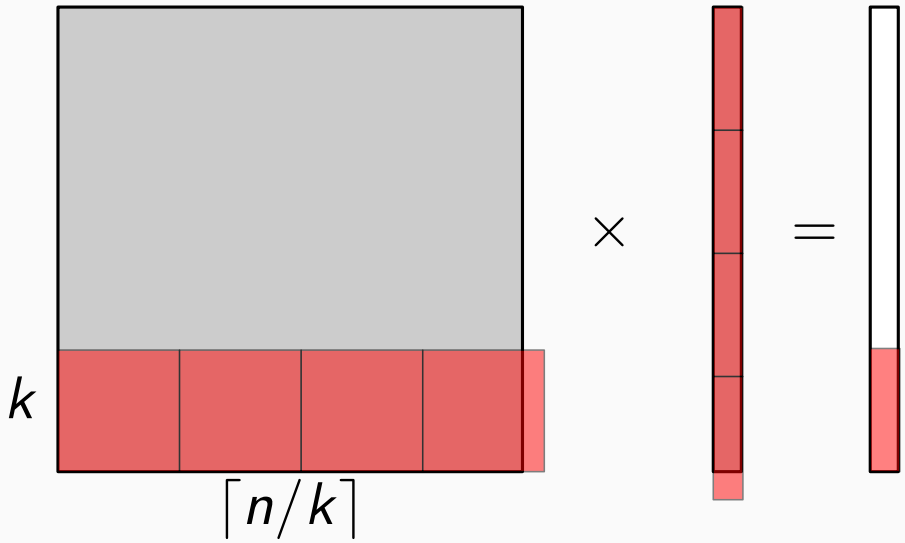


$$T(n) \leq (2c + 5)\mathrm{M}(n) + o(\mathrm{M}(n))$$

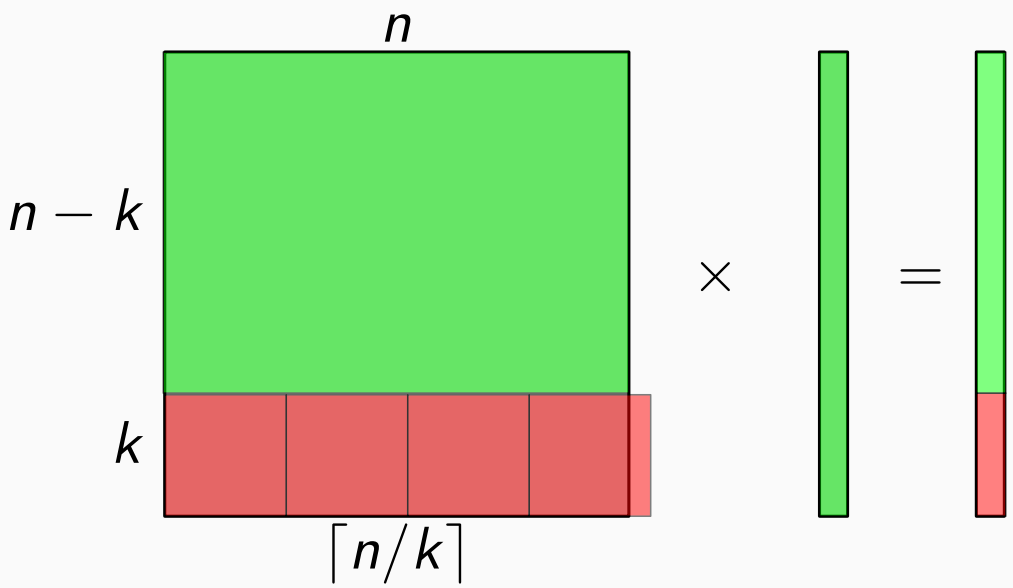
In-place middle product



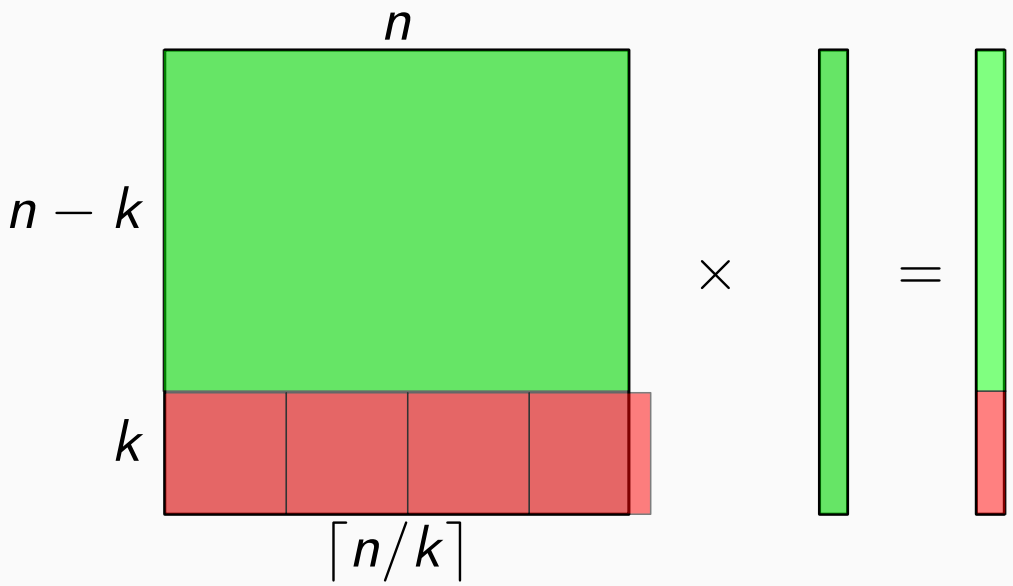
In-place middle product



In-place middle product

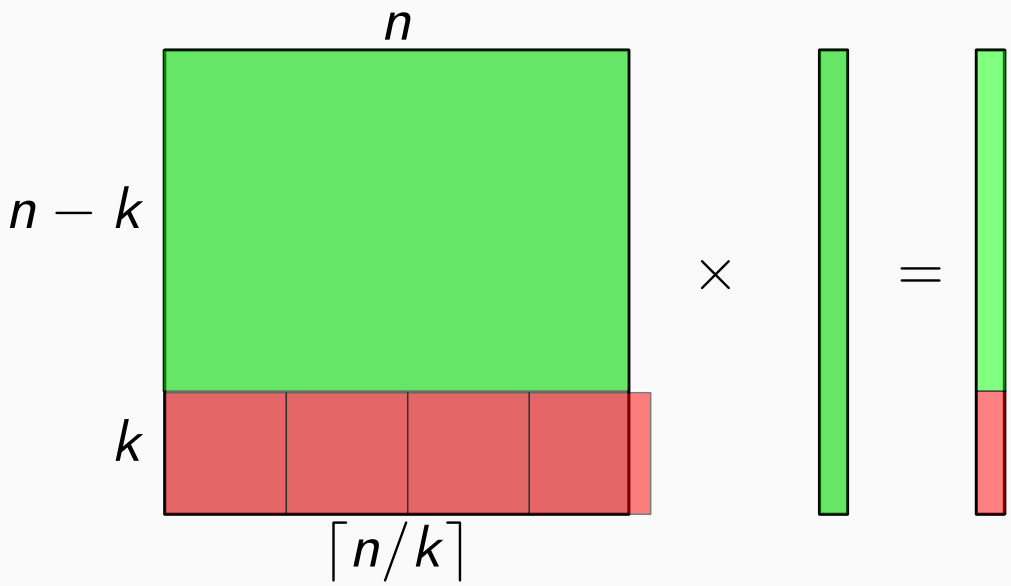


In-place middle product



- Recursive call:
  - the size of f decreases...
  - but **not the size of g!**
- $T(n, m) = \lceil n/k \rceil \mathrm{M}(k) + (\lceil n/k \rceil - 1)k + T(n, m - k)$

In-place middle product



- Recursive call:
  - the size of f decreases...
  - but **not the size of g!**
- $T(n,m) = \lceil n/k \rceil \mathrm{M}(k) + (\lceil n/k \rceil - 1)k + T(n, m-k)$

$$T(n,n) \leq \begin{cases} \mathrm{M}(n) \log_{\frac{c+2}{c+1}}(n) + o(\mathrm{M}(n) \log n) & \text{if } \mathrm{M}(n) \text{ is quasi-linear} \\ O(\mathrm{M}(n)) & \text{otherwise} \end{cases}$$

Conclusion

- Space-preserving reductions between polynomial products
- Self-reductions to obtain in-place algorithms

Conclusion

- Space-preserving reductions between polynomial products
- Self-reductions to obtain in-place algorithms

Comparisons

- Better use specialized in-place algorithms. . .
- . . . when they exist!

Conclusion

- Space-preserving reductions between polynomial products
- Self-reductions to obtain in-place algorithms

Comparisons

- Better use specialized in-place algorithms...
- ... when they exist!

Main open problems

- Other polynomial operations $\rightarrow$ work in progress!
- Extra $\log(n)$ for the middle product
  - Remove it or prove a lower bound
  - General result on the transposition principle
- In-place integer arithmetic?

Conclusion

- Space-preserving reductions between polynomial products
- Self-reductions to obtain in-place algorithms

Comparisons

- Better use specialized in-place algorithms...
- ... when they exist!

Main open problems

- Other polynomial operations $\rightarrow$ work in progress!
- Extra $\log(n)$ for the middle product
  - Remove it or prove a lower bound
  - General result on the transposition principle
- In-place integer arithmetic?

Thank you!