Counting points on hyperelliptic curves

David Harvey

University of New South Wales

19th November 2012, CARMA, University of Newcastle

Elliptic curves

Let p be a prime.

Let X be an elliptic curve over $\mathbf{F}_p$.

Want to compute $\#X(\mathbf{F}_p)$, the number of $\mathbf{F}_p$-rational points on X.

Example: X might be given by a Weierstrass equation

$$y^2 = x^3 + ax + b$$

for some $a, b \in \mathbf{F}_p$.

Then $\#X(\mathbf{F}_p)$ is simply the number of solutions $(x, y) \in \mathbf{F}_p \times \mathbf{F}_p$, plus the point at infinity.

Applications: number theory, cryptography, ...

Elliptic curves

Some algorithms:

- Naive: try every pair $x, y \in \mathbf{F}_p$. Complexity: $O(p^{2+\varepsilon})$.
- Slightly better: try every $x \in \mathbf{F}_p$, test if $x^3 + ax + b$ is a square modulo p. Complexity: $O(p^{1+\varepsilon})$.
- Shanks–Mestre: use group structure on $X(\mathbf{F}_p)$, "baby-step, giant-step" algorithm, and the Hasse bound

$$|\#X(\mathbf{F}_p) - (p+1)| \leq 2\sqrt{p}.$$

  Complexity: $O(p^{1/4+\varepsilon})$.
- Schoof: compute Frobenius action on ℓ-torsion. Complexity: $O((\log p)^{5+\varepsilon})$. Polynomial time!
- Schoof–Elkies–Atkin (SEA): heuristically $O((\log p)^{4+\varepsilon})$.

Hyperelliptic curves

Now let X be a hyperelliptic curve over $\mathbf{F}_p$ of genus g.

Example: take an equation

$$y^2 = f(x)$$

where $f \in \mathbf{F}_p[x]$ is monic, squarefree, of degree $2g + 1$.

Elliptic curves are the special case where $g = 1$.

We want $\#X(\mathbf{F}_p)$, or more generally the zeta function of X:

$$Z_X(T) = \exp\left(\sum_{n \geq 1} \frac{\#X(\mathbf{F}_{p^n})}{n} T^n\right) \in \mathbf{Z}[[T]].$$

This encodes the number of points over all finite extensions of $\mathbf{F}_p$.

Hyperelliptic curves

$Z_X(T)$ is a rational function, and is determined by $\#X(\mathbf{F}_p)$, $\#X(\mathbf{F}_{p^2}), \ldots, \#X(\mathbf{F}_{p^g})$.

Genus one (elliptic curves):

$$Z_X(T) = \frac{1 - a_1 T + pT^2}{(1-T)(1-pT)}.$$

Here $\#X(\mathbf{F}_p)$ determines $a_1 \in \mathbf{Z}$, and thus $Z_X(T)$.

Genus two:

$$Z_X(T) = \frac{1 - a_1 T + a_2 T^2 - p a_1 T^3 + p^2 T^4}{(1-T)(1-pT)}.$$

Here $\#X(\mathbf{F}_p), \#X(\mathbf{F}_{p^2})$ determine $a_1, a_2 \in \mathbf{Z}$, and thus $Z_X(T)$.

Hyperelliptic curves

Some algorithms:

- Naive: count $\#X(\mathbf{F}_{p^i})$ for $i = 1, \ldots, g$. Complexity: $O(p^{g+\varepsilon})$.
- Use group structure of Jacobian of X. Complexity: saves only a constant power of p over naive method (Elkies).
- Schoof–Pila: compute Frobenius action on ℓ-torsion of Jacobian. Complexity: $O((\log p)^{C_g+\varepsilon})$. Adleman–Huang: can take $C_g = O(g^{2+\varepsilon})$, e.g. for $g = 2$, get $O((\log p)^{8+\varepsilon})$ (?).
- Kedlaya: compute Frobenius action on p-adic cohomology. Complexity: $O(p^{1+\varepsilon} g^{4+\varepsilon})$.
- Many others!

All have complexity exponential in either $\log p$ or g (or both).

Hyperelliptic curves

Unsolved problem:

> *Let X be a hyperelliptic curve of genus g over $\mathbf{F}_p$.*
>
> *Does there exist an algorithm that computes the zeta function of X — or even just $\#X(\mathbf{F}_p)$ — whose complexity is polynomial in $\log p$ and g?*

Note that the input size is $O(g \log p)$ bits.

Today I will discuss a new algorithm that gives a partial positive solution to this problem.

Main theorem

Setup: let $f \in \mathbf{Z}[x]$ of degree $2g+1$, monic and squarefree.

Let X be the hyperelliptic curve $y^2 = f(x)$ over $\mathbf{Q}$.

