Counting points on elliptic curves

David Harvey

University of New South Wales

6th December 2012, Australian National University

Contents

- Counting points on elliptic curves, and why we care
- Statement of main result
- Beginning of proof
- Detour via Wilson primes
- End of proof
- Generalisations (time permitting)

Elliptic curves

Let p be a prime, and let E be an elliptic curve over $\mathbf{F}_p$.

Goal: compute $\#E(\mathbf{F}_p)$, the number of $\mathbf{F}_p$-rational points on E.

Concretely, if E is given by a Weierstrass equation

$$y^2 = x^3 + ax + b, \qquad a, b \in \mathbf{F}_p,$$

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

Example: take the curve $y^2 = x^3 + x + 2$ over $\mathbf{F}_5$.

Then $E(\mathbf{F}_p)$ consists of

$$(1, \pm 2), (-1, 0),$$

and the point at infinity, so $\#E(\mathbf{F}_p) = 4$.

Elliptic curves

Why do we care?

Among other things, applications in:

- cryptography
- number theory

Cryptography

The set of points $E(\mathbf{F}_p)$ forms an abelian group under the elliptic curve group law.

The discrete logarithm problem (DLP): given $P \in E(\mathbf{F}_p)$ and some multiple $nP \in E(\mathbf{F}_p)$, find n.

It is believed that solving DLP is very difficult for this group in general. Best known complexity bound is essentially $O(\sqrt{p})$. No subexponential-time algorithms are known.

This makes the group suitable for cryptographic protocols such as Diffie–Hellman key exchange.

For these schemes, it is expected that an attacker would need to solve the DLP (or a related problem of similar difficulty).

Cryptography

However, if $\#E(\mathbf{F}_p)$ is composite, the DLP can be reduced to solving several smaller DLPs in subgroups of $E(\mathbf{F}_p)$.

The overall difficulty is controlled by the largest prime factor of $\#E(\mathbf{F}_p)$.

To avoid this attack, ideally we want $\#E(\mathbf{F}_p)$ to be prime.

How do we find such an E?

One simple method: choose E randomly. Compute $\#E(\mathbf{F}_p)$. If it is composite, throw it away and try again. Repeat until we win.

Cryptography

Example: take $p = 10^{77} + 21$ and the curve $y^2 = x^3 + x + 61$.

Then $\#E(\mathbf{F}_p)$ is

$$100000000000000000000000000000000000000000613196722711727187812910225277803876 03,$$

which is prime.

This is big enough for NSA 'Secret' classification.

(For 'Top Secret' you need about 50% more digits.)

My laptop computed $\#E(\mathbf{F}_p)$ in about 6 seconds, using the software package Pari.

Clearly Pari is not simply enumerating the points!

Cryptography

Pari uses the Schoof–Elkies–Atkin (SEA) algorithm when p is this large.

Heuristic complexity of SEA is $\log^{4+\varepsilon} p$ bit operations — polynomial time.

Schoof's original algorithm (1985) has a fully established deterministic complexity bound of $\log^{5+\varepsilon} p$.

Basic idea: compute action of p-power Frobenius on the ℓ-torsion points of the curve, for small primes ℓ. This determines $\#E(\mathbf{F}_p) \pmod{\ell}$. Now use CRT.

Number theory

Let E be an elliptic curve over $\mathbf{Q}$, say

$$y^2 = x^3 + ax + b, \qquad a, b \in \mathbf{Q}.$$

The Mordell–Weil group $E(\mathbf{Q})$ is the set of rational solutions, with the usual elliptic curve group law.

It is a finitely generated abelian group (Mordell–Weil theorem).

The rank of $E(\mathbf{Q})$ is a very mysterious quantity.

We do not know any algorithm that is guaranteed to compute the rank!

Equivalently, we do not know how to find generators of $E(\mathbf{Q})$.

Number theory

The Birch–Swinnerton-Dyer conjecture (BSD) proposes a conjectural relationship between the rank of $E(\mathbf{Q})$ and the quantities $\#E(\mathbf{F}_p)$, where p varies over primes.

Here $E(\mathbf{F}_p)$ denotes the reduction of E modulo p — essentially, just reduce the defining equation modulo p.

One consequence of BSD is that

$$\prod_{p \le x} \frac{\#E(\mathbf{F}_p)}{p} \sim C(\log x)^{\operatorname{rank} E(\mathbf{Q})}$$

as $x \to \infty$.

