Faster deterministic integer factorisation

David Harvey
(joint work with Edgar Costa, NYU)

University of New South Wales

6th January 2012, Joint Mathematics Meetings, Boston

The integer factorisation problem

Input: a positive integer N.

Output: complete prime factorisation of N.

Example 1: Input is $N = 91$. Output is 7×13.

Example 2 (RSA-768 challenge): Input is $N =$

$$\begin{array}{c}
1230186684530117755130494958384962720772853569595334792197 \\
3224521517264005072636575187452021997864693899564749427740 \\
6384592519255732630345373154826850791702612214291346167042 \\
921431160222124047927473779408066535141959745985690214341 3.
\end{array}$$

Output is

$$\begin{array}{ccc}
334780716989568987860441698 48 & & 3674604366679959042824463379 9 \\
212690817704794983713768568 91 & \times & 6279526322791581643430876426 7 \\
243138898288379387800228761 47 & & 6032283815739666511279233373 4 \\
116525317430877378144679994 89 & & 1714339681027009279873630891 7.
\end{array}$$

The integer factorisation problem

A few factoring algorithms:

- Quadratic sieve (QS). Non-rigorous subexponential time.
- Number field sieve (NFS). Non-rigorous subexponential time. Fastest known algorithm in practice for large N (see previous slide).
- Elliptic curve method (ECM). Subexponential time, complexity depends mainly on smallest prime factor p. Rigorously established, modulo a hard conjecture on distribution of smooth numbers in short intervals.
- Class groups of quadratic forms. Rigorously established subexponential probabilistic (Las Vegas) time bound.
- Shanks: deterministic exponential time bound $O(N^{1/5+o(1)})$, proof conditional on ERH.

Unconditional deterministic bounds

What about rigorously established deterministic time bounds, not conditional on ERH (or anything else)?

Best known bounds have complexity of the shape

$$O(N^{1/4+o(1)}).$$

Unconditional deterministic bounds

Let $M_{\mathrm{int}}(d)$ = cost of multiplying integers with d bits.

Fürer's algorithm: $M_{\mathrm{int}}(d) = O(d \log d \, 2^{\log^* d})$.

Unconditional deterministic factoring complexity bounds:

- Strassen (1976):

$$O(M_{\mathrm{int}}(N^{1/4} \log N) \log N).$$

- Bostan–Gaudry–Schost (2007):

$$O(M_{\mathrm{int}}(N^{1/4} \log N)).$$

- Our result:

$$O\left(M_{\mathrm{int}}\left(\frac{N^{1/4} \log N}{\sqrt{\log \log N}}\right)\right).$$

Unconditional deterministic bounds

Where does the $\sqrt{\log \log N}$ speedup come from?

Last week Andrew Sutherland asked me:

"Does it come from some sort of sieve over primes in some sort of baby-step/giant-step algorithm?"

Yes it does! Nice guess!

Strassen's algorithm

Let $M(d)$ = cost of multiplying polynomials of degree d over in $(\mathbf{Z}/N\mathbf{Z})[x]$.

Assuming $d = O(N)$, we have

$$M(d) = O(M_{\mathrm{int}}(d \log N))$$

using Kronecker substitution.

Strassen's algorithm

Basic idea of Strassen's algorithm:

Assume the simple case $N = pq$ with $p < \sqrt{N}$, $q > \sqrt{N}$.

Put $L = N^{1/4}$ (we'll pretend this an integer).

Let

$$f(x) = (x+1)(x+2)\cdots(x+L).$$

Then

$$(\sqrt{N})! = (L^2)! = f(0)f(L)\cdots f((L-1)L).$$

Compute $f(x)$ in time $O(M(L)\log L)$ using a product tree.

Evaluate $f(x)$ at $0, L, \ldots (L-1)L$ in time $O(M(L)\log L)$ using a remainder tree (i.e. fast multipoint evaluation).

Finally $\gcd((\sqrt{N})!, N) = p$.

The Bostan–Gaudry–Schost algorithm

BGS Lemma 1 (shifting evaluation points):

Let $P(x)$ have degree d. Given as input

$$P(0), P(\beta), \dots, P(d\beta),$$

and assuming certain invertibility conditions (ignored in this talk), we can compute

$$P(\alpha), P(\alpha+\beta), \dots, P(\alpha+d\beta)$$

in time $O(M(d))$.

(Coefficients of $P(x)$ are *not* part of the input!)

Proof: write down Lagrange interpolation formula and reinterpret it as a polynomial multiplication.

