Let g be a group element, g cyclic order p
(such as a point on an elliptic curve).

we want to compute

$$ng$$

or really $[n]g$ (mod p) for some large n

we can do it like

$$\cancel{g+g+\cdots}\ ((P+P)+P)+P \cdots$$

in $n-1$ additions.

or we can use a loop, if we want $2^k P$,

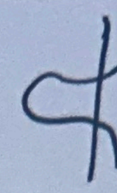
for := 1 to k
  set $P = 2P = P+P$

equivalently reusing intermediate results

$P_0 = P$
$P_1 = 2P_0$
$P_2 = 2P_1 = 4P$
$P_3 = 2P_2 = 8P$
⋮

"repeated doublings" or "repeated squarings"
in this area, the group notation changes from additive to multiplicative, sorry

That's pretty much it, but lets be more precise as we will generalize a bit.



DAA ②

First, we write out the binary expansion of n,

$$n = e_0 + e_1 \cdot 2 + e_2 \cdot 2^2 + \cdots + e_{\lambda-1} 2^{\lambda-1} \qquad \lambda - 1 = t$$

so $e_0, \ldots e_{\lambda-1}$ is the binary representation of n

Set our double accumulator $Q = P$, $R = \begin{cases} O & e_0 = 0 \\ P & e_0 = 1 \end{cases}$ ← output accumulator ← OP identity, basepoint, point at $(0,1,0)$

For $i = 1$ to t:

set $Q = [2]Q$

if $e_i = 1$, set $R = R + Q$

return R (equals nP)

"double and add" or "square and multiply"

to impl the $\mathbb{Z}$-action

For a multiplicative group (any group, abelian not even cyclic) G, $g \in G$,

if $n = \sum e_i 2^i$

compute $g^n = \prod (g^{2^i})^{e_i}$

in $\leq \log_2 n$ squarings

and $\leq \log_2 n$ multiplications

if n is random, expect $\frac{1}{2}$ ones & $\frac{1}{2}$ zeros in binary expansion, so

$\log_2 n$ doublings

$\frac{1}{2}\log_2 n$ additions.

can we reduce the average running time by using a larger base?

$$n = \varepsilon_0 + \varepsilon_1 \cdot 2 + \varepsilon_2 \cdot 2^2 + \cdots + \varepsilon_t 2^t \quad \varepsilon_i \in \{-1, 0, 1\}$$

Ex show every integer has a $\{\pm 1, 0\}$ ternary exp w/ no two consecutive nonzero coeff.

- ternary is sparser, so fewer additions
- negation is basically free in an ellptic curve (why?)

Ex is it used?

Pippenger (1980) combined some earlier results start w/ N=3 ①
operates on level of gp - nothing to do w/ elliptic curves.

$G = g_1^{e_1} g_2^{e_2} g_3^{e_3}$ G a group

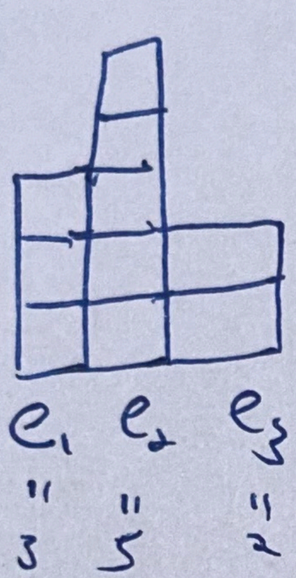
If $e_1, e_2, e_3 \in \{0,1\}$ a "multi-product." write $\|ell\|_1 = \sum_{i=1}^{N} e_i$

need $g_i^0 = 1$ as well, the group identity.

Q: what is id in an elliptic curve group

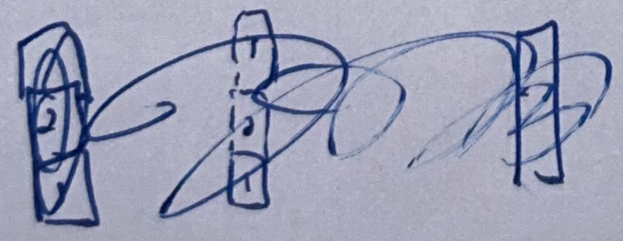
Naive is $q_1 = \|ell\|_1 - 1$
just mult w/o intermediate