In other words, curves with high rank over $\mathbf{Q}$ tend to have more points modulo p, on average, over many p.

Number theory

Another important question is the Sato–Tate conjecture (proved for curves over totally real fields?)

The Hasse bound states that

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

The Sato–Tate conjecture states that the limiting distribution of the normalised error term

$$\frac{\#E(\mathbf{F}_p) - (p+1)}{2\sqrt{p}} \in [-1, 1]$$

is given by the semicircular distribution

$$\frac{2}{\pi}\sqrt{1-t^2}\,dt$$

(except for curves with complex multiplication).

Number theory

For numerical investigation of phenomena such as BSD and Sato–Tate, we need an efficient algorithm for the following problem:

> *Let E be an elliptic curve over $\mathbf{Q}$, and let $N \geq 1$.*
>
> *Compute $\#E(\mathbf{F}_p)$ for all $p < N$.*

This is quite different to the cryptography situation, where we want to compute $\#E(\mathbf{F}_p)$ for a single, very large prime p.

Can we do better than just computing $\#E(\mathbf{F}_p)$ for each p separately?

Main theorem

Today I will sketch a new, quite elementary algorithm that proves the following:

Theorem

Let E be an elliptic curve over $\mathbf{Q}$. We may compute $\#E(\mathbf{F}_p)$ for all $p < N$ in $N \log^{3+\varepsilon} N$ (deterministic) bit operations.

In other words: $\log^{4+\varepsilon} N$ bit operations per prime (unconditional, deterministic).

Schoof's algorithm for each prime separately: $\log^{5+\varepsilon} N$ per prime (unconditional, deterministic).

SEA for each prime separately: $\log^{4+\varepsilon} N$ (heuristic, probabilistic).

Main theorem

The new algorithm is the fastest known unconditional algorithm for this problem (deterministic or otherwise).

Unfortunately, it is probably not practical. Over the feasible range of computation — say up to about $N = 10^{12}$ or so — it is almost certainly better to use the Shanks–Mestre algorithm, which has complexity $O(p^{1/4})$ per prime.

Nevertheless, the main interest of this algorithm is that is generalises readily to curves of higher genus, where it is most certainly practical. I will discuss this at the end of the talk if there is enough time.

Beginning of proof

For simplicity, in this talk I will restrict to the case of an elliptic curve with a rational 2-torsion point.

Such a curve is given by an equation

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

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

It is not difficult to generalise to an arbitrary Weierstrass equation.

Beginning of proof

Claim: let $(x^2 + bx + c)^{(p-1)/2} = x^{p-1} + \cdots + s_1 x + s_0$. Then

$$\#E(\mathbf{F}_p) = 1 - s_{(p-1)/2} \pmod{p}.$$

Proof:

$$\begin{aligned}
\#E(\mathbf{F}_p) &= 1 + \sum_{x=0}^{p-1} \left[1 + \left(\frac{x^3 + bx^2 + cx}{p} \right) \right] \\
&\equiv 1 + \sum_{x=0}^{p-1} (x^3 + bx^2 + cx)^{(p-1)/2} \pmod{p} \\
&\equiv 1 + \sum_{x=0}^{p-1} x^{(p-1)/2} (x^{p-1} + \cdots + s_1 x + s_0) \\
&\equiv 1 - s_{(p-1)/2}.
\end{aligned}$$

Beginning of proof

Note that $\#E(\mathbf{F}_p) \pmod p$ determines $\#E(\mathbf{F}_p)$ for large enough p, because we know $|\#E(\mathbf{F}_p) - (p+1)| \leq 2\sqrt{p}$.

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

So now our problem is:

> *Compute the 'central' coefficient of* $(x^2 + bx + c)^{(p-1)/2}$, *modulo* p, *for all* $p < N$.

Beginning of proof

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

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

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.

Beginning of proof

Challenges:

- We want an algorithm that runs in time $N^{1+\varepsilon}$.
- We don't have time to write down all the f^m's; they occupy $N^{3+\varepsilon}$ bits altogether.
  (Each has $O(m)$ coefficients with $m^{1+\varepsilon}$ bits.)
- We don't even have time to write down just the middle coefficient of each f^m; that would occupy $N^{2+\varepsilon}$ bits.
- Each coefficient of f^m depends on many coefficients of previous f^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 $92553315517865357744346828885 \pmod{101}$.

