May 31: Integer Factorization: "The Problem that Needs No Introduction"

Why care? Such a fundamental problem....
"Dignity of Science" – Gauss
RSA, Rabin, ...

Throughout, we will work w/ composite $N = pq$ ($p < q$ prime)
↳ Easily generalizes to other cases ($n = \log_2 N$ ← input length)
↳ Assume $p \approx q \approx \sqrt{N}$.

N.B: $O(N)$ is exponential in input len $= O(2^n)$; $O(\text{poly}\log(N))$ is poly. in input

First algorithm: Trial Division

1) If we can find the second-largest factor of N, can factor.
2) We know second-largest has size $< \sqrt{N}$.

Alg

Recall Euclids algorithm ("first alg")

Thm Given a, b, can find $\gcd(a,b)$ in time $\text{poly}(|a|, |b|)$.

⇒ You can test whether p is a factor of N in time $\text{poly}(|p|, |N|)$.

Compute $\gcd(p, N)$, check is $\neq 1$.

Trial Division

Input: $N = pq$
Output: factor p of N.

For $i = 1, \ldots, \sqrt{N}$:

IF $\{\gcd(p, N) \neq 1\}$

return p

$\}$

Trial Division

Running time: $O(\sqrt{N})$ gcd calculations

Input len: $n = \log N \Rightarrow O(2^{n/2})$ $\Rightarrow$ "exponential time"

N.B. Can compute gcd w/ only primes to get a $\approx$ factor of $\log N$ speedup.

N.B. There is a variant of Pollard's "rho" algorithm that factors (heuristically) in $O(N^{1/4})$ time ← still exponential, but much better than trial division!

Pollard's $p-1$ Method (1974?)

Let's say you are lucky (or: unlucky) and you get $N = pq$ where $p = p_1 p_2 p_3 \cdots p_k + 1$ for p_i "small" $< B$.

Then there is a subexponential alg that factors N.

Alg:

$a \xleftarrow{R} \mathbb{Z}_N$

Check that a is not a factor of N.

Let $E \leftarrow \prod_i p_i^{e_i}$ (where e_i is large enough s.t. $p_i^{e_i} > \sqrt{N}$).

$b \leftarrow a^E \bmod N$

If $\gcd(b-1, N) \neq 1$, return factor,
Else repeat

Pollard's p-1 Method

(Correctness)

Analysis: Recall the C.R.T. Whenever we work mod $N = pq$, can work mod p and mod q.

$$b \leftarrow a^E \bmod N$$

$b_p \leftarrow a^E \bmod p$

BUT $E \equiv (p-1)E'$, so...

$b_p = a^{(p-1)E'} \bmod p$

$= 1^{E'} \bmod p$

$= 1 \bmod p$

$b_p - 1 = 0 \bmod p$

$b_q \leftarrow a^E \bmod q$

$= a^{E \pmod{q-1}} \bmod q$

↳ Probability that $E \pmod{q-1} = 0$ is "small," so very likely that $b_q \neq 1 \bmod q$

$b_q - 1 \neq 0 \bmod q$

$(b-1) \mid p$ AND $(b-1) \nmid q$.

So $\gcd(b-1, N)$ yields factor of N.

Fermat's Little Thm

p prime

$\gcd(a, p) = 1$

$a^{p-1} = 1 \bmod p$

Running time:

We are doing B exponentiations mod N.

If $N = pq$ and $p = 2 \cdot 3 \cdot 5 \cdot 7 \cdots 121 + 1$, you're in trouble!

Pollard's $p-1$ Method

Is your RSA modulus in danger? What is $\Pr_{\text{RSA mod } N}[\text{pollard eff factors } N]$?

We say that $p-1$ is "B-smooth" if all prime divisors of $p-1$ are $\leq B$.

$\Rightarrow$ Roughly, a random integer $\{1, \ldots, N\}$ is B-smooth w.p. $\approx u^{-u}$, $u = \frac{\ln N}{\ln B}$.

So, if we want $p-1$ to run in ppt for random RSA mod N, we want $B = \text{poly}(\log N)$

$$u = \frac{\log N}{\log(\text{poly}\log(N))} \quad \text{say} \quad u = \frac{\log N}{\log\log N}$$

Then $\Pr_N[\text{factor } N \text{ efficiently}] = \left(\frac{\log N}{\log\log N}\right)^{-\frac{\log N}{\log\log N}} = \left(\frac{n}{\log n}\right)^{-\frac{n}{\log N}}$

Basically inverse exponential in n.

$\downarrow$

Pollard's $p-1$ alg factors almost no RSA moduli in ppt!

However! Lentra's ECM factors all RSA moduli in time $L[\cdot, \cdot]$

Idea: "Randomize" $p-1$. Eventually "$p-1$" will be smooth and you can factor N.

$\downarrow$

ECM is good at finding small factors of N when they exist.

Dixon's Algorithm (1981)

Factors all N in "subexponential time." Requires no unproved assumptions but is not as fast as GNFS, which does use assumptions.

Running time uses notation $L_x[\alpha, \beta] = e^{\beta(\log N)^\alpha (\log\log N)^{1-\alpha}}$.

$L[1, 1] = e^{\log N} = N$ ← exponential time

$L[0, 1] = e^{\log\log N} = \log N$ ← polynomial time

$L[\frac{1}{2}, 1] = e^{\sqrt{\log N}\sqrt{\log\log N}}$ ← "subexponential time"

Best general-purpose factoring alg runs in

$L[\frac{1}{3}, \,] \approx e^{\tilde{O}(\sqrt[3]{\log N})}$ time

Dixon's alg runs in

$L[\frac{1}{2}, \,] \approx e^{\tilde{O}(\sqrt{\log N})}$ time

↖ Constant "in the exponent" matters in practice!