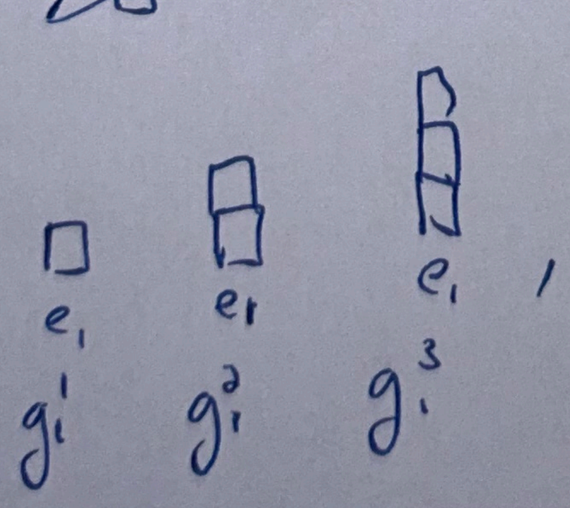
$g_1 \cdot g_1 \cdot g_2 \cdot g_2 \cdot g_2 \cdot g_3 \cdot g_3 \cdot g_3 \cdot g_3 \cdot g_3$

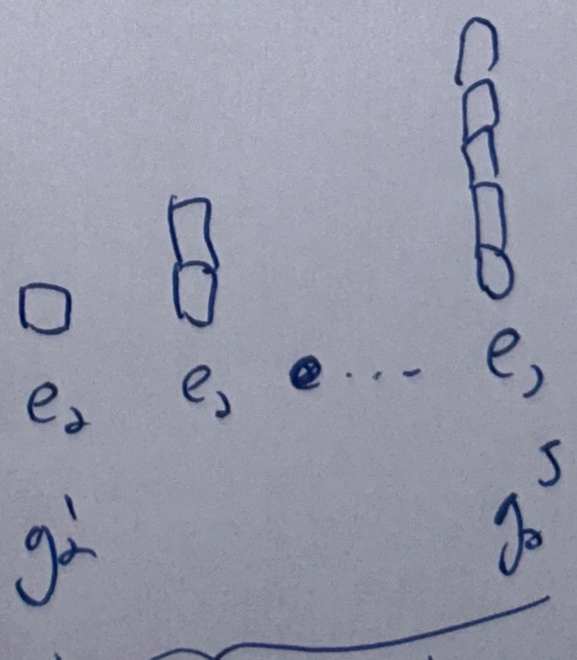


Model the cost as other multiplications $g \cdot h$ or exponentiations $g^2 = g \cdot g$ as same, grp op.

Some obvious optimizations: compute each power separately/ then combine:







can ignore by repeated squaring

$(g_2^2)^2 g_2$ is three multiplications

then multiply the $g_i^{e_i}$ together.

So, $x_1 = (g_1 \cdot g_1) \cdot g_1$ $\quad y_1 = (g_2 \cdot g_2) = g_2^2$ $\quad z_1 = g_3 \cdot g_3$

~~$x_2 = (g_2 \cdot g_2)$~~ $\quad y_2 = y_1 \cdot y_1 = g_2^4$

$y_3 = y_2 \cdot g_2 = g_2^5$

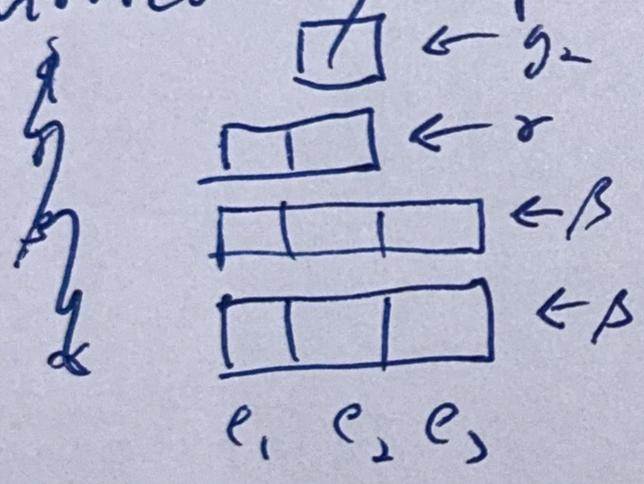
3. 3. 1.

Total = 7 gp ops + 2 to combine $= 9$

$(x_1 \cdot y_3) \cdot z_1$

Alternatively we could do "Lebesgue" integration, compute

horizontally first



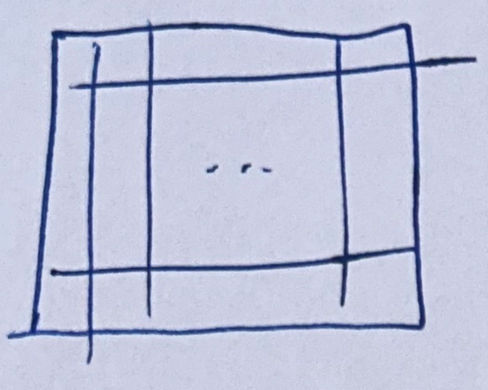
$\alpha = g_1 \cdot g_2$ 1 mult

$\beta = \alpha \cdot g_3 = g_1 \cdot g_2 \cdot g_3$ 1 mult.

combine G $= \beta \cdot \beta \cdot \alpha \cdot g_2$

total = 5 multiplications

Let's look at another example, then generalize and compute the asymptotic complexity.