Isolating the middle coefficient

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

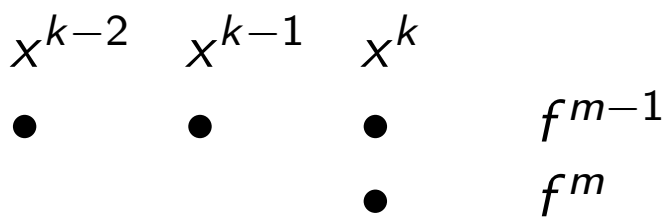
Let $u_{m,k} =$ coefficient of x^k in $f^m = (x^2 + bx + c)^m$.

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

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

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

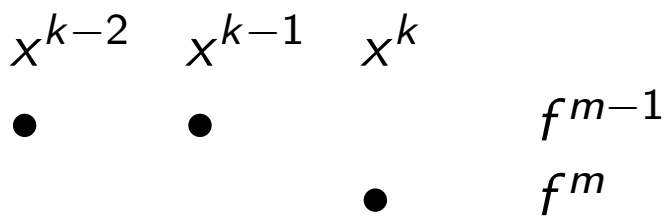
This gives a linear relation between four coefficients:



Isolating the middle coefficient

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

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

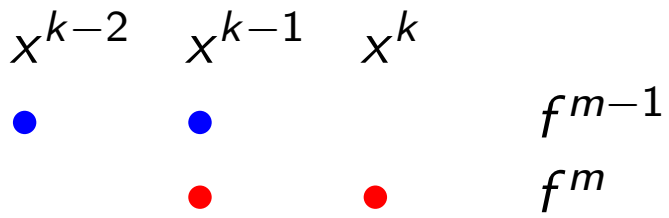


Isolating the middle coefficient

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:

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

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

(Interesting divisibility properties!)

Isolating the middle coefficient

In other words, we want to compute the lower-right entries of

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

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.

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

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

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: $p^{1+\varepsilon}$.
- Strassen: clever polynomial evaluation scheme. Complexity: $p^{1/2+\varepsilon}$.
- Costa–Gerbicz–H.: can compute u_p for all $p < N$ in $N \log^{3+\varepsilon} N$ bit operations, i.e. $\log^{4+\varepsilon} N$ 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 $n \log^{1+\varepsilon} n$ bit operations.

(Main tool is the fast Fourier transform.)

Wilson primes

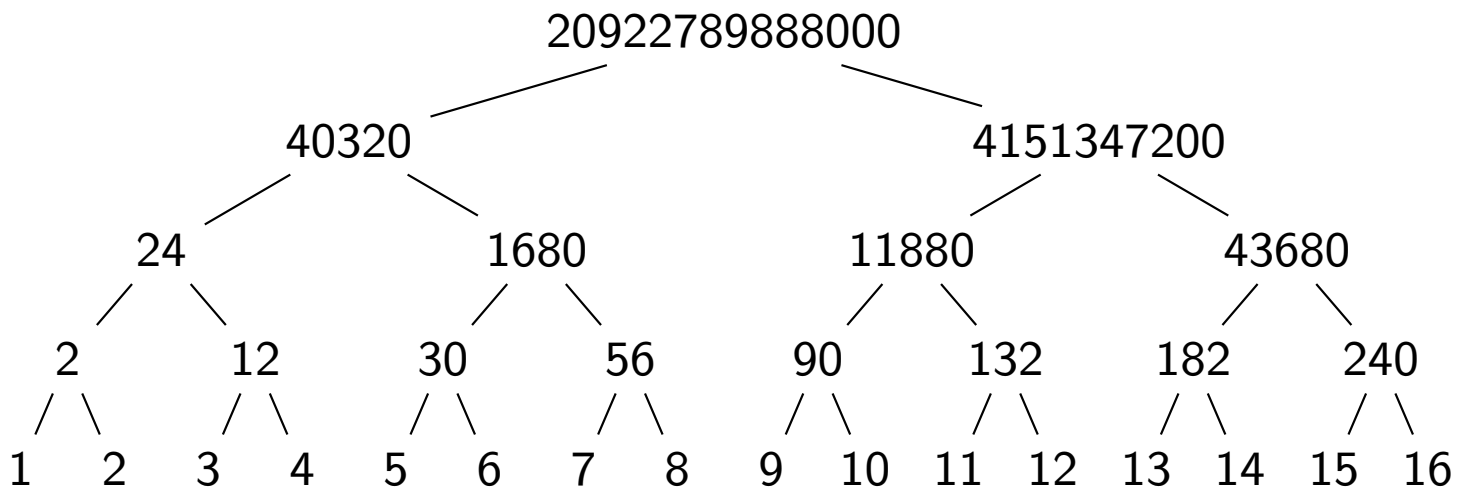
We want to compute:

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

(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:



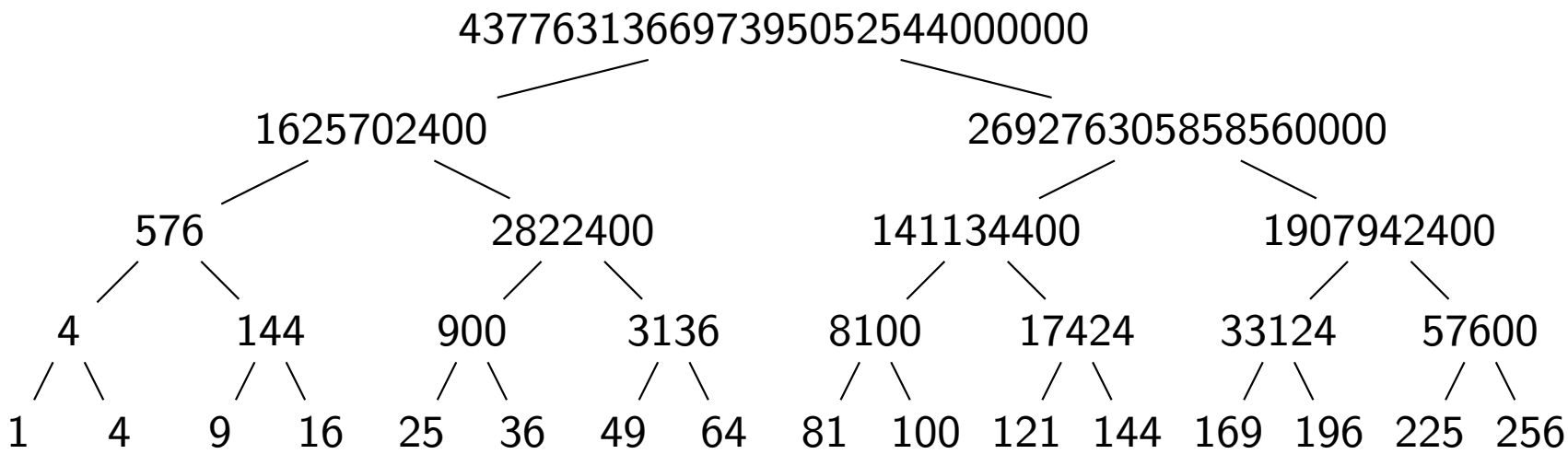
Total bits at each level: $O(N \log N)$.