For $p \nmid \operatorname{disc} f$, let X_p be the reduction of X modulo p. It is a hyperelliptic curve of genus g over $\mathbf{F}_p$.

Let $\|f\|$ be the max of the absolute values of the coefficients of f.

Theorem (H., 2012)

Let $N \geq 3$. *Assume* $\log g = O(\log N)$ *and* $\log \|f\| = O(\log N)$.

Can compute $Z_{X_p}(T)$ *for all* $p < N$ *simultaneously*, $p \nmid \operatorname{disc} f$, *in*

$$O(g^{8+\varepsilon} N \log^{3+\varepsilon} N)$$

bit operations.

Main theorem

The average complexity per prime is

$$O(g^{8+\varepsilon} \log^{4+\varepsilon} N),$$

which is polynomial in the input size.

Why would we want to compute $Z_{X_p}(T)$ for all $p < N$?

Main application: numerical approximation of the L-function for X,

$$L(X, s) = \prod_p L_p(p^{-s})^{-1}.$$

Here $L_p(T)$ = numerator of $Z_{X_p}(T)$ (or a slightly different definition for the bad primes).

Useful for investigating Birch–Swinnerton-Dyer conjecture, Sato–Tate conjecture, ...

Main theorem

Proof is based on Kedlaya's approach via p-adic cohomology.

It is fairly technical — see arXiv paper for details ("Counting points on hyperelliptic curves in average polynomial time").

I won't talk about it today.

Elliptic curves again

Instead I will sketch a proof of the following much weaker result, using a method that conveys some of the key new ideas needed for the main theorem.

Theorem

Let X be an elliptic curve over $\mathbf{Q}$ with a rational 2-torsion point. We may compute $\#X(\mathbf{F}_p)$ for all $p < N$ in $O(N \log^{3+\varepsilon} N)$ bit operations.

Very explicitly: we are given an equation

$$y^2 = x^3 + bx^2 + cx$$

with $b, c \in \mathbf{Z}$, $c \neq 0$, $b^2 - 4c \neq 0$.

Want to count the number of solutions modulo p for every $p < N$.

Elliptic curves again

As before, let $a_1 = p + 1 - \#X(\mathbf{F}_p)$.

Write $g(x)$ for the polynomial $x^2 + bx + c$.

Claim: $a_1 \equiv$ the coefficient of $x^{(p-1)/2}$ in $g(x)^{(p-1)/2} \pmod{p}$.

Proof:

$$\begin{aligned}
\#X(\mathbf{F}_p) &= 1 + \sum_{x=0}^{p-1} \left[1 + \left(\frac{xg(x)}{p} \right) \right] \\
&\equiv 1 + \sum_{x=0}^{p-1} (xg(x))^{(p-1)/2} \pmod{p} \\
&\equiv 1 + \sum_{x=0}^{p-1} x^{(p-1)/2} (c_0 + c_1 x + \cdots + c_{p-1} x^{p-1}) \\
&\equiv 1 - c_{(p-1)/2}.
\end{aligned}$$

Elliptic curves again

Note that $a_1 \pmod p$ determines a_1 (and hence $\#X(\mathbf{F}_p)$) for large enough p, because we know $|a_1| \leq 2\sqrt{p}$.

(In fact $p \geq 17$ is enough.)

So now our problem is:

Given $g(x) = x^2 + bx + c \in \mathbf{Z}[x]$, compute the coefficient of $x^{(p-1)/2}$ in $g^{(p-1)/2}$, modulo p, for all $p < N$.

Elliptic curves again

Example: $y^2 = x^3 + x^2 + 2x$.

$$
\begin{aligned}
g &= x^2 + x + 2 \\
g^2 &= x^4 + 2x^3 + 5x^2 + 4x + 4 \\
g^3 &= x^6 + 3x^5 + 9x^4 + 13x^3 + 18x^2 + 12x + 8 \\
g^4 &= x^8 + 4x^7 + 14x^6 + 28x^5 + 49x^4 + 56x^3 + 56x^2 + 32x + 16 \\
g^5 &= x^{10} + 5x^9 + 20x^8 + 50x^7 + 105x^6 + 161x^5 + 210x^4 + \cdots \\
&\vdots \\
g^{50} &= \cdots + 92553315517865357744346828 85x^{50} + \cdots
\end{aligned}
$$

We want $1 \pmod 3$, $5 \pmod 5$, $13 \pmod 7$, $161 \pmod{11}$, $\ldots$, $92553315517865357744346828 85 \pmod{101}$, for all p up to some bound N.