The Bostan–Gaudry–Schost algorithm

BGS Lemma 2:

Given values of

$$(x+1)(x+2)\cdots(x+d)$$

at

$$x = 0, \beta, \ldots, d\beta,$$

we can compute values of

$$(x+1)(x+2)\cdots(x+2d)$$

at

$$x = 0, \beta, \ldots, 2d\beta$$

in time $O(M(d))$.

The Bostan–Gaudry–Schost algorithm

Proof:

$$
\begin{tikzcd}[column sep=large, row sep=large]
\boxed{\begin{array}{c}(x+1)\cdots(x+d)\\ \text{at } x = 0, \beta, \ldots, d\beta\end{array}} \arrow[r, bend left, "\text{shift } (d+1)\beta"] \arrow[d, "\text{shift } d"] \arrow[dd, bend right] & \boxed{\begin{array}{c}(x+1)\cdots(x+d)\\ \text{at } x = (d+1)\beta, \ldots, (2d+1)\beta\end{array}} \arrow[d, "\text{shift } d"'] \arrow[dd, bend left] \\
\boxed{\begin{array}{c}(x+d+1)\cdots(x+2d)\\ \text{at } x = 0, \beta, \ldots, d\beta\end{array}} \arrow[d, "\text{multiply}"'] & \boxed{\begin{array}{c}(x+d+1)\cdots(x+2d)\\ \text{at } x = (d+1)\beta, \ldots, (2d+1)\beta\end{array}} \arrow[d, "\text{multiply}"] \\
\boxed{\begin{array}{c}(x+1)\cdots(x+2d)\\ \text{at } x = 0, \beta, \ldots, d\beta\end{array}} & \boxed{\begin{array}{c}(x+1)\cdots(x+2d)\\ \text{at } x = (d+1)\beta, \ldots, (2d+1)\beta\end{array}}
\end{tikzcd}
$$

The Bostan–Gaudry–Schost algorithm

Main BGS algorithm:

First evaluate $(x+1)$ at $x = 0, L$.

Next evaluate $(x+1)(x+2)$ at $x = 0, L, 2L$.

Next evaluate $(x+1)(x+2)(x+3)(x+4)$ at $x = 0, L, 2L, 3L, 4L$.

...

Finally evaluate $(x+1)\cdots(x+L)$ at $x = 0, L, \ldots, (L-1)L$.

Total time is $O(M(1) + M(2) + M(4) + \cdots + M(L)) = O(M(L))$.

This improves on Strassen by $O(\log L) = O(\log N)$.

New algorithm

Observation: if N is even then it's easy to find a factor!

If N is composite and odd, it must have an *odd* factor.

Instead of taking GCD with

$$1 \times 2 \times 3 \times \cdots \times \sqrt{N},$$

we should take GCD with

$$1 \times 3 \times 5 \times \cdots \times \sqrt{N}.$$

We apply the BGS algorithm to the polynomial

$$f(x) = (x+1)(x+3)\cdots(x+2L+1)$$

where $L = N^{1/4}/\sqrt{2}$. Immediately this saves a factor of $\sqrt{2}$.

New algorithm

Why stop at 2?

Suppose N composite and not divisible by 2 or 3.

We should then use

$$\begin{aligned} f(x) &= (x+1)(x+5)(x+7)(x+11)\cdots(x+6L+1)(x+6L+5) \\ &= h(x)h(x+6)\cdots h(x+6L) \end{aligned}$$

where $h(x) = (x+1)(x+5)$.

It is straightforward to generalise the BGS algorithm to the case where $h(x)$ has degree > 1.

The overall speedup is $\sqrt{\frac{6}{2}} = \sqrt{3}$.

New algorithm

Why stop at 3?

Suppose N composite and not divisible by any primes $< B$.

Let $Q = \prod_{p<B} p$, and apply generalised BGS algorithm to

$$h(x) = \prod_{\substack{j=1 \\ (j,Q)=1}}^{Q} (x+j).$$

Overall speedup is

$$\prod_{p<B} \sqrt{\frac{p}{p-1}} = \Theta(\sqrt{\log B})$$

by Mertens' theorem.

New algorithm

We can't choose B arbitrarily large, since we need time to compute $h(x)$, whose degree is $\phi(Q)$, which is exponential in B.

It is safe to take say $Q \approx N^{1/8}$, so $B \approx \frac{1}{8} \log N$.

Then the overall speedup is

$$\Theta(\sqrt{\log \log N}).$$

Thank you!