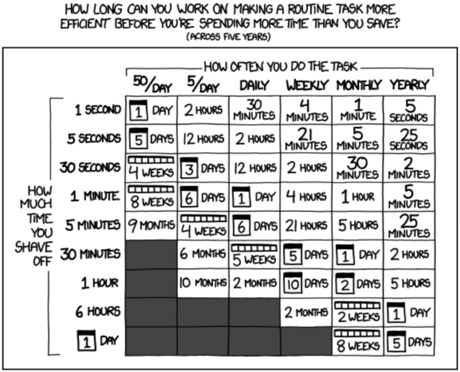
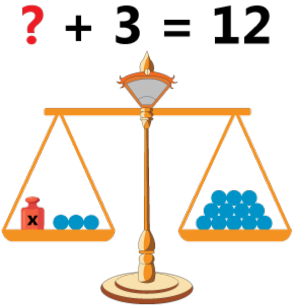
A primer on computer algebra

Or: Faster than expected

~~Accept~~Change what you cannot ~~change~~accept



Computer algebra



- Equations are everywhere: differential equations, linear or polynomial equations or inequalities, recurrences, equations in groups, algebras or categories, tensor equations etc.
- There are two ways of solving such equations: approximately or exactly
- Oversimplified, numerical analysis studies efficient ways to get approximate solutions; computer algebra wants exact solutions

To get started, an example from chemistry

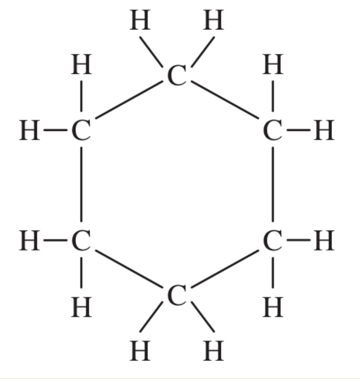
Watch out for three (very typical but usually highly nontrivial) steps:

- create a mathematical model
- “solve” the model (enter e.g. computer algebra)
- interpret the solution

We will see C_6H_{12} next, so here it is:

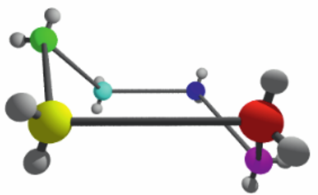
Based on a conjecture of **Sachse ∼1890**

and a solution by **Levelt ∼1997**

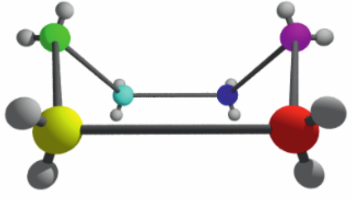


Computer algebra

chair:



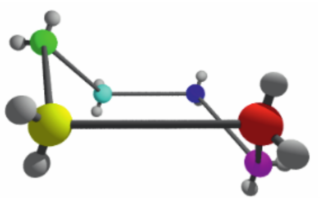
one boat:

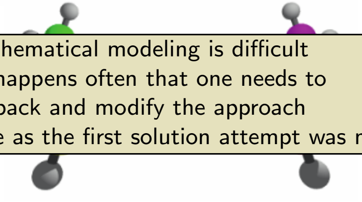


- C_6H_{12} occurs in incongruent conformations: chair (one) and boats (many) mod mirrors
- Chair occurs far more frequently than the boats
- Chair is stiff while the boats can twist into one another

Computer algebra

chair:

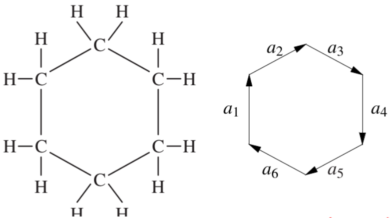




Mathematical modeling is difficult
so it happens often that one needs to
go back and modify the approach
This happened here as the first solution attempt was not helpful

- C_6H_{12} occurs in incongruent conformations: chair (one) and boats (many) mod mirrors
- Chair occurs far more frequently than the boats
- Chair is stiff while the boats can twist into one another

Computer algebra



$$a_1 \star a_1 = a_2 \star a_2 = \cdots = a_6 \star a_6 = 1,$$ Length between bonds

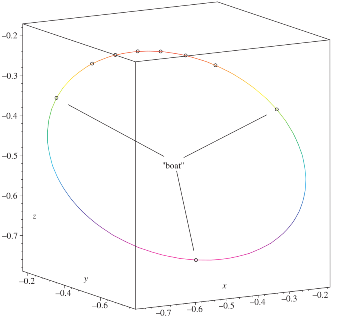
$$a_1 \star a_2 = a_2 \star a_3 = \cdots = a_6 \star a_1 = \frac{1}{3},$$ Angle ≈ 109°

$$a_1 + a_2 + \cdots + a_6 = 0.$$ Cyclic

- They then modeled the bods as vectors a_i and $a_i \star a_j$=inner product
- Model $S_{ij} = a_i \star a_j$ as variables
- One gets polynomial variables subject to the relations above ⇒ get solution via Gröbner bases

Comp

One gets that the inflexible solution chair is an isolated point while the boats lie on a curve:

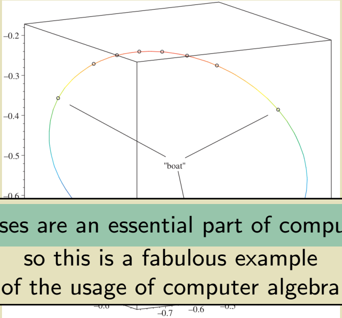


Chair won't move
and boats can be twisted
when build from tubes

- Th
- M
- O ... ution
vi

Comp

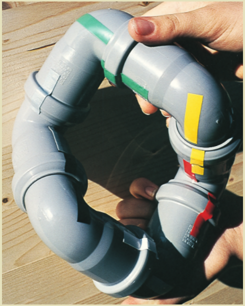
One gets that the inflexible solution chair is an isolated point while the boats lie on a curve:



Gröbner bases are an essential part of computer algebra

so this is a fabulous example
of the usage of computer algebra

Chair won't move
and boats can be twisted
when build from tubes

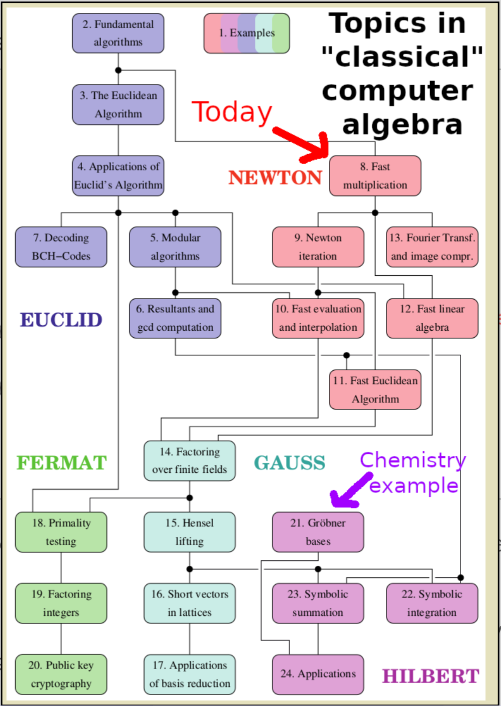


▶ Tl

▶ M

▶ O ... ution
vi

Computer algebr



$a_1 \star a$ ... een

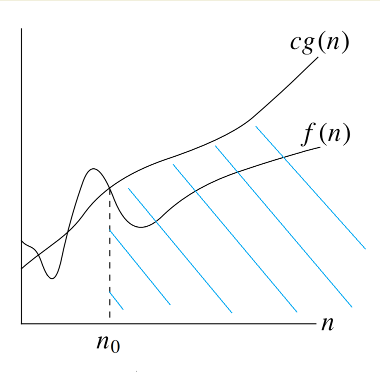
$a_1 \star a$

- They then mo ... er product
- Model $S_{ij} = a$
- One gets poly ... e ⇒ get solution via Gröbner ba

Computer algebra

What we are using throughout is worst-case-analysis using:

$f \in O(g)$:



Careful This is different from:

- Average-case-analysis
- Computational implications due to overhead ($\approx$ the part before n_0)

- The
- Mo
- One … olution via Gröbner bases

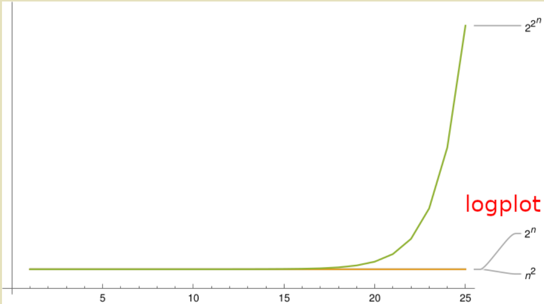
Compute

A Gröbner basis of an ideal $I \subset R[X_1, ..., X_n]$ and a monomial order is a set $G \subset I$ such that $\langle lt(G)\rangle = \langle lt(I)\rangle$; lt=leading term

Theorem (Buchberger ∼1965) Gröbner bases exist
can be computed algorithmically
and can be used to solve:

- ideal membership
- ideal containment
- properties of $V(I)$, e.g. $V(I) = \emptyset \Leftrightarrow G = \{1\}$

Problem Gröbner $\in O(\text{poly in } d^{2^n})$ for d=largest degree

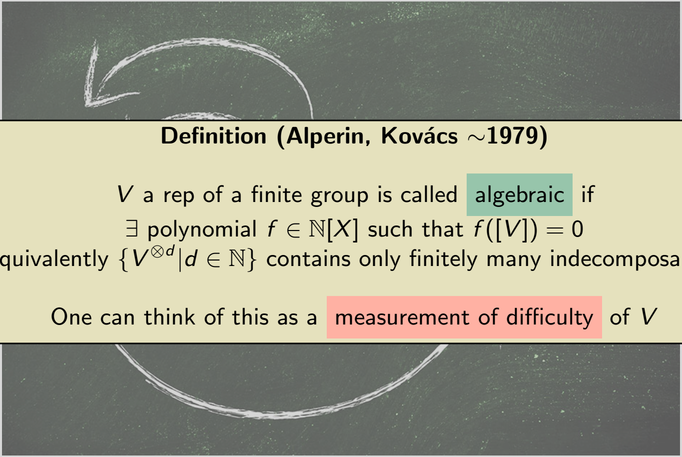


- They ... t
- Mod
- One ... solution via G

Computer algebra

- Now two more examples from representation theory that I recently learned
- Watch out for success and failure of experimenting with computer algebra

Computer algebra



Definition (Alperin, Kovács ∼1979)

V a rep of a finite group is called algebraic if
$\exists$ polynomial $f \in \mathbb{N}[X]$ such that $f([V]) = 0$
equivalently $\{V^{\otimes d} | d \in \mathbb{N}\}$ contains only finitely many indecomposables

One can think of this as a measurement of difficulty of V

- Now two more examples from representation theory that I recently learned
- Watch out for success and failure of experimenting with computer algebra

Compute

Success **A computer can check that for you (Craven ∼2015)**

V any simple of $G = \mathrm{SL}_2(\mathbb{F}_{p^k})$ over characteristic p is algebraic

e.g. $p = 5$, $\mathbb{K} = \mathbb{F}_5$, $k = 2$, $V = (\mathbb{F}_{25})^2$:

simples in $\mathscr{R}\mathrm{ep}(G, \mathbb{K})$:

```
[
    GModule of dimension 1 over GF(5),
    GModule of dimension 4 over GF(5),
    GModule of dimension 4 over GF(5),
    GModule of dimension 6 over GF(5),
    GModule of dimension 8 over GF(5),
    GModule of dimension 9 over GF(5),
    GModule of dimension 10 over GF(5),
    GModule of dimension 12 over GF(5),
    GModule of dimension 16 over GF(5),
    GModule of dimension 16 over GF(5),
    GModule of dimension 20 over GF(5),
    GModule of dimension 24 over GF(5),
    GModule of dimension 25 over GF(5),
    GModule of dimension 30 over GF(5),
    GModule of dimension 40 over GF(5)
]
```

indecomposables in $\mathscr{R}\mathrm{ep}(G, \mathbb{K})$:

```
G:=SpecialLinearGroup(2,5^2);
IsCyclic(SylowSubgroup(G,5));
false
```

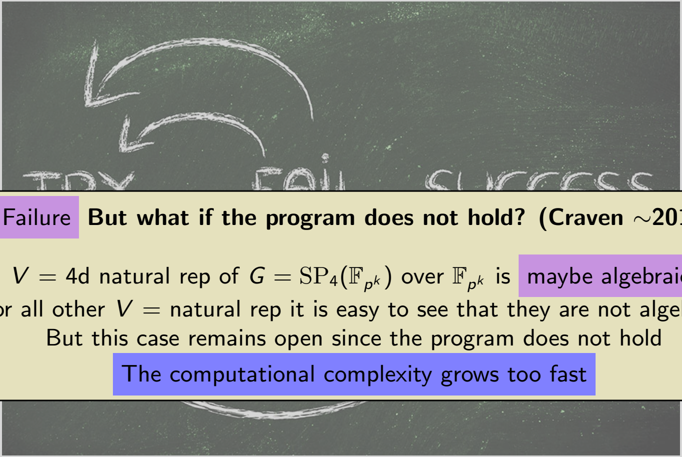
indecomposables in $\{V^{\otimes d} | d \in \mathbb{N}\}$:

```
[
    GModule of dimension 1 over GF(5),
    GModule M of dimension 4 over GF(5),
    GModule of dimension 4 over GF(5),
    GModule of dimension 6 over GF(5),
    GModule of dimension 12 over GF(5),
    GModule of dimension 8 over GF(5),
    GModule of dimension 9 over GF(5),
    GModule of dimension 16 over GF(5),
    GModule of dimension 10 over GF(5),
    GModule of dimension 24 over GF(5),
    GModule of dimension 20 over GF(5),
    GModule of dimension 20 over GF(5),
    GModule of dimension 16 over GF(5),
    GModule of dimension 30 over GF(5),
    GModule of dimension 40 over GF(5),
    GModule of dimension 20 over GF(5),
    GModule of dimension 40 over GF(5),
    GModule of dimension 60 over GF(5),
```