$g^{2^3} = g^8 = \left(\left(g^2\right)^2\right)^2$

$\alpha = g \cdot g$

$\beta = \alpha \cdot \alpha$ 3 mul

$\gamma = \beta \cdot \beta$

basecase: raise to $n^k - 1$

$g_1^k g_2^k \cdots g_n^k = \underbrace{(g_1 \cdot g_2 \cdots g_n)^k}_{n-1}$

$(n-1)k$ multiplications.

Now we'll use some information about the group, Assume G is a cyclic group of order p, where p is a λ-bit prime, 254 bit so $\lambda = 256$.

$$G = \prod_{i=0}^{N-1} g_i^{e_i} \quad \text{msm}$$

we Assume $\lambda \geq N$ and decompose λ into s legs, scrap $\lambda = st$ s.t , say $s = 4$ legs and $t = 64$ of size t each $\sim / N = 16$. (turns out $s \approx \sqrt{\frac{\lambda}{N}}$ and $t \approx \sqrt{\lambda N}$ = optimal and convenient)

Let $e_i = \sum_{\ell=0}^{\lambda-1} e_{i,\ell} 2^\ell$ ($e_{i,\ell}$ is $e[i][\ell]$) so the $e_{i,\ell}$ are the binary digits of e, lowest-order to highest. little-endian

splitting λ into legs

$$e_i = \sum_{j=0}^{s-1} \sum_{k=0}^{t-1} 2^{j+sk} \underbrace{e_{i,j+sk}}_{0 \text{ or } 1}$$

Then $g_i^{e_i} = \prod_{\ell=0}^{\lambda-1} g_i^{2^\ell e_{i,\ell}} \underset{\lambda = st}{=} \prod_{j=0}^{s-1} \prod_{k=0}^{t-1} g_i^{2^{j+sk} \underbrace{e_{i,j+sk}}_{0 \text{ or } 1}}$

so $G = \prod_{i=0}^{N-1} g_i^{e_i} = \prod_{i=0}^{N-1} \left(\prod_{j=0}^{s-1} \prod_{k=0}^{t-1} g_i^{2^{j+sk} e_{i,j+sk}} \right)$

and rearranging the product,

$$G = \prod_{k=0}^{t-1} \left(\underbrace{\prod_{i=0}^{N-1} \prod_{j=0}^{s-1} g_i^{2^j e_{i,j+sk}}}_{:= G'_k} \right)^{2^{sk}}$$

↑ repeated squaring

So let's assume that repeated squaring, then multiplying these inner G'_k terms is the best we can do.

The above reduces the problem to computing the inner G'_k terms, from the g_i.