Elliptic curves again

Challenges:

- We want an algorithm that runs in time $O(N^{1+\varepsilon})$.
- We don't have time to write down all the g^m's; they occupy $O(N^{3+\varepsilon})$ bits altogether.
  (Each has $O(m)$ coefficients with $O(m^{1+\varepsilon})$ bits.)
- We don't even have time to write down just the middle coefficient of each g^m; that would occupy $O(N^{2+\varepsilon})$ bits.
- Each coefficient of g^m depends on many coefficients of previous g^m's... how can we get just the middle one?
- Even if we found a relation that isolated just the middle coefficients... we want each one modulo a different prime! Knowledge of $161 \pmod{11}$ tells us *nothing* about $92553315517865357744346828 85 \pmod{101}$.

Isolating the middle coefficient

First we deal with the problem of isolating the middle coefficient.

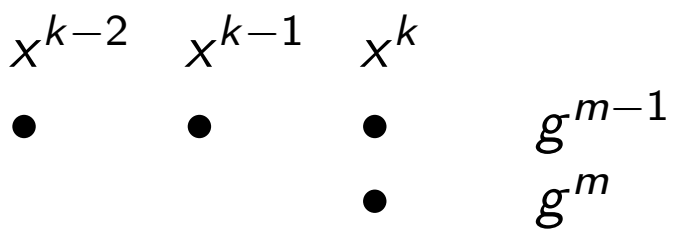
Let $u_{m,k}$ = coefficient of x^k in $(x^2 + bx + c)^m$.

We can write down some recurrences for the $u_{m,k}$.

Equating coefficients in $g^m = g \cdot g^{m-1}$ yields:

$$u_{m,k} = u_{m-1,k-2} + bu_{m-1,k-1} + cu_{m-1,k}.$$

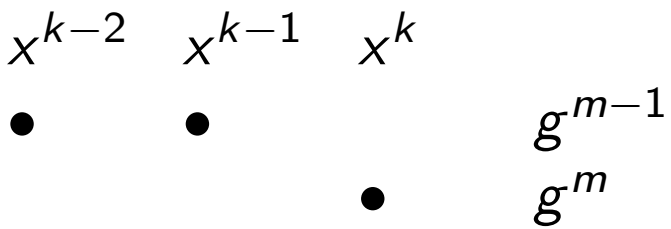
This gives a linear relation between four coefficients:



Isolating the middle coefficient

There is another independent relation, obtained by equating coefficients in $(g^m)' = mg' \cdot g^{m-1}$:

$$ku_{m,k} = 2mu_{m-1,k-2} + bmu_{m-1,k-1}.$$



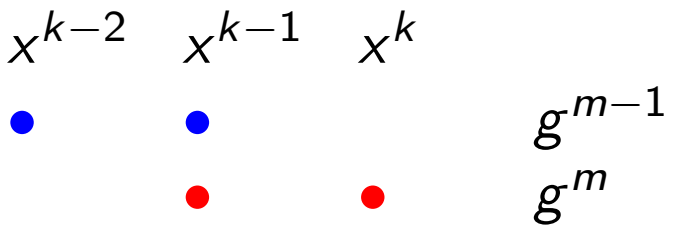
(For the experts: taking the derivative here is closely related to de Rham cohomology that shows up in the definition of the Monsky–Washnitzer cohomology used in Kedlaya's algorithm.)

Isolating the middle coefficient

After some algebra we get

$$\begin{pmatrix} u_{m,k-1} \\ u_{m,k} \end{pmatrix} = \begin{pmatrix} \frac{m}{2m-k+1} b & \frac{2m}{2m-k+1} c \\ \frac{2m}{k} & \frac{m}{k} b \end{pmatrix} \begin{pmatrix} u_{m-1,k-2} \\ u_{m-1,k-1} \end{pmatrix}.$$

This expresses the red coefficients in terms of the blue coefficients:



Isolating the middle coefficient

Specialising to $k = m$, we get

$$\begin{pmatrix} u_{m,m-1} \\ u_{m,m} \end{pmatrix} = \frac{1}{m+1} \begin{pmatrix} bm & 2cm \\ 2(m+1) & b(m+1) \end{pmatrix} \begin{pmatrix} u_{m-1,m-2} \\ u_{m-1,m-1} \end{pmatrix}.$$

Recall our example:

$$\begin{aligned} g &= x^2 + x + 2 \\ g^2 &= x^4 + 2x^3 + 5x^2 + 4x + 4 \\ g^3 &= x^6 + 3x^5 + 9x^4 + 13x^3 + 18x^2 + 12x + 8 \\ g^4 &= x^8 + 4x^7 + 14x^6 + 28x^5 + 49x^4 + 56x^3 + 56x^2 + 32x + 16 \end{aligned}$$

$$\begin{pmatrix} 18 \\ 13 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 3 & 12 \\ 8 & 4 \end{pmatrix} \begin{pmatrix} 4 \\ 5 \end{pmatrix}, \qquad \begin{pmatrix} 56 \\ 49 \end{pmatrix} = \frac{1}{5} \begin{pmatrix} 4 & 16 \\ 10 & 5 \end{pmatrix} \begin{pmatrix} 18 \\ 13 \end{pmatrix}, \ldots$$