+a few more (45 in total)

- Now t … learned
- Watc … algebra

Computer algebra



Failure **But what if the program does not hold? (Craven ∼2015)**

V = 4d natural rep of $G = \mathrm{SP}_4(\mathbb{F}_{p^k})$ over $\mathbb{F}_{p^k}$ is maybe algebraic

For all other V = natural rep it is easy to see that they are not algebraic

But this case remains open since the program does not hold

The computational complexity grows too fast

- Now two more examples from representation theory that I recently learned
- Watch out for success and failure of experimenting with computer algebra

char table of M_{11} :

		1	2	3	4	5	6	7	8	9	10
Class |		1	2	3	4	5	6	7	8	9	10
Size |		1	165	440	990	1584	1320	990	990	720	720
Order |		1	2	3	4	5	6	8	8	11	11
p = 2		1	1	3	2	5	3	4	4	10	9
p = 3		1	2	1	4	5	2	7	8	9	10
p = 5		1	2	3	4	1	6	8	7	9	10
p = 11		1	2	3	4	5	6	7	8	1	1
X.1	+	1	1	1	1	1	1	1	1	1	1
X.2	+	10	2	1	2	0	-1	0	0	-1	-1
X.3	0	10	-2	1	0	0	1	Z1	-Z1	-1	-1
X.4	0	10	-2	1	0	0	1	-Z1	Z1	-1	-1
X.5	+	11	3	2	-1	1	0	-1	-1	0	0
X.6	0	16	0	-2	0	1	0	0	0	Z2	Z2#2
X.7	0	16	0	-2	0	1	0	0	0	Z2#2	Z2
X.8	+	44	4	-1	0	-1	1	0	0	0	0
X.9	+	45	-3	0	1	0	0	-1	-1	1	1
X.10	+	55	-1	1	-1	0	-1	1	1	0	0

- We now discuss finite groups G with fd reps over $\mathbb{C}$
- **Burnside ∼1911** Every >1d simple character has zeros
- Question Determine where the zeros are

- No… …rned
- Watch out for success and failure of experimenting with computer algebra

Computer algebra

char table of S_4 :

```
-------------------------
Class |   1  2  3  4  5
Size  |   1  3  6  8  6
Order |   1  2  2  3  4
-------------------------
p  =  2   1  1  1  4  2
p  =  3   1  2  3  1  5
-------------------------
X.1    +  1  1  1  1  1
X.2    +  1  1 -1  1 -1
X.3    +  2  2  0 -1  0
X.4    +  3 -1 -1  0  1
X.5    +  3 -1  1  0 -1
```

$P(\chi(g)=0)=24/120\approx 0.194,\quad P(\chi(C)=0)=4/25=0.16$

- Problem Determine for which $g\in G$ we have $\chi(g)=0$ Too hard!
- Better(?) problem $P(\chi(g)=0)$ or $P(\chi(C)=0)$ (probability) for randomly chosen $g\in G$ or conjugacy class C
- Watch out for success and failure of experimenting with computer algebra

char table of S_7:

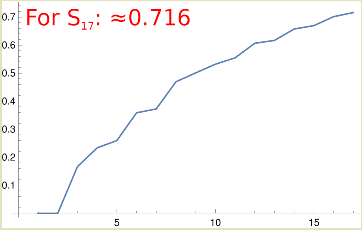
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Class		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Size		1	21	105	105	70	280	210	630	504	210	420	840	720	504	420
Order		1	2	2	2	3	3	4	4	5	6	6	6	7	10	12
p = 2		1	1	1	1	5	6	4	4	9	5	5	6	13	9	10
p = 3		1	2	3	4	1	1	7	8	9	4	2	3	13	14	7
p = 5		1	2	3	4	5	6	7	8	1	10	11	12	13	2	15
p = 7		1	2	3	4	5	6	7	8	9	10	11	12	1	14	15
X.1	+	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
X.2	+	1	-1	-1	1	1	1	-1	1	1	1	-1	-1	1	-1	-1
X.3	+	6	-4	0	2	3	0	-2	0	1	-1	-1	0	-1	1	1
X.4	+	6	4	0	2	3	0	2	0	1	-1	1	0	-1	-1	-1
X.5	+	14	6	2	2	2	-1	0	0	-1	2	0	-1	0	1	0
X.6	+	14	-6	-2	2	2	-1	0	0	-1	2	0	1	0	-1	0
X.7	+	14	-4	0	2	-1	2	2	0	-1	-1	-1	0	0	1	-1
X.8	+	14	4	0	2	-1	2	-2	0	-1	-1	1	0	0	-1	1
X.9	+	15	5	-3	-1	3	0	1	-1	0	-1	-1	0	1	0	1
X.10	+	15	-5	3	-1	3	0	-1	-1	0	-1	1	0	1	0	-1
X.11	+	20	0	0	-4	2	2	0	0	0	2	0	0	-1	0	0
X.12	+	21	1	-3	1	-3	0	-1	-1	1	1	1	0	0	1	-1
X.13	+	21	-1	3	1	-3	0	1	-1	1	1	-1	0	0	-1	1
X.14	+	35	-5	-1	-1	-1	-1	1	1	0	-1	1	-1	0	0	1
X.15	+	35	5	1	-1	-1	-1	-1	1	0	-1	-1	1	0	0	-1

$P(\chi(g)=0) = 28146/75600 \approx 0.372, \quad P(\chi(C)=0) = 55/225 \approx 0.24$

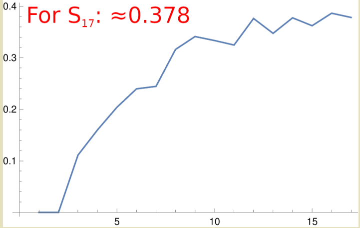
- Now two more examples from representation theory that I recently learned
- Watch out for success and failure of experimenting with computer algebra

Computer algebra

Here is $P(\chi(g)=0)$:



Here is $P(\chi(C)=0)$:



My silly 10-min-code only made it to S_{17}, pathetic, sorry for that!