Or, incorporating the squaring, from the

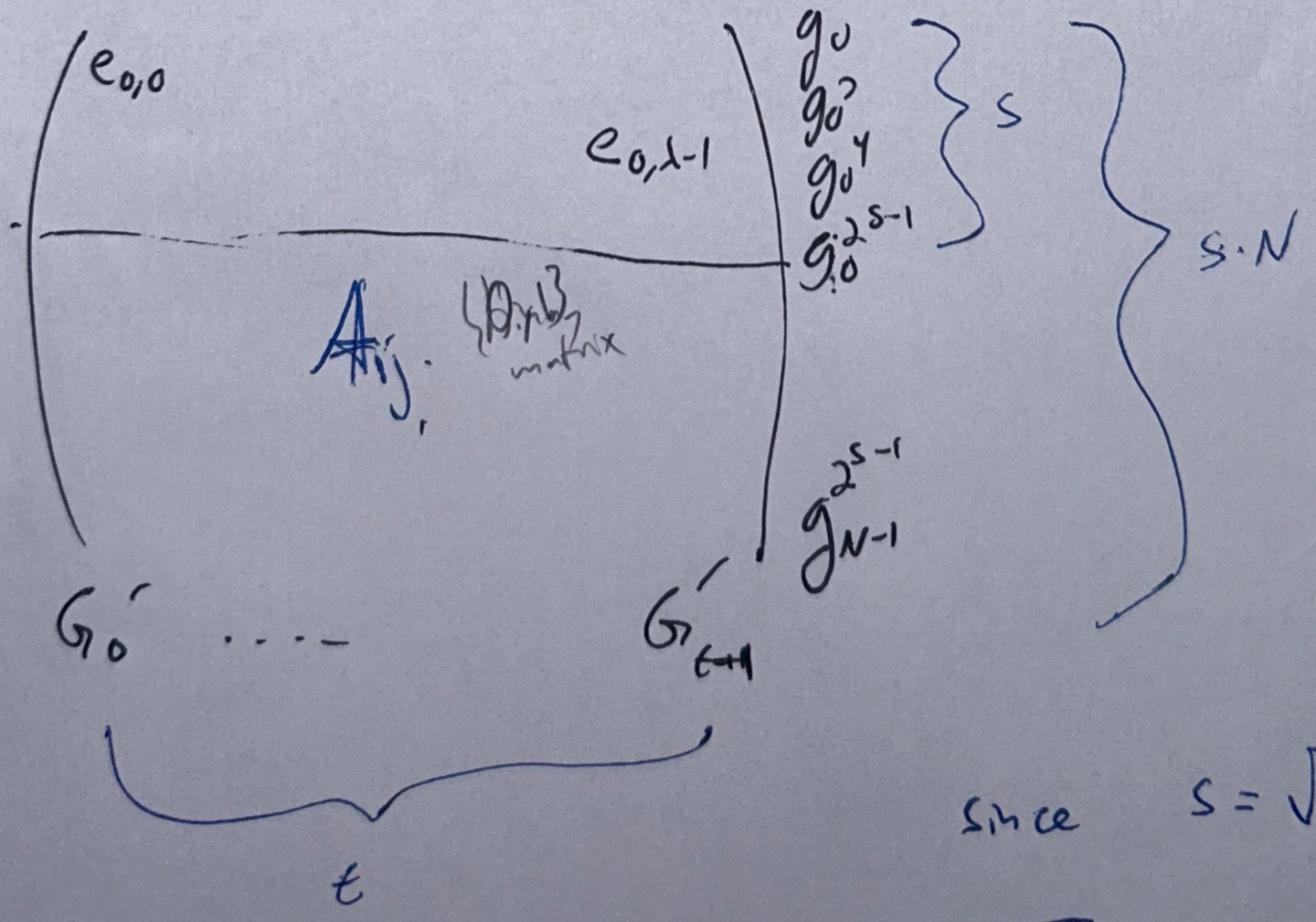
$$g_i, g_i^2, g_i^4, \ldots g_i^{2^j}, \ldots g_i^{2^{s-1}}$$

From such ↗ inputs, each inner term

$$G'_k = \prod_{i=0}^{N-1} \prod_{j=0}^{s-1} g_i^{2^j e_{i,j+sk}}$$

inputs. This costs $sN = \sqrt{\lambda N}$ gp ops since $s \cdot N = \sqrt{\frac{1}{N}} N = \sqrt{\lambda N}$

is a binary $\{0,1\}$ "linear combination" of the g^{2^j}



since $s = \sqrt{\frac{1}{N}}$, $t = \sqrt{\lambda N}$,

$t = sN =: M$

and this is a square

So now let's look at this ~~mess~~ simplified problem w/ a square $M \times M$ binary matrix.

(Pippinger's uses a D-ary matrix, non-square; indeterminates $x_1, \ldots x_k$ & and monomials $y_1 \ldots y_p$ in Rem of max degree D)

we ~~can~~ now forget ~~that~~ any structure of the $g_i^{2^j}$, we "used up" the structure of the $g_i^{2^j}$, by creating them via repeated squarings. Now we just treat them as M group elements $h_1, \ldots, h_M$ and we want to ~~compute~~

compute M monomials

$$H_j = \prod_{i=0}^{M-1} h_i^{a_{ij}} \leftarrow a_{ij}$$

each of which has a binary exponent vector.

This is a matrix-vector multiply w/ a binary matrix.

A simple strategy for evaluating this (obtaining $\{H_j\}$ from $\{g_i\}$ and A) is to pick a partition into sets $S_0, \dots S_{M/b-1}$ of at most b elements.

Then compute all the multiproducts in each set, one set T_i

So if $b=3$, $S_0 = \{h_0, h_1, h_2\}$,

$T_0 = \{g_0, g_1, g_2, g_0g_1, g_0g_2, g_1g_2, g_0g_1g_2\}$

Then each H_i only requires 1 elt from each set T_i.

Analysis: each S_i has b elts, 2^b gp ops to compute all possible multiproducts / contributions.