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

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

Wilson primes

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



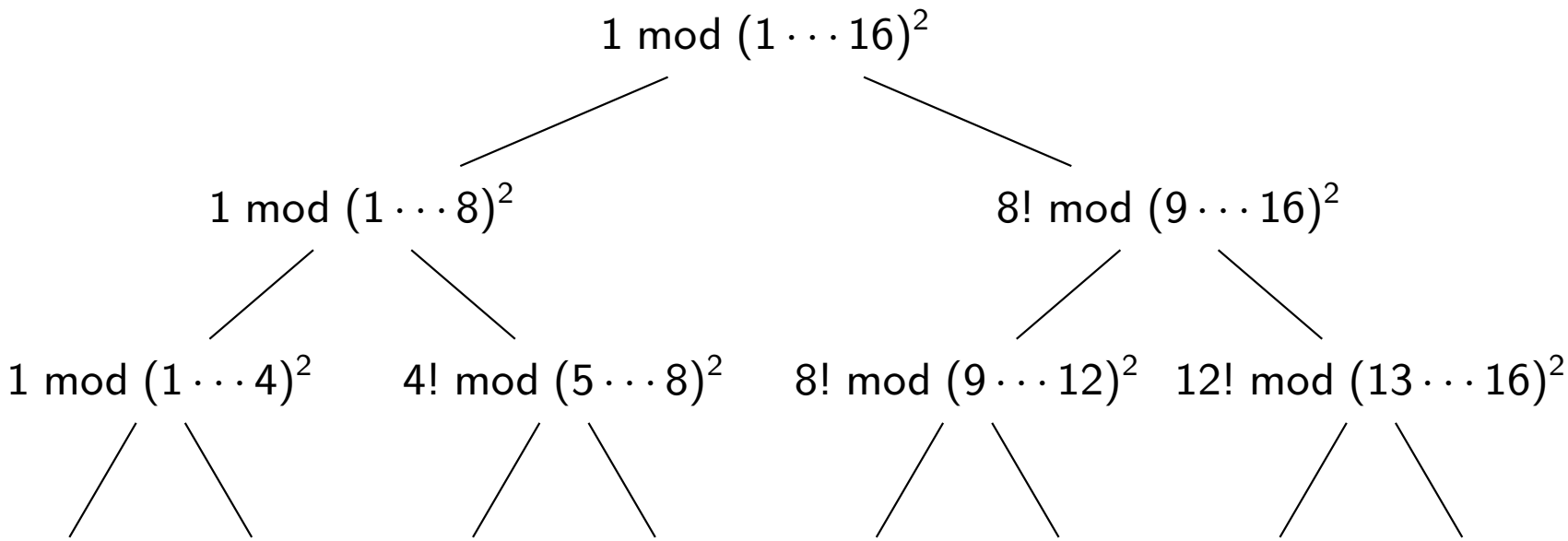
Cost is the same: $N \log^{3+\varepsilon} N$.

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.

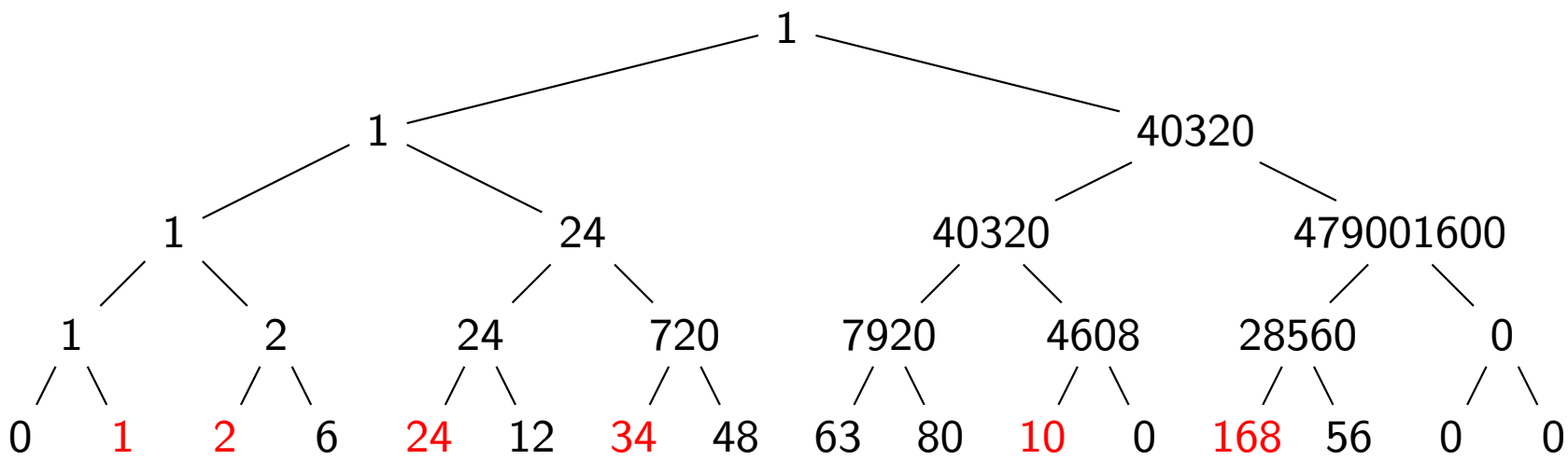
Here are the first few layers:



Wilson primes

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

Here are the actual values for our example:



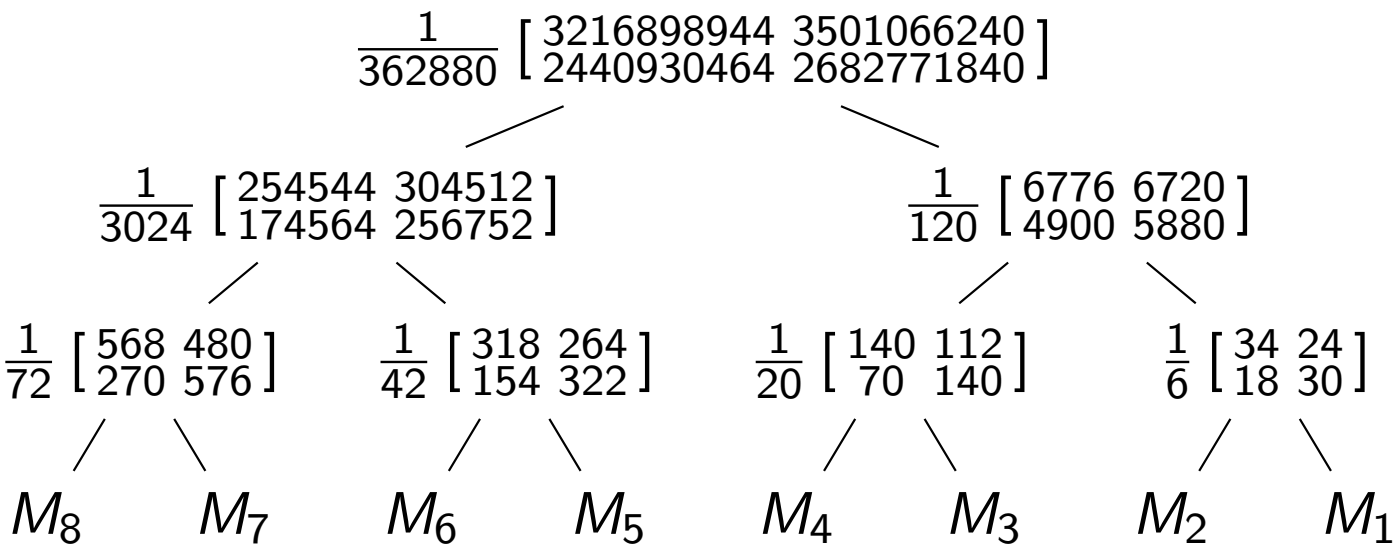
We can read off the bottom of the tree:

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

End of the proof

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

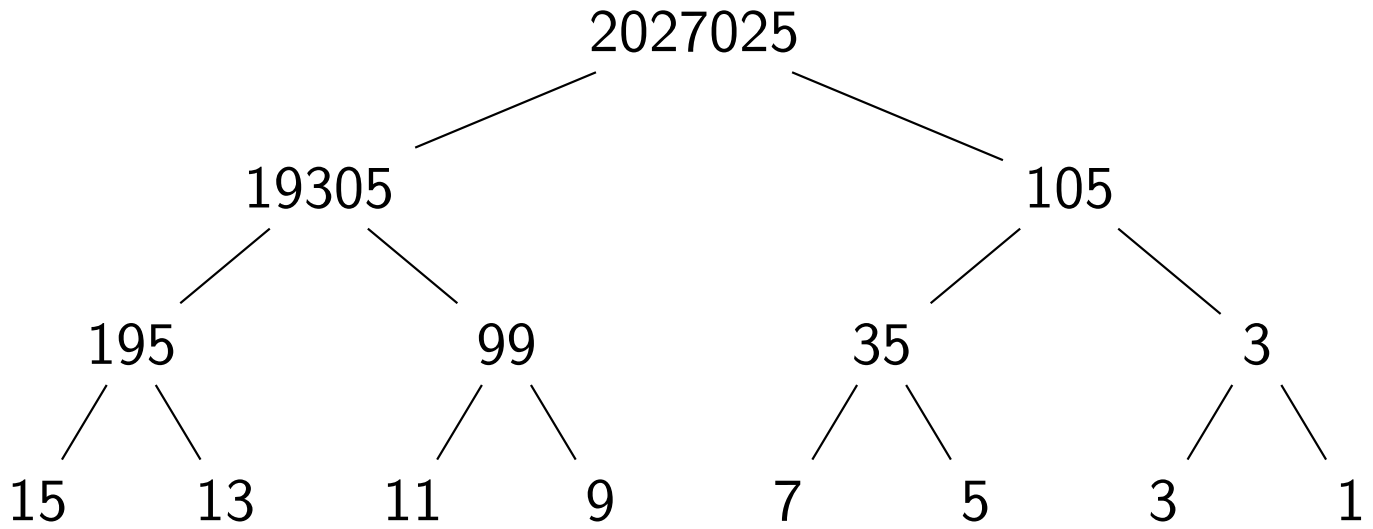
Here is the value tree, for our example $y^2 = x(x^2 + x + 2)$:



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

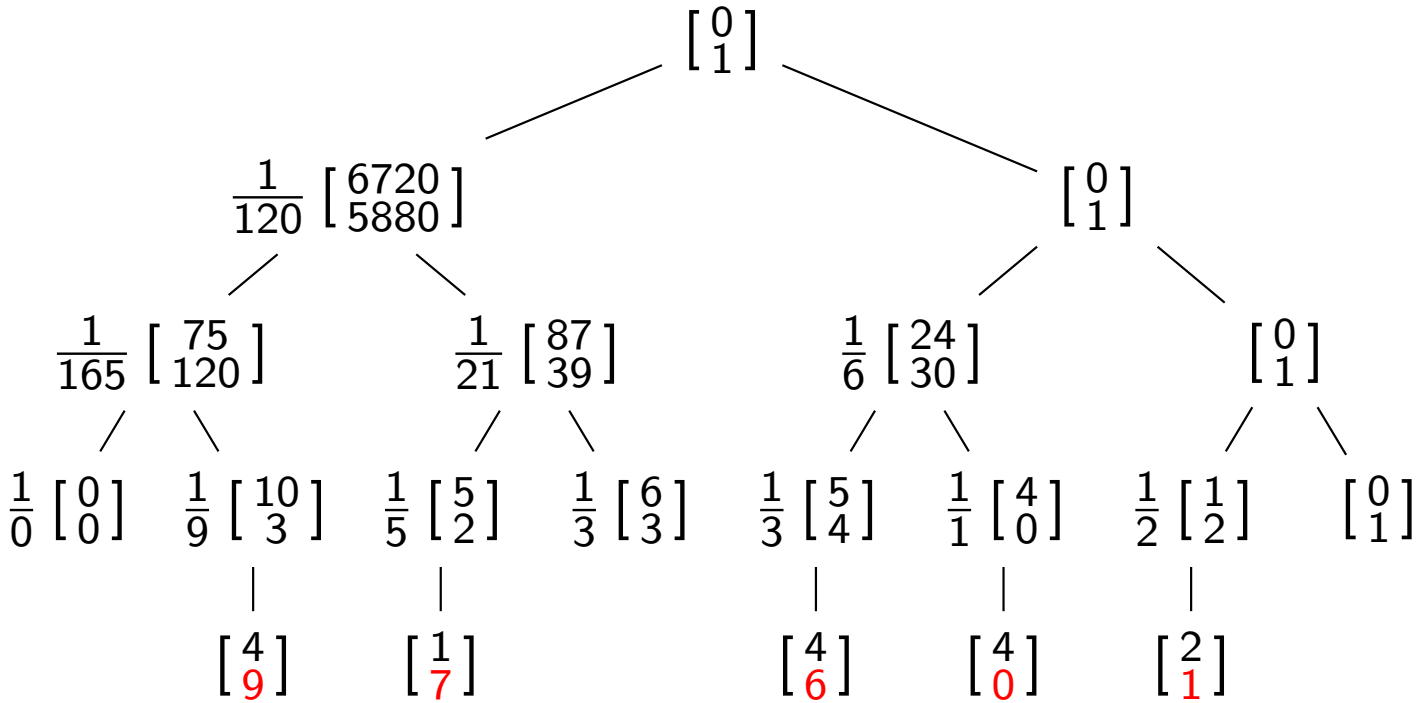
End of the proof

Here is the modulus tree:



End of the proof

And here is the accumulating remainder tree:



Therefore

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

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

End of the proof

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

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

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.

Generalisations

What about a smooth projective curve X of genus $g > 1$?

There exist analogues of the Birch–Swinnerton-Dyer and Sato–Tate conjectures for these curves.

Much less is known about these cases, partly because the existing algorithms do not make it easy to generate large amounts of data, which are needed to formulate and test conjectures!

Generalisations

One possible question is to compute $\#X(\mathbf{F}_p)$.

More generally we may ask to compute the zeta function of X,

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

a generating function that encodes the number of solutions over all finite extensions of $\mathbf{F}_p$.