Alexander Miller computed these up to S_{38}

Anyway, we can guess from here for $P(\chi(g)=0)$

but the data is not good enough for $P(\chi(C)=0)$

- Now … earned
- Watc… algebra

Fast multiplication

(x - 3)(4x – 5)

	x	-3
4x	$4x^2$	-12x
-5	-5x	15

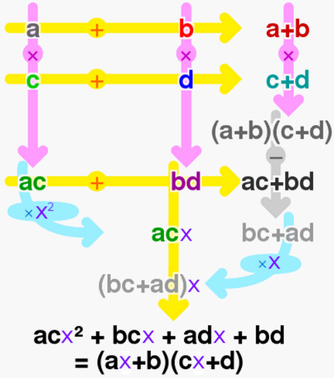
$4x^2$ - 12x – 5x + 15

$4x^2$ - 17x + 15

	x^2	$-4x$	-2
$2x^2$	$2x^4$	$-8x^3$	$-4x^2$
$-x$	$-x^3$	$4x^2$	$2x$
-1	$-x^2$	$4x$	2

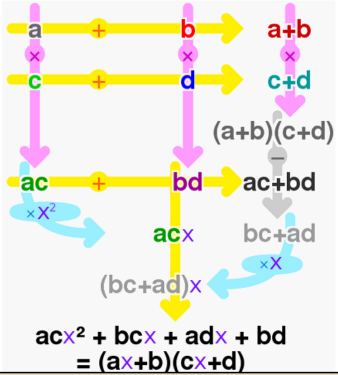
- Given two polynomials f and g of degree $< n$; we want fg
- Classical polynomial multiplication needs n^2 multiplications and $(n-1)^2$ additions; thus $mult(poly) \in O(n^2)$
- It doesn't appear that we can do faster

Fast multiplication



- **Karatsuba ∼1960** It gets faster!
- Reduce multiplication cost even when potentially increasing addition cost
- Second, apply divide-and-conquer

Fast multiplication



We compute ac, bd, $u = (a+b)(c+d)$, $v = ac + bd$, $u - v$
with 3 multiplications and 4 additions = 7 operations
The total has increased, but a recursive application will drastically reduce the overall cost
Upshot We only have 3 multiplications not 4

- Reduce multiplication cost even when potentially increasing addition cost
- Second, apply divide-and-conquer

Fast multiplication

ALGORITHM 8.1 Karatsuba's polynomial multiplication algorithm.

Input: $f, g \in R[x]$ of degrees less than n, where R is a ring (commutative, with 1) and n a power of 2.

Output: $fg \in R[x]$.

1. **if** $n = 1$ **then return** $f \cdot g \in R$
2. let $f = F_1 x^{n/2} + F_0$ and $g = G_1 x^{n/2} + G_0$, with $F_0, F_1, G_0, G_1 \in R[x]$ of degrees less than $n/2$
3. compute $F_0 G_0$, $F_1 G_1$, and $(F_0 + F_1)(G_0 + G_1)$ by a recursive call
4. **return** $F_1 G_1 x^n + ((F_0 + F_1)(G_0 + G_1) - F_0 G_0 - F_1 G_1) x^{n/2} + F_0 G_0$

Example

$f = g = x^3 + x^2 + x + 1$ is equal to $F_1 + F_0 = (x + 1)x^2 + x + 1$

$F_0^2 = F_1^2 = (x + 1)^2$ and $(2x + 2)(2x + 2)$ need 7 ops = 21 ops

To get fg we then need two more ops = 23 ops

Classical we need $4^2 + (4 - 1)^2$ = 25 ops

Fast multiplicat

ALGORITH ... m.

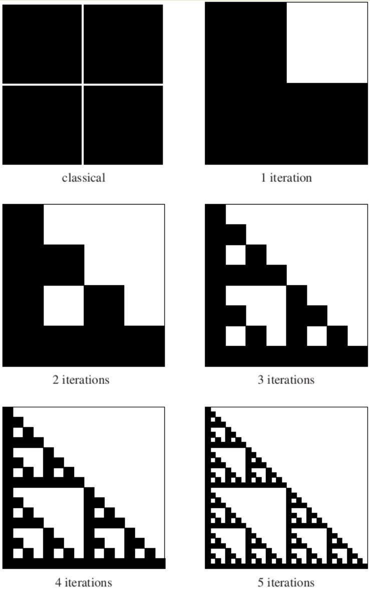
Input: $f, g \in R[$ ... ative, with 1)

and n a pow

Output: $fg \in R$

1. **if** $n = 1$ t
2. let $f = F_1$ ... $[x]$ of degrees less than
3. compute ... ll
4. **return** F_1 ... G_0

This applies recursively, so we actually save a lot:



8.2: Cost (= black area) of Karatsuba's algorithm for increasing recursion depths.
ge approaches a fractal of dimension $\log_2 3 \approx 1.59$.

Example

$f = g = x^3 + x^2$

$F_0^2 = F_1^2 = (x +$

To get fg we the

Classical we need

Fast multiplication

ALGORITHM 8.1 Karatsuba's polynomial multiplication algorithm.

Input: $f, g \in R[x]$ of degrees less than n, where R is a ring (commutative, with 1) and n a power of 2.

Output: $fg \in R[x]$.

1. **if** $n = 1$ **then return** $f \cdot g \in R$

Theorem (Karatsuba ∼1960)

For $n = 2^k$ we have $mult(poly) \in O(n^{1.59})$ $(1.59 \approx \log(3)$; always: $\log = \log_2)$

There is also a version for general n but the analysis is somewhat more involved

3. compute F_0G_0, F_1G_1, and $(F_0 + F_1)(G_0 + G_1)$ by a recursive call

4. **return** $F_1G_1x^n + ((F_0 + F_1)(G_0 + G_1) - F_0G_0 - F_1G_1)x^{n/2} + F_0G_0$

Example

$f = g = x^3 + x^2 + x + 1$ is equal to $F_1 + F_0 = (x+1)x^2 + x + 1$

$F_0^2 = F_1^2 = (x+1)^2$ and $(2x+2)(2x+2)$ need 7 ops = 21 ops

To get fg we then need two more ops = 23 ops

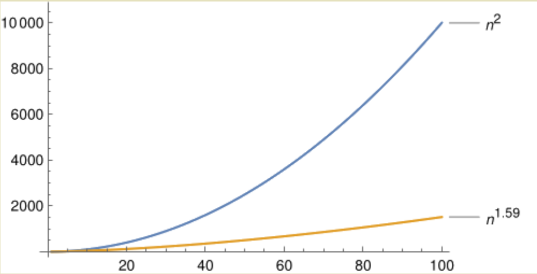
Classical we need $4^2 + (4-1)^2 = 25$ ops

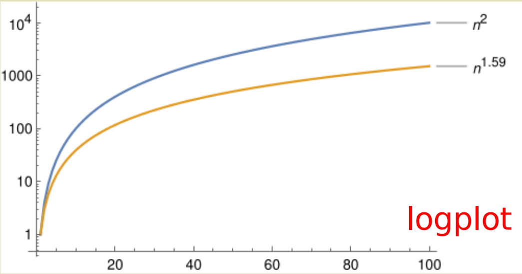
Fast multipl

Theorem (Karatsuba ∼1960)

For $n = 2^k$ we have $mult(poly) \in O(n^{1.59})$ $(1.59 \approx \log(3))$

This is much faster than before:





ALGORITH … m.