Isolating the middle coefficient

In other words, we want to compute

- $M_1 \pmod 3$
- $M_2 M_1 \pmod 5$
- $M_3 M_2 M_1 \pmod 7$
- ...

where

$$M_m = \frac{1}{m+1} \begin{pmatrix} bm & 2cm \\ 2(m+1) & b(m+1) \end{pmatrix}.$$


AND NOW FOR
SOMETHING
COMPLETELY
DIFFERENT

Wilson primes

Recall Wilson's theorem:

$$(p-1)! = -1 \pmod{p}.$$

A *Wilson prime* is a prime satisfying

$$(p-1)! = -1 \pmod{p^2}.$$

Only known examples: 5, 13, 563.

Costa–Gerbicz–H. (2012): there are no others for $p < 2 \times 10^{13}$.

Heuristics: there are $O(\log \log N)$ Wilson primes less than N.

Wilson primes

How do we test if p is a Wilson prime?

Need to compute $u_p = (p-1)! \pmod{p^2}$.

- Naive: just multiply out the product, reduce modulo p^2 occasionally. Complexity: $O(p^{1+\varepsilon})$.
- Strassen: clever polynomial evaluation scheme. Complexity: $O(p^{1/2+\varepsilon})$.
- Costa–Gerbicz–H.: can compute u_p for all $p < N$ in $O(N \log^{3+\varepsilon} N)$ bit operations, i.e. $O(\log^{4+\varepsilon} N)$ per prime.

In the next few slides I will sketch this last algorithm.

Wilson primes

I will assume standard results on fast integer arithmetic.

Namely: given integers a, b with at most n bits, we can compute

- the product ab,
- the quotient $\lfloor a/b \rfloor$, and
- the remainder $a \bmod b$

in $O(n \log^{1+\varepsilon} n)$ bit operations.

(Main tool is the fast Fourier transform.)

Wilson primes

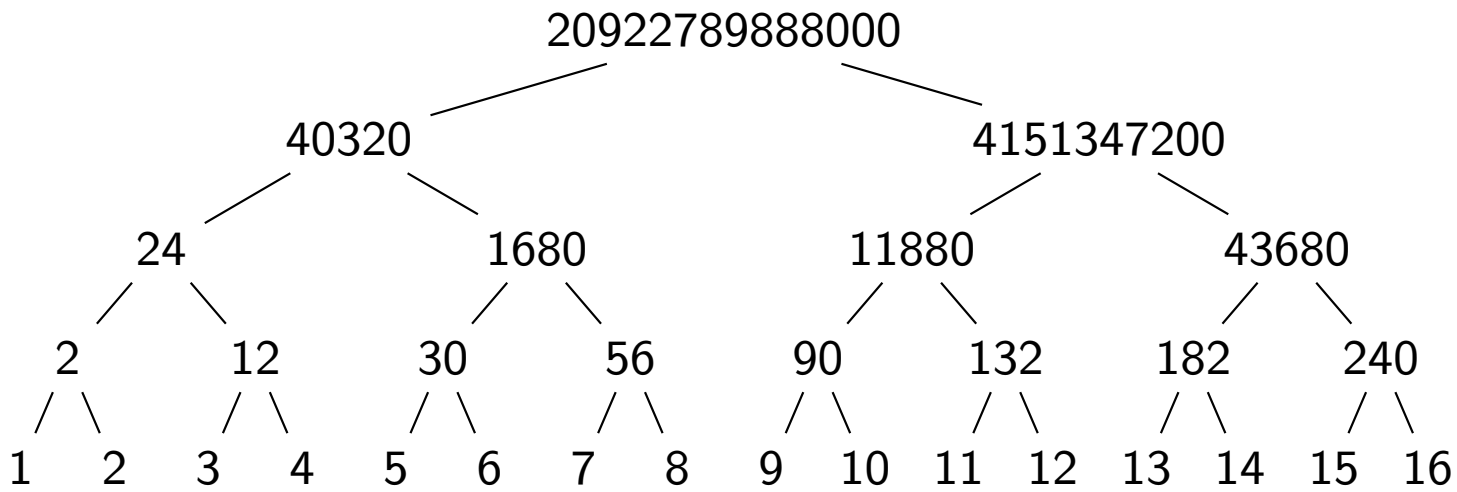
We want to compute:

- $1 \cdot 2 \pmod{3^2}$
- $1 \cdot 2 \cdot 3 \cdot 4 \pmod{5^2}$
- $1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \pmod{7^2}$
- $\ldots$

(Look familiar?)