M/b sets, so This precompute is $\frac{2^b M}{b}$ gp ops.

Given the T_i, Each H_i needs at most one elt from each set, so $\frac{M}{b}$ gp ops

There are M of the H_i's, so $\frac{M^2}{b}$ given the T_i

Finally we recombine inputs

$$G = \prod_{k=0}^{t-1} G_k'^{2^{sk}}$$

using $st = \lambda$ squarings.

square G'_{t-1} s times,

multiply G'_{t-2}, square that s times, ...

$$M = \sqrt{\lambda N} \;\Rightarrow\; t = sN$$

$$\lambda + M + \frac{2^b M}{b} + \frac{M^2}{b}$$

- λ: squarings in $\prod_{k=0}^{t-1} G_k'^{2^{sk}}$
- M: multiplications in ←
- $\frac{2^b M}{b}$: precompute contributions T_i
- $\frac{M^2}{b}$: compute $H_k = G_k^{\,\cdot}$ given contributions T_i

$n/b = \log M - \log\log M,$

$$\lambda + M + \frac{M^2}{(\log M - \log\log M)(\log M)} + \frac{M^2}{\log M - \log\log M}$$

$$= \lambda + (1+o(1))\frac{M^2}{\log M}$$

$M = \sqrt{\lambda N}$ so cost is

$$\lambda + (1+o(1)) \frac{2\lambda N}{\log \lambda N}$$

why doesn't it depend on the degree bound?

2λ is the degree bound!

Maybe if actual exponents are $e \ll p$, there is a better way.

FFT / Möbius inversion / NTT / evaluation ↔ coeff form NTT (1)

Here are some related dualities

Time vs Freq domain

Möbius inversion

FFT/DFT/

NTT

DNF / CNF

evaluation form vs coeff form for polynomial.

—

3-SAT $(x_1 \vee x_7 \vee x_{10}) \wedge (x_2 \vee x_7 \vee x_{50}) \wedge \cdots$ CNF "factored"

number of clauses

"multiply out"

$(x_1 \wedge \neg x_2 \wedge \cdots) \vee (x_1 \wedge \neg x_2 \wedge \cdots) \vee \cdots$ DNF "expanded"

number of solutions

~~Some problems are easier~~

is a "sparse" evaluation form / truth table

$x_1 x_2 \cdots x_n$	f
0 0 0 ⋯ 0 0	0
0 0 0 ⋯ 0 1	0
	1 0
	1 1
	⋮

V of nonzero entries

Greg Hom

- "Just" changing the representation, from CNF to DNF, is hard, (#P)-hard

- Some things are easier in CNF, eg evaluating on a given input: just plug in the values, compact representation

- Some things are easier in DNF/evaluation form whether a solution exists, if a certificate exists, search problem. not very storage-efficient

The analogy here (can be made precise!) is that a polynomial can be given in several forms as well, including ~~extended form~~ expanded

① as coefficients $3 + 4x + 7x^2 + 5x^3 + \ldots$

or

$3x_1x_2x_{10} + 17x_3x_7x_{30} + \ldots$

or ② evaluated at points (need degree + 1 points for a univariate polynomial, # of monomials of degree $\leq n$ o/w w/ caveats

$\mathbb{F}[x_1, \dots, x_m]$

Char 0; $\rho = \binom{m+n}{n}$ monoms of degree $\le n$
w/ multiindices $I_1, \dots, I_\rho$, $|I_i| \le n$

<u>Thm</u> if $x_1, \dots, x_\rho \in \mathbb{F}^m$ distinct points in $\mathbb{F}^m$, TFAE

(1) given $y_1, \dots, y_\rho \in \mathbb{F}$
$\exists!$ $f \in \mathbb{F}[\vec{x}]$ of deg $\le n$
s.t. $f(x_i) = y_i$ $\forall$ $1 \le i \le \rho$

(2) Sample $\rho \times \rho$ matrix
$M = (x_i^{I_j})$ is invertible

Vandermonde w/ $m = 1$.

see refs

Factored form

(3)

- small representation, not always available
- few multiplications (tensor rank / sometimes matrix)
- rapid evaluation: can eval polys too big to write down
  - algebraic HASH FUNCTIONS