Input: $f, g \in R[$ … ative, with 1)

and n a pow …

Output: $fg \in R$ …

1. **if** $n = 1$ …
2. let $f = F$ … $x]$ of degrees less than …
3. compute … ll
4. **return** F … G_0

Example

$f = g = x^3 + x^2$ …

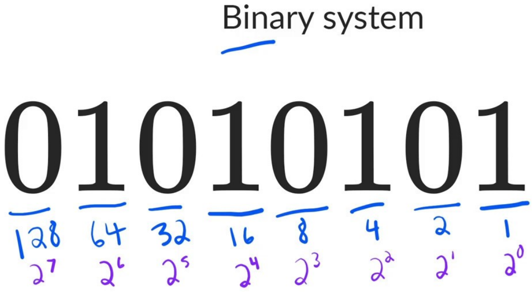
$F_0^2 = F_1^2 = (x +$ …

To get fg we the …

Classical we need …

Fast multiplication

$k = 2$:



Replace x^k by e.g. 2^k and do the same as before

- **Karatsuba** ∼**1960** Using k-adic expansion, this works for numbers as well
- **Theorem (Karatsuba** ∼**1960)** For $n = 2^k$ (n=#digits) we have $mult \in O(n^{1.59})$
- Multiplication is everywhere so this is fabulous

Fast multiplication

My silly 5 minute Python code:

```
from math import ceil, floor
#math.ceil(x) Return the ceiling of x as a float, the smallest integer value greater than or equal to x.
#math.floor(x) Return the floor of x as a float, the largest integer value less than or equal to x.

def karatsuba(x,y):
    #base case
    if x < 10 and y < 10: # in other words, if x and y are single digits
        return x*y

    n = max(len(str(x)), len(str(y)))
    m = ceil(n/2)   #Cast n into a float because n might lie outside the representable range of integers.

    x_H  = floor(x / 10**m)
    x_L = x % (10**m)

    y_H = floor(y / 10**m)
    y_L = y % (10**m)

    #recursive steps
    a = karatsuba(x_H,y_H)
    d = karatsuba(x_L,y_L)
    e = karatsuba(x_H + x_L, y_H + y_L) - a - d

    return int(a*(10**(m*2)) + e*(10**m) + d)

%time karatsuba(3141592653589793238462643383279502884197169399375105820974944592,
                2718281828459045235360287471352662497757247093699959574966967627)
```

▶ Theorem (Karatsuba ∼1960) For $n = 2^k$ (n=#digits) we have $mult \in O(n^{[illegible]})$

▶ Multiplication is everywhere so this is fabulous

My silly 5 minute code is still much slower than SageMath's:

Type some Sage code below and press Evaluate.

```
    y_H = floor(y / 10**m)
    y_L = y % (10**m)

    #recursive steps
    a = karatsuba(x_H,y_H)
    d = karatsuba(x_L,y_L)
    e = karatsuba(x_H + x_L, y_H + y_L) - a - d

    return int(a*(10**(m*2)) + e*(10**m) + d)

%time karatsuba(3141592653589793238462643383279502884197169399375105820974944592,2718281828459045235360287471352662497757247093699959574966967627)
```

Evaluate

```
CPU times: user 10.3 ms, sys: 412 µs, total: 10.7 ms
Wall time: 10.8 ms ←
8539734222673566957498846900491595793628487889746454950813687461572372213054499114931277629325900131223124341791952806582723184
```

Type some Sage code below and press Evaluate.

```
%time 3141592653589793238462643383279502884197169399375105820974944592*2718281828459045235360287471352662497757247093699959574966967627
```

Evaluate

```
CPU times: user 55 µs, sys: 22 µs, total: 77 µs
Wall time: 82.5 µs ←
8539734222673567065463550869546574495034888535765114961879601127067743044893204848617875072216249073013374895871952806582723184
```

Why is that? Well: (1) I am stupid $\Rightarrow$ too much overhead

(2) Nowadays computer algebra systems have beefed-up versions of Karatsuba's algorithm build in

Actually in use today:

Toom–Cook algorithm ∼1963 with $O(n^{1.46})$ ($1.46 \approx \log(5/3)$)

Schönhage–Strassen algorithm ∼1971 with $O(n \log n \log\log n)$

Toom–Cook generalizes Karatsuba; Schönhage–Strassen is based on FFT

Maybe in use soon (?):

Harvey–van der Hoeven algorithm ∼2019 with $O(n \log n)$

Annals of Mathematics **193** (2021), 563–617
https://doi.org/10.4007/annals.2021.193.2.4

Integer multiplication in time $O(n \log n)$

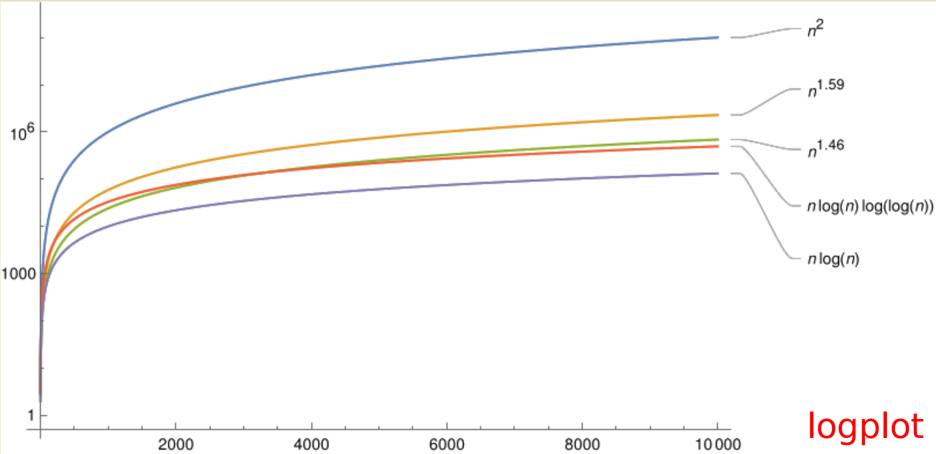
By David Harvey and Joris van der Hoeven

Conjecture (Schönhage–Strassen ∼1971) $O(n \log n)$ is the best possible

So maybe that's it!