In fact $Z_X(T)$ is a rational function of the form

$$Z_X(T) = \frac{F_X(T)}{(1-T)(1-pT)},$$

where $F_X(T) \in \mathbf{Z}[T]$ is a polynomial of degree $2g$.

Generalisations

$Z_X(T)$ satisfies a certain functional equation, which implies that $F_X(T)$ is determined by $\#X(\mathbf{F}_p), \ldots, \#X(\mathbf{F}_{p^g})$.

Genus 1 (elliptic curves):

$$F_X(T) = 1 - a_1 T + pT^2,$$

where $a_1 = p + 1 - \#X(\mathbf{F}_p)$.

Genus 2:

$$F_X(T) = 1 - a_1 T + a_2 T^2 - pa_1 T^3 + p^2 T^4,$$

where knowledge of $a_1, a_2 \in \mathbf{Z}$ is equivalent to knowledge of $\#X(\mathbf{F}_p), \#X(\mathbf{F}_{p^2})$.

Generalisations

For a curve of genus g, the right generalisation of the central coefficient of $(x^2 + bx + c)^{(p-1)/2}$ is the Hasse–Witt matrix.

This is a $g \times g$ matrix, whose characteristic polynomial is exactly $F_X(T)$ modulo p.

For $y^2 = f(x)$, the matrix entries are the coefficients of x^{pi-j} of $f(x)^{(p-1)/2}$, for $1 \leq i, j \leq g$.

Generalisations

Suppose that X is such a curve, $y^2 = f(x)$, over $\mathbf{Q}$.

Let X_p denote its reduction modulo p.

Using the same technique described in this talk, we can compute the Hasse–Witt matrix, and hence $Z_{X_p}(T)$ modulo p, for all $p < N$.

The complexity is (probably) $g^{5+\varepsilon} N \log^{3+\varepsilon} N$ (H.–Sutherland), i.e. $g^{5+\varepsilon} \log^{4+\varepsilon} N$ per prime.

Note: complexity per prime is polynomial in both g and $\log N$!

By contrast, the appropriate generalisation of Schoof's algorithm has complexity exponential in the genus, e.g. for genus 2 we get $\log^{8+\varepsilon} p$. Never been tried for $g \geq 3$.

Generalisations

Unfortunately in general $Z_X(T)$ modulo p does not determine $Z_X(T)$. The coefficients can be much larger than p.

In genus 1 we have seen that modulo p is enough, for $p \geq 17$.

In genus 2, knowledge of $Z_X(T)$ modulo p leaves $O(1)$ candidates for $Z_X(T)$. These can be checked quickly by performing suitable group operations in the Jacobian of the curve.

Andrew Sutherland has been trying this out for $g = 2$. Currently the new algorithm is winning at around $N = 2^{22}$, compared to his `smalljac` package. This is about one hour of computation. We still expect to push this threshold down considerably.

Generalisations

In genus 3, knowledge of $Z_X(T)$ modulo p leaves $O(p^{1/2})$ candidates for $Z_X(T)$. One can find the right one using $O(p^{1/4})$ Jacobian group operations.

This is exponential time again, but still probably feasible, at least for hyperelliptic curves. We have not tried it yet. We expect this to be much faster than existing algorithms in genus 3.

For nonhyperelliptic genus 3 curves, the same techniques should work, but everything is messier. Currently I need 28×28 matrices. Would be nice to shrink this!

For $g \geq 4$, the exponential time part of the computation is too expensive, and this approach is likely no longer practical.

Generalisations

Nevertheless, a much more general result can be proved, by abandoning the Hasse–Witt matrix in favour of the more high-powered machinery of p-adic cohomology (following Kedlaya):

Theorem (H., 2012)

Let X be a hyperelliptic curve over $\mathbf{Q}$ of genus g, and let $N \geq 1$.

Impose some mild restrictions on the size of the coefficients of the defining polynomial, and on g, relative to N.

Then we may compute $Z_{X_p}(T)$ for all primes $p < N$ of good reduction, simultaneously in

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

bit operations.

Generalisations

Proof: see "Counting points on hyperelliptic curves in average polynomial time" on arXiv.

This works for curves of any genus.

The average time per prime is

$$g^{8+\varepsilon} \log^{4+\varepsilon} N.$$

This is the first algorithm for this problem achieving complexity polynomial in the input size, in this case averaged over primes.

Thank you!