Wilson primes

First compute the *value tree*. This is a product tree for the integers up to N. We work from the bottom to the top:



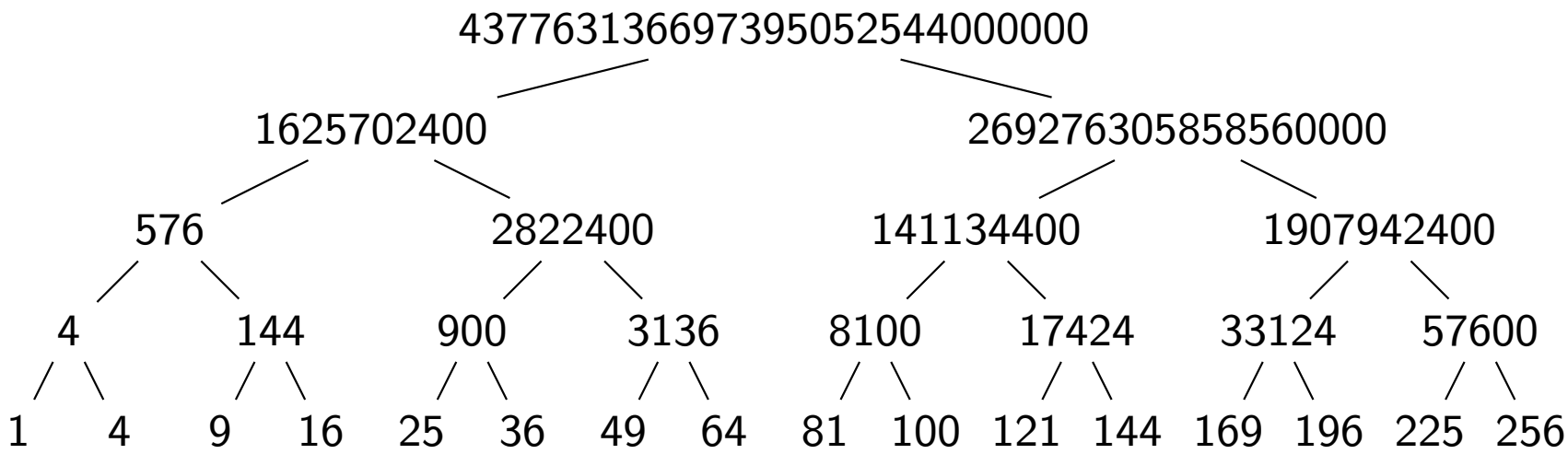
Total bits at each level: $O(N \log N)$.

Cost to compute each level: $O(N \log^{2+\varepsilon} N)$.

Cost to compute whole tree: $O(N \log^{3+\varepsilon} N)$.

Wilson primes

Next compute the *modulus tree*. This is a product tree for the corresponding squares:



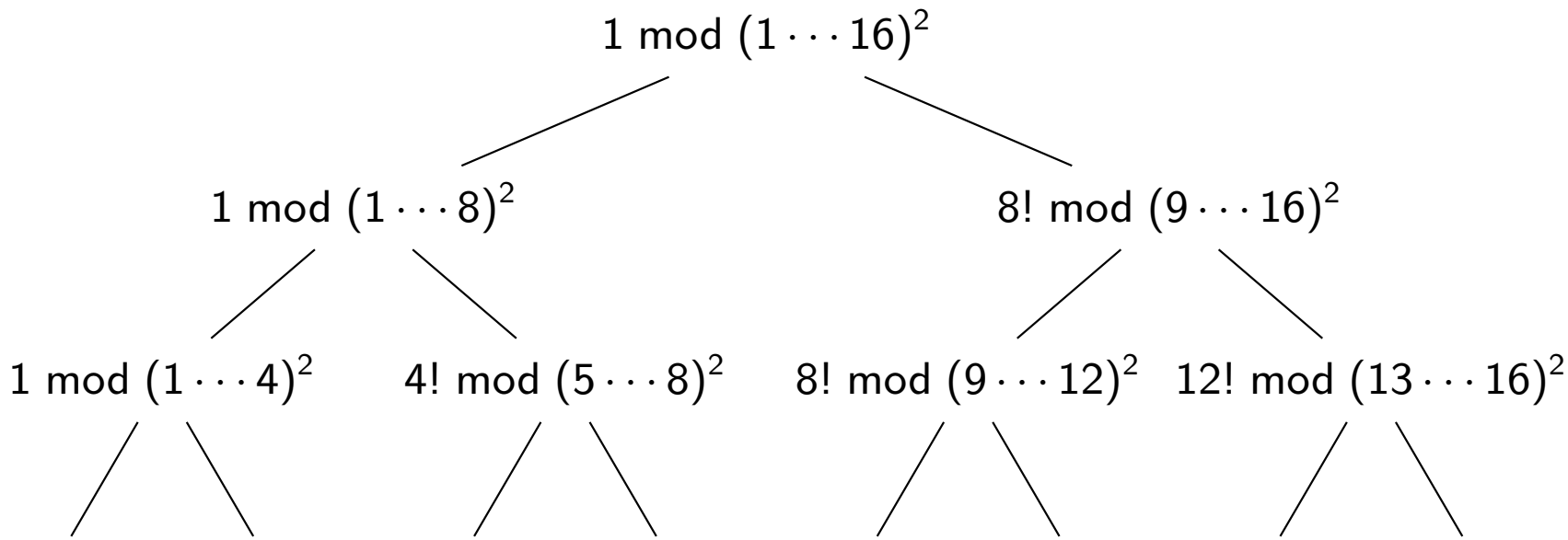
Cost is the same: $O(N \log^{3+\varepsilon} N)$.