Conceptually DFT is

- logically evaluation form

  coeff → eval multiply → coeff

- "factored" linear transformation

  from coeff to eval

  form & back

Polynomial addition

In coefficient form:

$$\begin{array}{rl} & a_0 + a_1 x + a_2 x^2 + a_3 x^3 = f \\ + & b_0 + b_1 x + b_2 x^2 + b_3 x^3 = g \\ \hline & (a_0+b_0) + (a_1+b_1)x + (a_2+b_2)x^2 + (a_3+b_3)x^3 \end{array}$$

deg + 1 additions

In evaluation form

$(f+g)(1) = f(1) + g(1)$

$(f+g)(2) = f(2) + g(2)$

$(f+g)(3) = f(3) + g(3)$

$(f+g)(4) = f(4) + g(4)$

enough to determine $f+g$ in eval form

$\leftarrow$ ~~deg f + deg g~~ degree + 1

first do addition in coeff & eval form

NTT

Polynomial multiplication

- Zk-snark constructions use polynomial multiplications
- The polynomials involved are univariate and high degree.

Let's say we have coeff-form poly f, g & want coeff-form poly f·g

$$f(x) = a_0 + a_1 x + a_2 x^2 \qquad \deg f = 2$$

$$g(x) = b_0 + b_1 x + b_2 x^3 + b_3 x^3$$

naively we need $(\deg f + 1)(\deg g + 1)$ multiplications (field)

$$f \cdot g = (a_0 + a_1 x + a_2 x^2)(b_0 + b_1 x + b_2 x^2 + b_3 x^3)$$

$$= a_0 b_0 + (a_0 b_1 + a_1 b_0)x + (a_0 b_2 + a_1 b_1 + a_2 b_0)$$

$$(f \cdot g)_\ell = \sum_{j+k=\ell} f_j g_k = \sum_{j=0}^{\ell} a_j b_{\ell - j}$$

"$O(n^2)$"

But $(f \cdot g)(x) := f(x) \cdot g(x)$ (by definition above & fact)

so in evaluation form, we only need deg + 1 multiplications "~~nearly linear time~~" $O(n)$

So the idea is to multiply polynomials in coefficient form, change them to evaluation form, multiply, change back.

This is good if the changes are cheap, FFT makes it $O(n \log n)$

So the idea is that the
coefficient ring of the polynomials (here $\mathbb{F}_p$, a field)
should contain certain roots of unity. We
use, instead of evaluations $f(1), f(2), \ldots$,
evaluations $f(\omega^1), f(\omega^2), \ldots$, and
this makes interpolating f from its evaluations
efficient. Thus eval form → coeff form will be
efficient.

An n^{th} root of unity ω has $\omega^n = 1$ in $\mathbb{F}$.
It is primitive if (n is a unit in $\mathbb{F}$)
and $\omega^{n/q} - 1$ is not a zero divisor for any prime divisor q of n
(there are none if $\mathbb{F}$ a field)

$\mathbb{F}_q$ contains a primitive n^{th} root of unity ω if and only if
n divides $q-1$

Ex $x^{q-1}=1$ so every nonzero elt is a root of unity ($x^q = x$).

An $n \in \mathbb{Z}_{>0}$ is a primitive root of unity if $x^n = 1$ but $x^m \neq 1$ for any $m < n$.

If a is an n^{th} pr. o. u, $\mathbb{F}$ has all n roots of unity $1, a, a^2, \dots a^{n-1}$.

$\mathbb{F}_q$ contains n^{th} primitive root of unity iff $n \mid q-1$

if $n \mid q-1$, number of primitive n^{th} roots $= \varphi(n)$ Euler totient

n^{th} roots $= \gcd(n, q-1)$

So if $f = \sum_{j=0}^{n-1} f_j x^j$ is a polynomial of n-dim

coeff vector $f_0, \dots, f_{n-1}$,

we have the map "evaluate at powers of ω"

$DFT_\omega : \mathbb{F}_q^n \to \mathbb{F}_q^n$

$$f \longmapsto \left(f(1), f(\omega), f(\omega^2), \dots, f(\omega^{n-1}) \right)$$

this is the "discrete fourier transform" is an $\mathbb{F}_q$-linear map

to convolution

We have to adjust the notion of multiplication,

basically the same but degree is mod n

$$f = f_0 + f_1 x + \cdots + f_{n-1} x^{n-1}$$

$$g = g_0 + g_1 x + \cdots + g_{n-1} x^{n-1}$$

$$h = f *_n g = \sum_{\ell=0}^{n-1} h_\ell x^\ell$$

convolved

$0 \le \ell \le n$

$$h_\ell = \sum_{j+k \equiv \ell \bmod n} f_j g_k = \sum_{j=0}^{n-1} f_j g_{\ell - j}$$

new — mod n with $0 \le \ell \le n$

$*_n$ is cyclic convolution.