- Multiplication is everywhere so this is fabulous

This is fantastic for large numbers:



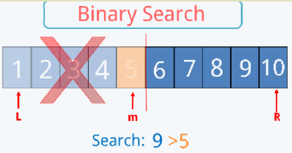
Do not try for small numbers due to overhead

- Multiplication is everywhere so this is fabulous

Fast multiplication

Honorable mentions These also run on divide-and-conquer

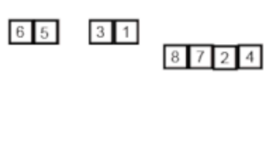
Binary search **(Babylonians ∼200BCE, Mauchly ∼1946, many others as well)** $\in O(\log n)$



Euclidean algorithm **(Euclid ∼300BCE)** $\in O\big(\log\min(a,b)\big)$

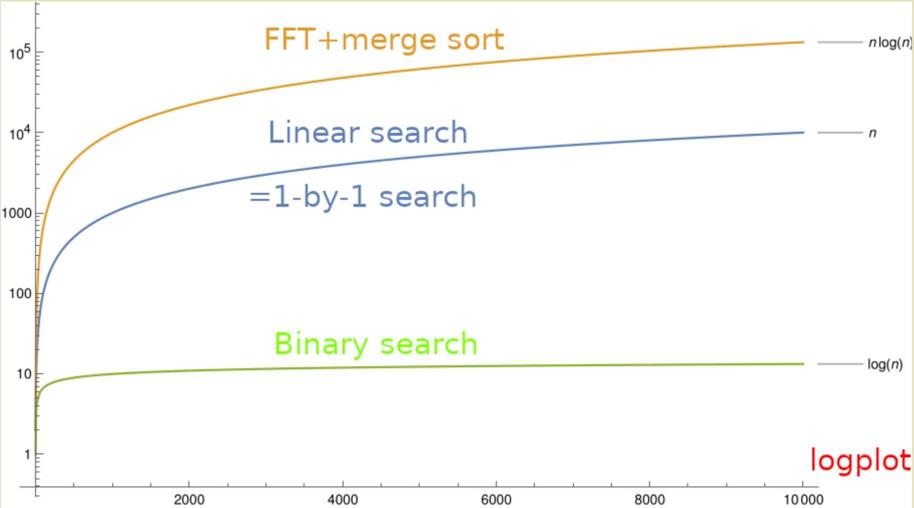
Fast Fourier transform **(Gauss ∼1805, Cooley–Tukey ∼1965)** $\in O(n\log n)$

Merge sort **(von Neumann ∼1945)** $\in O(n\log n)$



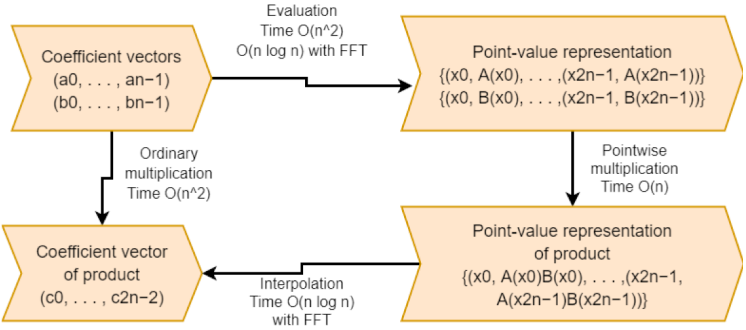
- Multiplication is everywhere so this is fabulous

For example, binary search does much better than linear search



- Multiplication is everywhere so this is fabulous

Discrete and fast Fourier transform



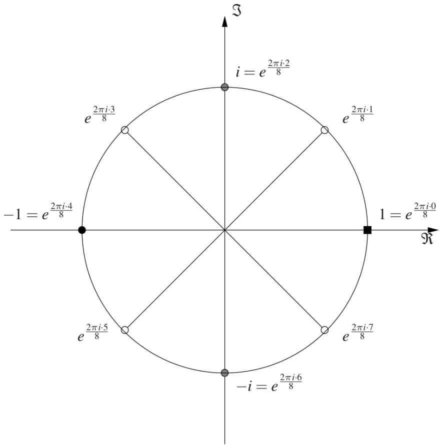
- Assume that there is an operation DFT_ω such that:

$$fg = DFT_\omega^{-1}\big(DFT_\omega(f)DFT_\omega(g)\big)$$

with DFT_ω and DFT_ω^{-1} and $DFT_\omega(f)DFT_\omega(g)$ being cheap

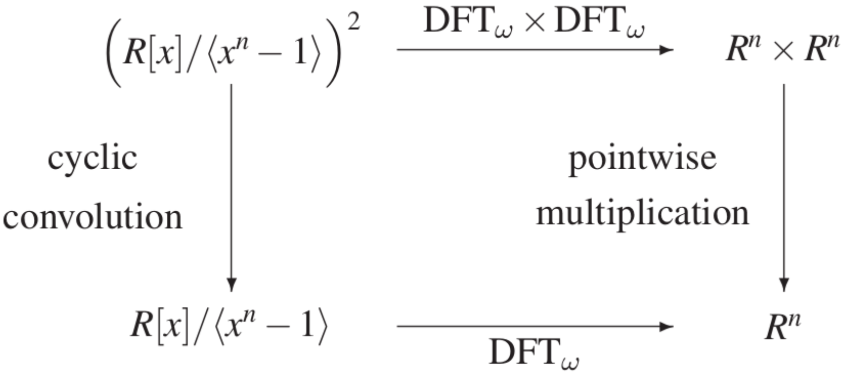
- Then compute fg for polynomials f and g is "cheap"

Discrete and fast Fourier transform



- In the following in need primitive roots of unity ω in some field R
- You can always assume $R = \mathbb{C}$ and $\omega = \exp(2\pi k/n)$

Discrete and fast Fourier transform



The R-linear map

$$DFT_\omega(f) = \big(1, f(\omega), f(\omega^2), ..., f(\omega^{n-1})\big)$$

that evaluates a polynomial at ω^i is called the **Discrete Fourier transform (DFT)**

Discrete and fast Fourier transform

Theorem (Fast Fourier transform (FFT) Cooley–Tukey ∼1965)

DFT_ω can computed in $O(n \log n)$

Theorem (FFT and Vandermonde ∼1770?)

DFT_ω^{-1} can computed in $O(n \log n)$

$\times R^n$

cyclic convolution

pointwise multiplication

$R[x]/\langle x^n - 1\rangle \xrightarrow{\mathrm{DFT}_\omega} R^n$

The R-linear map

$$DFT_\omega(f) = \big(1, f(\omega), f(\omega^2), \ldots, f(\omega^{n-1})\big)$$

that evaluates a polynomial at ω^i is called the Discrete Fourier transform (DFT)

Discrete and fast Fourier transform

Theorem (Fast Fourier transform (FFT) Cooley–Tukey ∼1965)

DFT_ω can computed in $O(n \log n)$

Theorem (FFT and Vandermonde ∼1770?)