Wilson primes

Now the interesting part: we compute, from top to bottom, an *accumulating remainder tree*.

Each node is the product of the nodes of the value tree to the *left* of that node, modulo the corresponding node in the modulus tree.

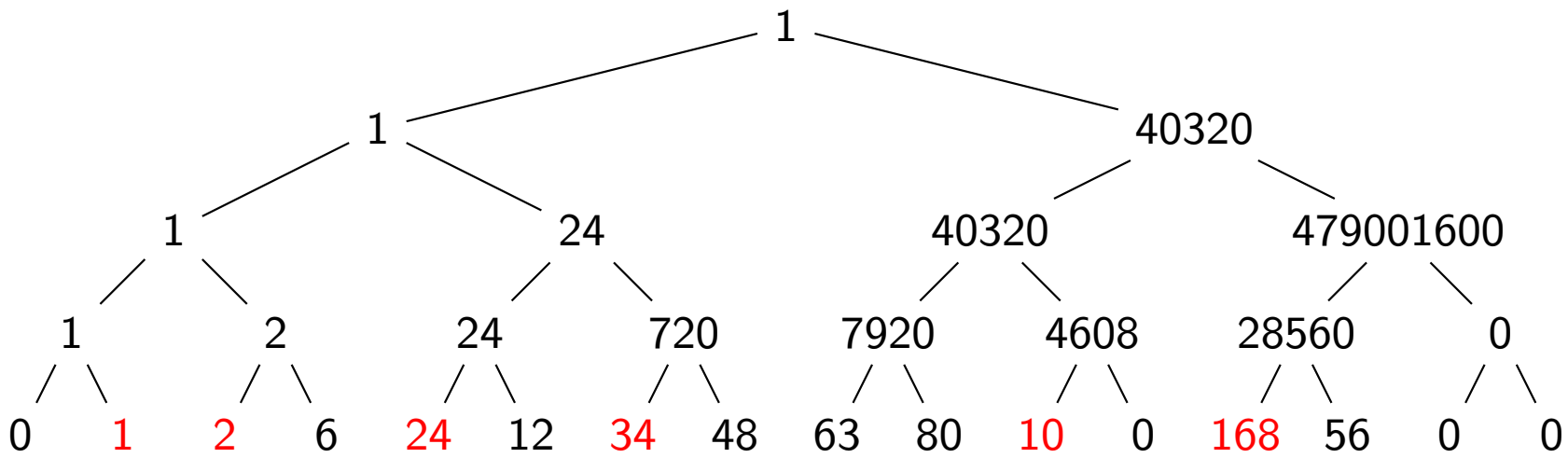
Here are the first few layers:



Wilson primes

Cost is the same: $O(N \log^{3+\varepsilon} N)$.

Here are the actual values for our example:



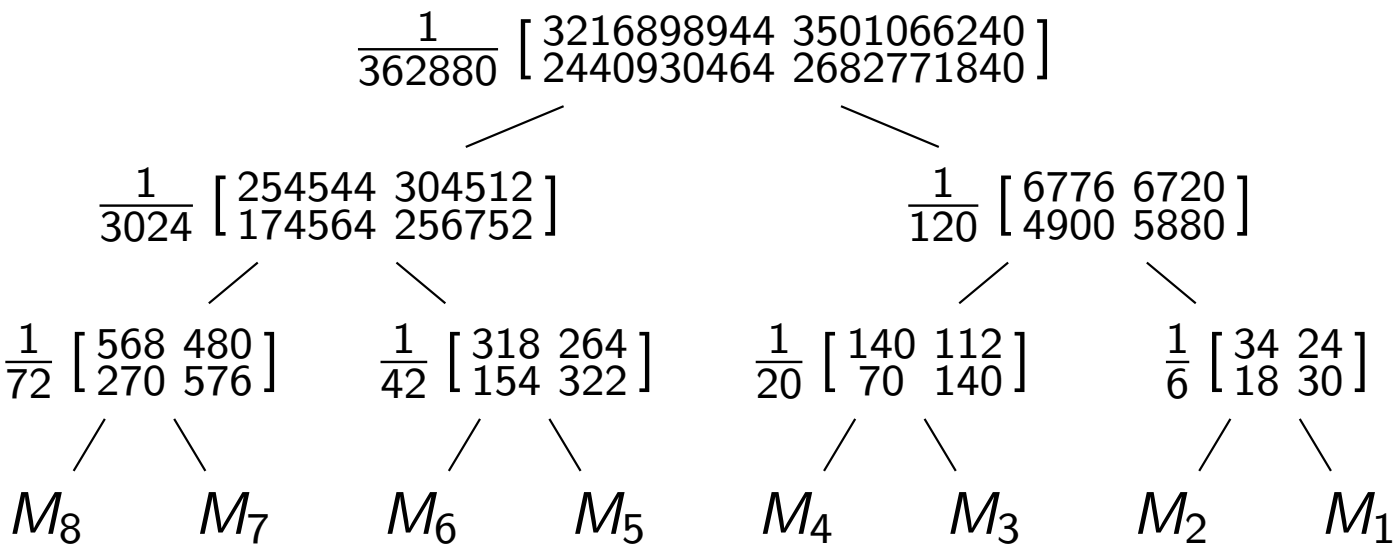
We can read off the bottom of the tree:

$$4! = 24 \pmod{5^2},$$
$$6! = 34 \pmod{7^2},$$
$$10! = 10 \pmod{11^2} \ldots$$

Back to elliptic curves

The same idea works almost verbatim for the elliptic curve problem.

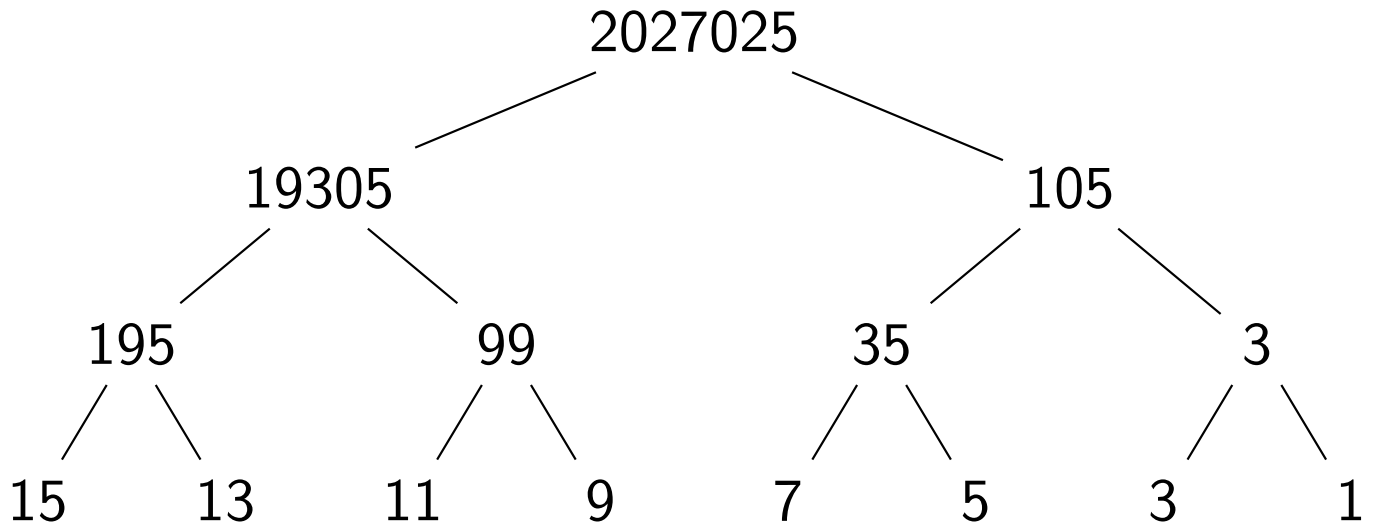
Here is the value tree, for our example $y^2 = x(x^2 + x + 2)$:



Note that we work from right-to-left instead of left-to-right.