Equivalent to polynomial multiplication in $\mathbb{F}_q[x] / \langle x^n - 1 \rangle$

$$f *_n g \equiv fg \bmod x^n - 1$$

NTT (9)

Thm is ~~[illegible]~~

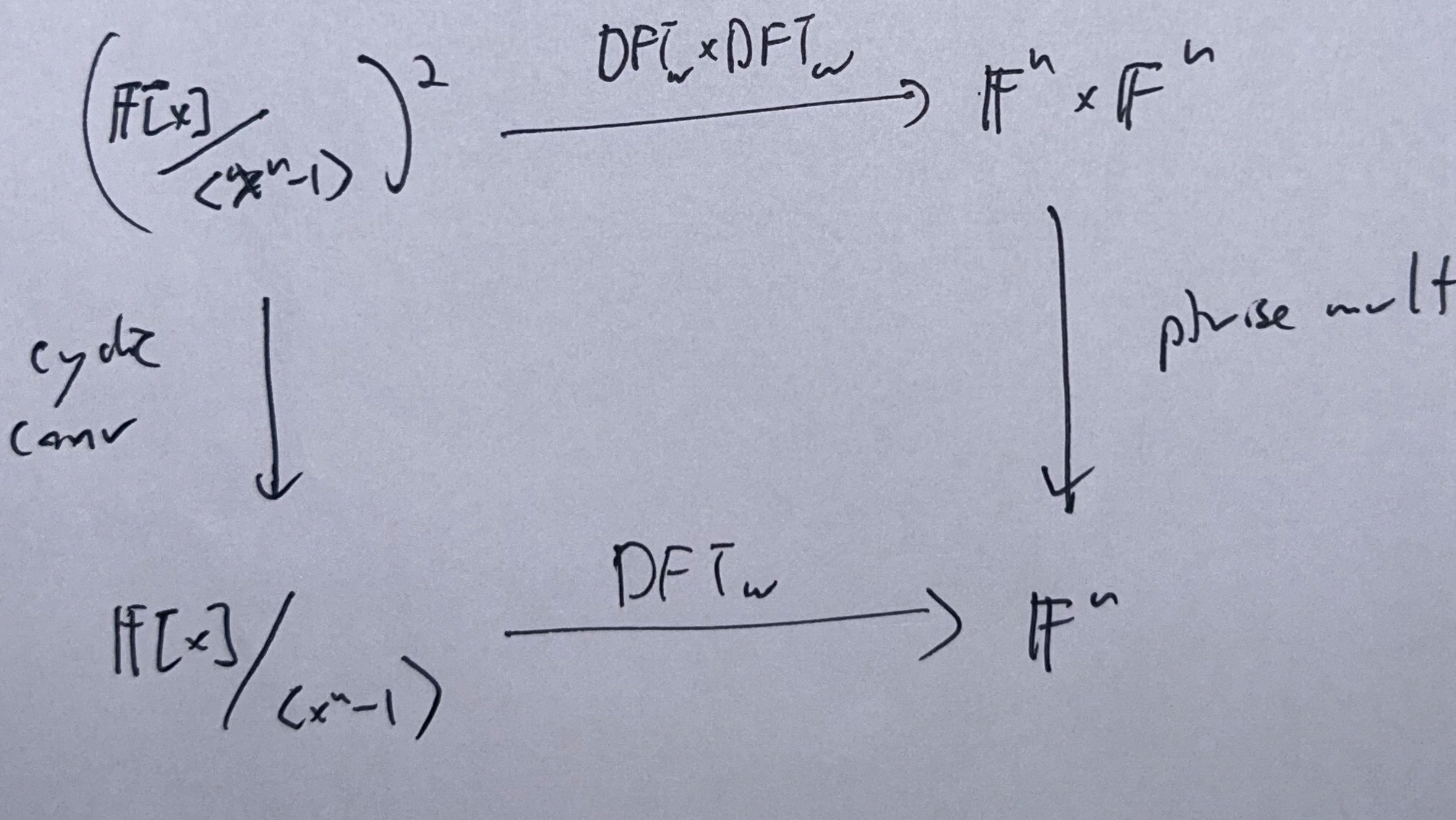
Prop If $f, g \in \mathbb{F}[x]$ have $\deg < n$,