DFT_ω^{-1} can computed in $O(n \log n)$

R^n

cyclic

pointwise

The Vandermonde matrix, matrix of the multipoint evaluation map DFT_ω,

$$V_\omega = \mathrm{VDM}(1, \omega, \dots, \omega^{n-1}) = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)^2} \end{pmatrix}$$

is easy to invert $V_\omega V_{\omega^{-1}} = nId$

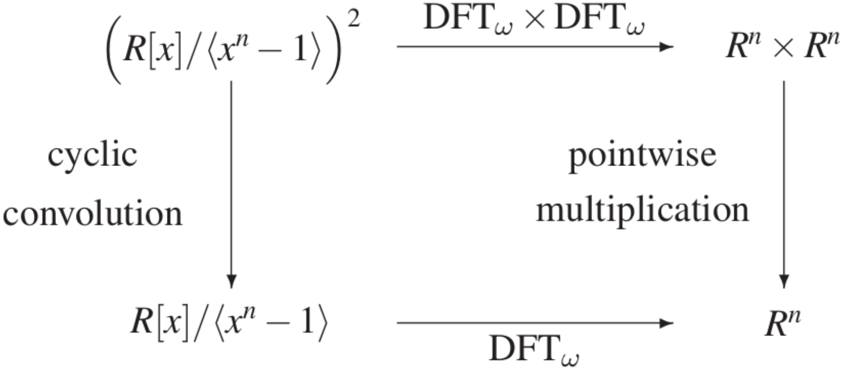
Th

$$V_i = \mathrm{VDM}(1, i, -1, -i) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

$$V_i^{-1} = \frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

that evaluates a polynomial at ω^i is called the Discrete Fourier transform (DFT)

Discrete and fast Fourier transform



Cyclic convolution of $f = f_{n-1}x^{n-1} + \ldots$ and $g = g_{n-1}x^{n-1} + \ldots$ is

$$h = f *_n g = \sum_{0 \le l < n} h_l x^l, \quad h_l = \sum_{j+k \equiv l \bmod n} f_j g_k$$

We see in a second why this is cyclic

Discrete an

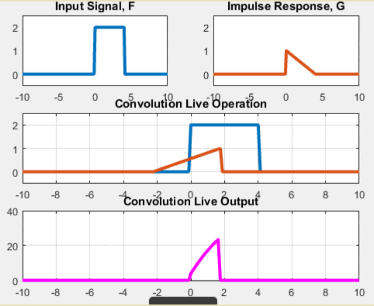
cycli

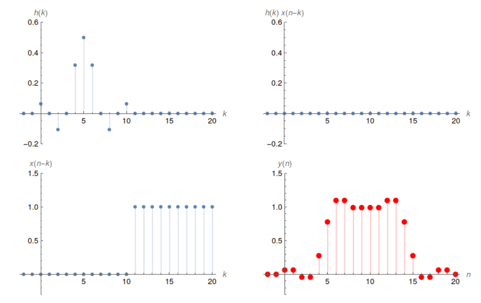
convolu

$\times R^n$

R^n

Slogan Convolution = area obtained by sliding f through g



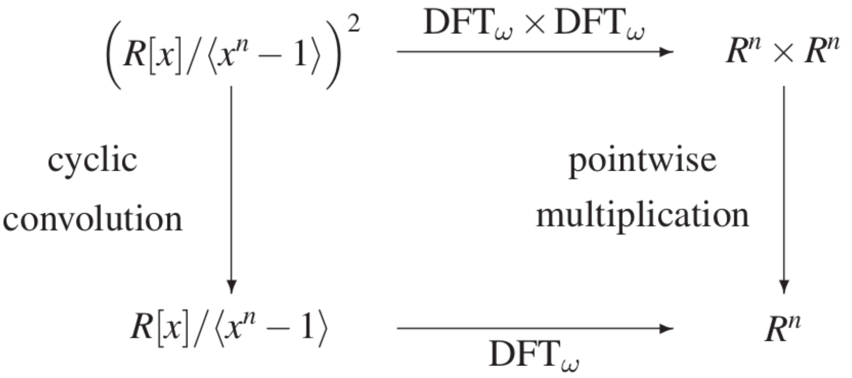


We have a cyclic version of this

Cyclic conv

We see in a

Discrete and fast Fourier transform

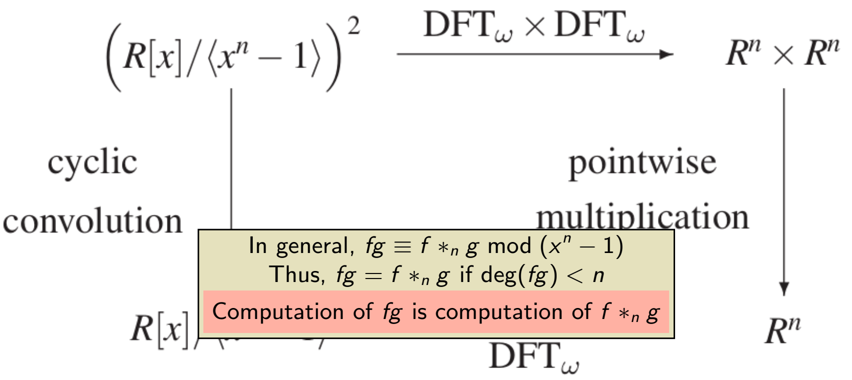


Example Take $f = x^3 + 1$ and $g = 2x^3 + 3x^2 + x + 1$

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

$$= (2x^2 + 3x + 1)(x^4 - 1) + 3x^3 + 5x^2 + 4x + 2 \equiv f *_4 g \bmod (x^4 - 1)$$

Discrete and fast Fourier transform



In general, $fg \equiv f *_n g \bmod (x^n - 1)$
Thus, $fg = f *_n g$ if $\deg(fg) < n$

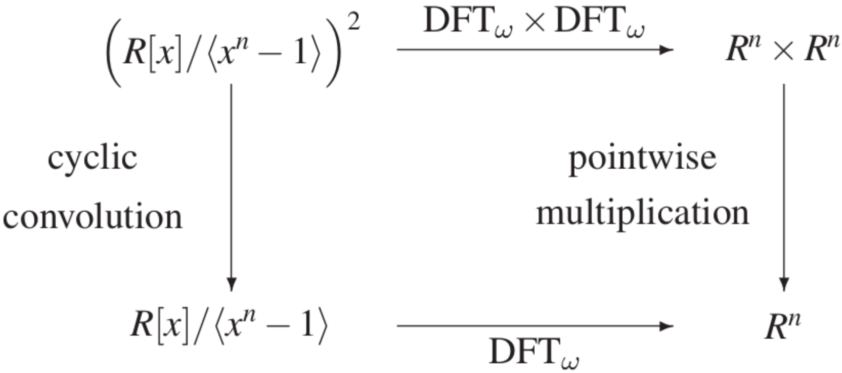
Computation of fg is computation of $f *_n g$

Example Take $f = x^3 + 1$ and $g = 2x^3 + 3x^2 + x + 1$

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

$$= (2x^2 + 3x + 1)(x^4 - 1) + 3x^3 + 5x^2 + 4x + 2 \equiv f *_4 g \bmod (x^4 - 1)$$

Discrete and fast Fourier transform



Final lemma we need

$$DFT_\omega(f *_n g) = DFT_\omega(f) \cdot_{\text{pointwise}} DFT_\omega(g)$$

$$\left(R[x]/\langle x^n - 1\rangle\right)^2 \xrightarrow{\mathrm{DFT}_\omega \times \mathrm{DFT}_\omega} R^n \times R^n$$

co

Theorem (Cooley–Tukey ∼1965)