Dixon's alg is interesting b/c

1) Simple
2) "Index calculus" style alg (also useful for dlog)
3) No assumptions needed / No heuristic

Before getting to Dixon's you should know

- There are pdy time fact algs for very special N ($N = p^k q$ for large k) (Boneh, Durfee, N-G '99)
- " " " when half of bits of p known (Coppersmith '96)
- Alg runs in time $e^{O(\sqrt{\log p})}$ for p being smallest factor of N.
  ↳ Contrast w/ Dixon & NFS

Dixon's Alg

Recall from discussion of Rabin's OWF, if you can find a, b,

$$a \neq b \quad \text{s.t.} \quad a^2 = b^2 \pmod{N}$$

$$a \neq \pm b \pmod{N}$$

Then $a^2 - b^2 = 0 \pmod{N}$

$$(a+b)(a-b) = 0 \pmod{N}$$

$\Rightarrow \gcd(a-b, N)$ reveals a factor of N.

Many factoring algs use this difference of sqs. trick!

Question: How do you find a and b?

Dixons Alg

Let $P = \{p_1, \ldots, p_k\}$ be the set of all primes $\leq B$. ← Chosen later

Input: $N = pq$
Output: (p, q)

Alg: 1) Repeat $k+1$ times:

$x_i \xleftarrow{R} \mathbb{Z}_N$

$y_i \leftarrow x_i^2 \pmod N$

if y_i is not B-smooth, throw away
else: write y_i in factored form

$$y_i = p_1^{e_{i1}} p_2^{e_{i2}} p_3^{e_{i3}} \cdots p_k^{e_{ik}}.$$

store $(y_i, \vec{e}_i)$ "relation" in table

2) Once you collect $k+1$ vectors $\vec{e}_i$, you are guaranteed to have a linear dependency among them (mod 2)

Non-zero linear combination of $\vec{e}_i$ s.t. $\sum \vec{e}_i = \vec{0}$.

$\vec{e}_1 = (1\ 0\ 1)$
$\vec{e}_2 = (0\ 1\ 1)$
$\vec{e}_4 = (1\ 1\ 0)$

$\vec{e}_1 \oplus \vec{e}_2 \oplus \vec{e}_4 = (0\ 0\ 0)$

Call the set of dependent vectors S.

3) Write

$$\prod_{i \in S} y_i = \prod_{i \in S} p_1^{e_{i1}} p_2^{e_{i2}} \cdots p_k^{e_{ik}} \pmod N$$

$$\left(\prod_{i \in S} x_i\right)^2 = \left(p_1^{\hat{e}_1} p_2^{\hat{e}_2} \cdots p_k^{\hat{e}_k}\right)^2 \pmod N$$

We found a, b s.t. $a^2 = b^2 \pmod N$.

Now, we hope $a \neq \pm b \bmod N$

$\Rightarrow$ Factored N!

Dixon's Alg

$B = u^u$ for $u = \frac{\ln N}{\ln B}$

Say we pick $B = L[\frac{1}{2}, 1] = e^{\sqrt{\ln N}\sqrt{\ln\ln N}}$

Then $u^u = \frac{\ln N}{\ln B}$

$$= \frac{\ln N}{\sqrt{\ln N}\sqrt{\ln\ln N}}$$

$$u = \sqrt{\frac{\ln N}{\ln\ln N}}$$

$$= e^{\frac{1}{2}\ln\ln N - \frac{1}{2}\ln\ln\ln N}$$

Then $u^u = \left(e^{\frac{1}{2}\ln\ln N}\right)^{\sqrt{\frac{\ln N}{\ln\ln N}}}$

$$B \approx u^u \approx e^{\frac{1}{2}\sqrt{\ln N}\sqrt{\ln\ln N}}$$

Running time is then $O(\text{poly}(B))$. Say $O(B^\alpha)$.

$$O(B^\alpha) = e^{\frac{\alpha}{2}\sqrt{\ln N}\sqrt{\ln\ln N}} = L[\tfrac{1}{2}, \tfrac{\alpha}{2}]$$

↖ Just changes "constant in exponent"

In practice, need to optimize carefully!

Analysis: [Very loose!]

Need to argue that last step succeeds often:

↳ Look at last relation. Choose indep of all others.
Probability that you hit a "bad" relation is $\frac{1}{a}$.

Complexity

We need $\sim B$ relations to factor.

Generating each relation requires finding a B-smooth #.
Heuristically:

$$\Pr[\text{relation is "good"}] \approx u^{-u} \quad \text{for} \quad u = \frac{\log N}{\log B}$$

Checking whether relation is good takes $\sim B$ time.

$$E[\text{time to find relation}] = B \cdot u^u$$

$$E[\text{time to find } B \text{ relations}] = B^2 u^u$$

Time for step 3: Can find dependency using Gaussian elimination: $O(B^3)$ arithmetic ops.

$$\text{Total time: } \underbrace{B^2 u^u}_{\text{step 1}} + \underbrace{B^3}_{\text{step 2}} + \underbrace{\text{Cost of finding } B \text{ primes}}_{\text{↳ ignore (precompute)}}$$

To minimize running time

$$B^2 u^u = B^3$$

$$u^u = B$$