$$DFT_\omega(f * g) = DFT_\omega(f) \cdot DFT_\omega(g)$$

↑ eltwise

DFT is an isom

Ex **Pf**



If $\deg(fg) < n$, then $f * g = fg \mod (x^n - 1)$

implies $fg = f * g$

so need 2^k th root of unity

w/ $2^{k-1} < 2n \leq 2^k$

$$2x^6+3x^5+x^4 + 3x^3+3x^2+x+1 \mod x^4-1$$

upper lower

$$= (2x^3+3x+1)(x^4-1) + \text{lower} + (2x^2+3x+1)$$

add back

$$= 3x^3+3x^2+x+1 + (2x^2+3x+1) \mod (x^4-1)$$

lower upper

$$= 3x^3+5x^2+4x+2 \mod (x^4-1)$$

So we want to find a $k \hat{a} \log n$

and show DFT & inverse is $O(n \log n)$

(poly mult will be $18 n \log n + O(n)$)

interpolation is also kind of a DFT

Vandermonde matrix

$$V_\omega = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \omega^{2(n-1)} \\ \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \omega^{(n-1)^2} \end{pmatrix} = (\omega^{jk})_{0 \le j,k < n}$$

is the matrix of the linear multipoint evaluation map coeffs $\to$ evals at $1, \omega, \ldots$

V_ω is invertible, $(V_\omega)^{-1} = \frac{1}{n} V_{\omega^{-1}}$ so everything works both ways

But applying V_ω can be done faster than taking n, n-wise dot products.

The idea is as follows

n even (usually $n = 2^k$ for some k)

w a primitive n^{th} root of unity

f deg $< n$

To evaluate f at $1, w, w^2, \ldots, w^{n-1}$,

we divide f by $x^{n/2} - 1$

and $x^{n/2} + 1$ with remainders:
obtaining quotients + remainders

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

w/ $q_0, r_0, q_1, r_1 \in \mathbb{F}[x]$, deg $< \frac{n}{2}$

we only need the remainders, and (Ex),

- we can get r_0 by adding the upper $\frac{n}{2}$ coeff of f to the lower $\frac{n}{2}$ coeff
- get r_1 by subtracting upper $\frac{n}{2}$ from lower $\frac{n}{2}$

Example 8.10

for just n additions, ie

$f = F_1 x^{\frac{n}{2}} + F_0 \Rightarrow (x^{\frac{n}{2}} - 1) \mid f - F_0 - F_1$ and $r_0 = F_0 + F_1$

$r_1 = F_0 - F_1$

Then plug in a power of ω

$$f(\omega^{2\ell}) = q_0(\omega^{2\ell})(\omega^{2\ell}-1) + r_0(\omega^{2\ell}) = r_0(\omega^{2\ell})$$

and $$f(\omega^{2\ell+1}) = q_1(\omega^{2\ell+1})(\omega^{n\ell}\omega^{\frac{n}{2}}+1) + r_1(\omega^{2\ell+1}) = r_1(\omega^{2\ell+1})$$

for $0 \le \ell < \frac{n}{2}$. (note $\omega^{n\ell} = 1$, $\omega^{\frac{n}{2}} = -1$)

since $0 = \omega^n - 1 = (\omega^{\frac{n}{2}}-1)(\omega^{\frac{n}{2}}+1)$

and $\omega^{\frac{n}{2}} - 1 \neq 0$.

So we have all the eval pts split into even & odd powers of deg $< n$ f

$f(\omega^{2\ell}) = r_0(\omega^{2\ell})$ top

$f(\omega^{2\ell+1}) = r_1(\omega^{2\ell+1}) = r_1(\omega(\omega^{2\ell}))$

but, r_0, r_1 have degree $< \frac{n}{2}$

- ω^2 is a primitive $(\frac{n}{2})^{th}$ root of unity

So we recurse.

FFT Alg

Input $n = 2^k$ $\quad f = \sum_{j=0}^{n-1} f_j x^j$

$w, w^2, \dots, w^{n-1}$ powers of a primitive n^{th} root of unity

Output $DFT_w(f) = (f(1), f(w), \dots, f(w^{n-1})) \in \mathbb{F}_q^n$

1. If $n = 1$ return f_0

2. $r_0 \leftarrow \sum_{0 \le j < \frac{n}{2}} (f_j + f_{j+\frac{n}{2}}) x^j$

$r_1^* \leftarrow \sum_{0 \le j < \frac{n}{2}} (f_j - f_{j+\frac{n}{2}}) w^j x^j$ $\qquad r_1^* = r_1(wx)$

3. recurse to evaluate r_0 and r_1^* at powers of w^2

4. return

$$r_0(1), r_1^*(1), r_0(w^2), r_1^*(w^2), \dots, r_0(w^{n-2}), r_1^*(w^{n-2})$$

$\frac{3}{2} n \log n$ field ops

$n \log n$ additions in $\mathbb{F}$

$\frac{n}{2} \log n$ multiplications by powers of w

Total $\frac{3}{2} n \log n$ field ops.