Computing fg is in $O(n \log n)$ for $\deg(fg) > n$

"Proof"

Take n so that $\deg(fg) > n$

Then $fg = f *_n g$, so it remains to show that computing $f *_n g$ is in $O(n \log n)$

But $f *_n g = DFT_\omega^{-1}(DFT_\omega(f) \cdot_{\text{pointwise}} DFT_\omega(g))$

$DFT_\omega(f)$ and $DFT_\omega(f)^{-1}$ is in $O(n \log n)$

Final lemma we need

$$DFT_\omega(f *_n g) = DFT_\omega(f) \cdot_{\text{pointwise}} DFT_\omega(g)$$

Discrete and fas

$\Big(R[x$ $R^n \times R^n$

cyclic

convolution

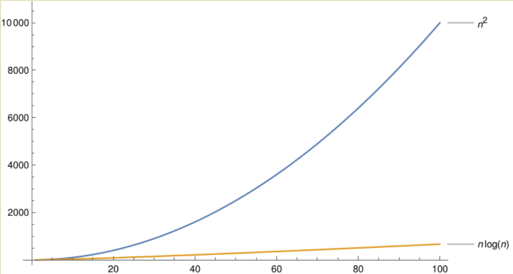
e

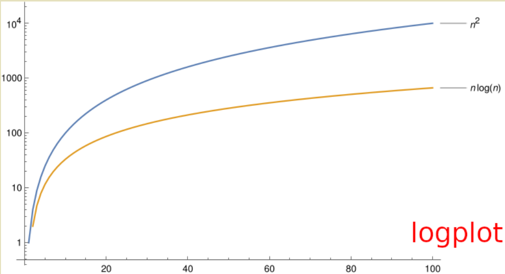
ion

$R[x$ R^n

Final lemma we

This is much faster than before:

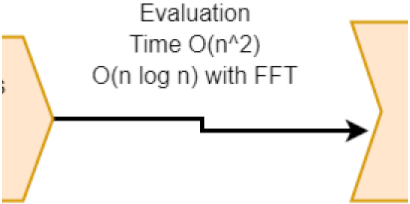




The overhead is however pretty large

Discrete and fast Fourier transform

What is FFT in this context?



- Assume $n = 2^k$ and note that, using Euclid's algorithm, writing

$$f = q_0(x^{n/2} - 1) + r_0 = q_1(x^{n/2} + 1) + r_1 \text{ gives}$$

$$f(\omega^{\text{even}}) = r_0(\omega^{\text{even}}), \quad f(\omega^{\text{odd}}) = r_1(\omega^{\text{odd}})$$

- Writing $r_1(\_)^* = r_1(\omega\_)$ we can use divide-and-conquer since ω^2 is a primitive $(n/2)$th root of unity:

$r_0(\omega^{\text{even}})$ and $r_1^*(\omega^{\text{even}})$ are DFTs of order $n/2$ $\Rightarrow$make recursive call

Discrete an

Theorem (Cooley–Tukey ∼1965) FFT runs in $O(n \log n)$

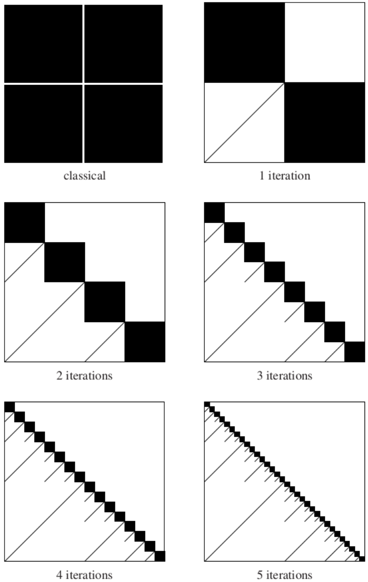
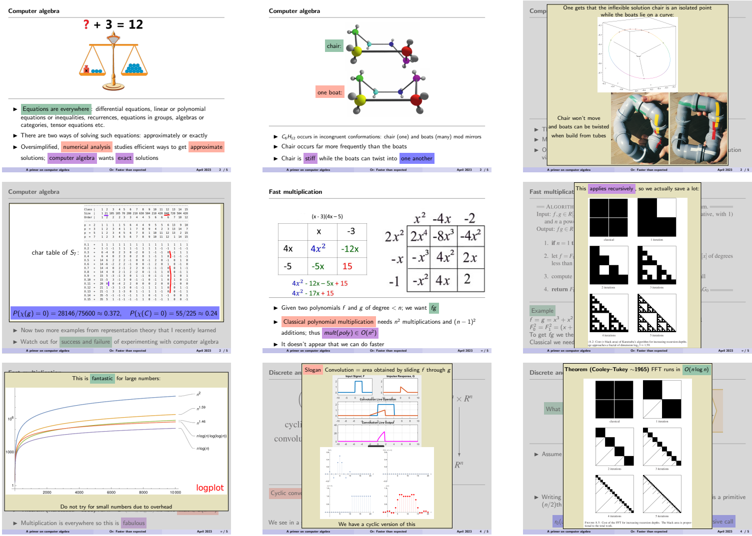


FIGURE 8.5: Cost of the FFT for increasing recursion depths. The black area is proportional to the total work.

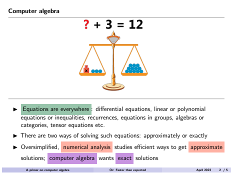
What

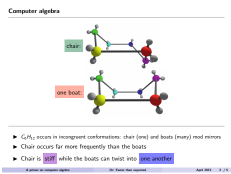
- Assume
- Writing … is a primitive $(n/2)$th

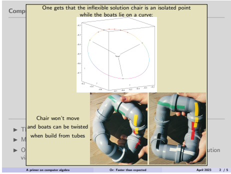
$r_0(\omega$ … sive call

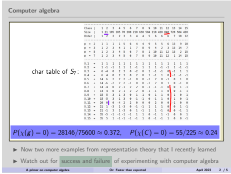


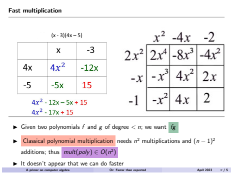
There is still much to do...

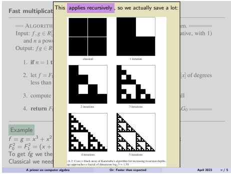


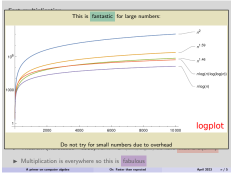


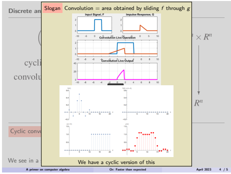


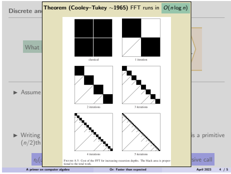












Thanks for your attention!