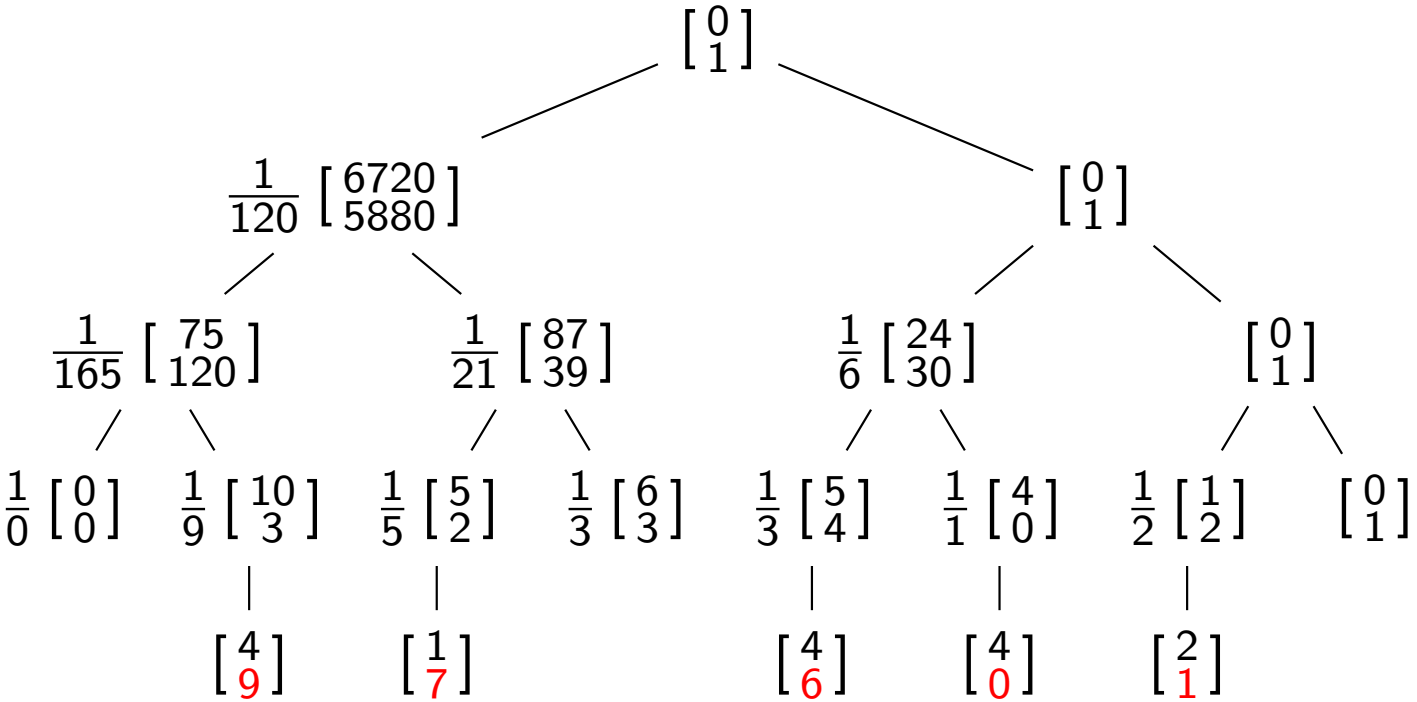
Back to elliptic curves

Here is the modulus tree:



Back to elliptic curves

And here is the accumulating remainder tree:



Therefore

$$\#X(\mathbf{F}_3) = 1 - 1 \pmod 3, \quad \#X(\mathbf{F}_{11}) = 1 - 7 \pmod{11},$$
$$\#X(\mathbf{F}_5) = 1 - 0 \pmod 5, \quad \#X(\mathbf{F}_{13}) = 1 - 9 \pmod{13}, \ldots$$

(Note that the curve is singular at $p = 7$.)

Back to elliptic curves

Conclusion: we can compute $\#X(\mathbf{F}_p)$ for all $p < N$ in $O(N \log^{3+\varepsilon} N)$ bit operations.

The algorithm is quite elementary — does not use any high-powered results about elliptic curves.

In fact it is the fastest known unconditional algorithm for this problem (deterministic or otherwise)!

Easy to generalise to the usual Weierstrass form $y^2 = x^3 + ax + b$, avoiding the restriction that the curve has a rational 2-torsion point. This leads to 3×3 matrices instead of 2×2.

Back to hyperelliptic curves

For a hyperelliptic curve of genus g, can generalise to computing the *Hasse–Witt* matrix. This is a $g \times g$ matrix whose characteristic polynomial is the numerator of the zeta function only *modulo* p.

For example, for a curve

$$y^2 = f(x) = x^5 + c_4 x^4 + \cdots + c_0,$$

we need to compute the coefficients of x^{p-1}, x^{p-2}, x^{2p-1}, x^{2p-2} of $f^{(p-1)/2}$.

For genus 2 this is essentially enough to determine $Z_X(T)$.

For genus 3 it doesn't determine $Z_X(T)$, but it is not hard to recover the missing information by other means.

For genus ≥ 4 there is probably too much information missing for this approach to be practical.

Thank you!