Generating random primes faster

D. J. Bernstein

---

pqRSA project team:

Daniel J. Bernstein
Josh Fried
Nadia Heninger
Paul Lou
Luke Valenta

cr.yp.to/papers.html#pqrsa

The standard algorithm
to generate random primes:

```
proof.arithmetic(False)
while True:
  p = randrange(2^(n-1),2^n)
  p = ZZ(p)
  if p.is_prime(): print p
```

$n^{1+o(1)}$ iterations per prime.

The standard algorithm
to generate random primes:

```
proof.arithmetic(False)
while True:
  p = randrange(2^(n-1),2^n)
  p = ZZ(p)
  if p.is_prime(): print p
```

$n^{1+o(1)}$ iterations per prime.

Standard speedup using wheels:
e.g., force $p \bmod 6 \in \{1,5\}$.
Wheel using all primes $q \le n^{O(1)}$:
$n^{1+o(1)}$ iterations per prime.
Recall $\prod_{q\le y}\left(1-\frac{1}{q}\right) \in \Theta\left(\frac{1}{\log y}\right)$.

2007 Mihailescu: conjecturally
$n^{3+o(1)}$ bit ops to prove p prime.

2007 Mihailescu: conjecturally $n^{3+o(1)}$ bit ops to prove p prime.

2010 Bernstein conjecture: correctly recognize primality using $n^{o(1)}$ tests, total $n^{2+o(1)}$ bit ops. Fermat test, then Lucas test (as in 1980 Baillie–Wagstaff, 1980 Pomerance–Selfridge–Wagstaff), then cubic test (1995 Atkin), etc.; or some elliptic-curve tests.

2007 Mihailescu: conjecturally $n^{3+o(1)}$ bit ops to prove p prime.

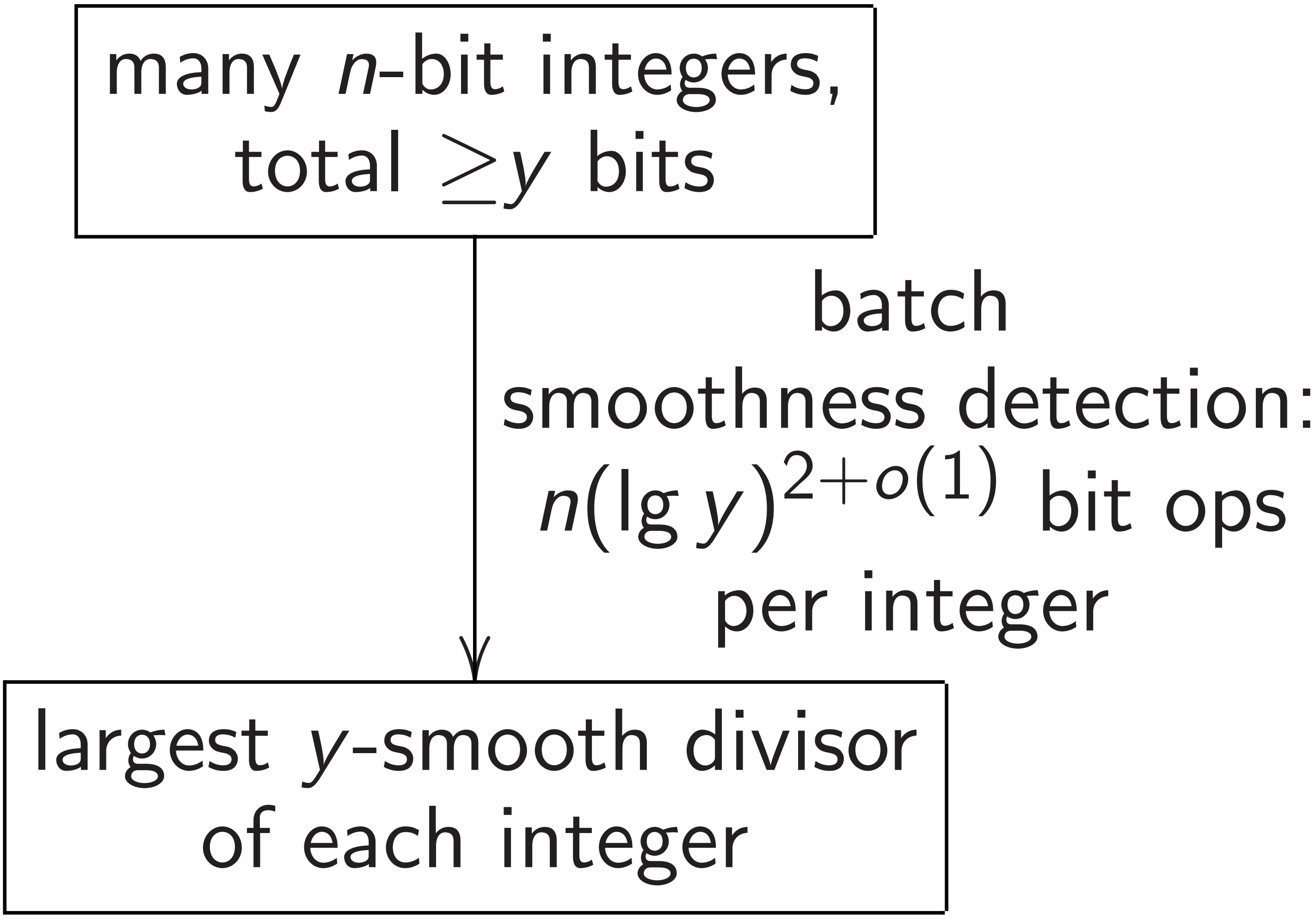
2010 Bernstein conjecture: correctly recognize primality using $n^{o(1)}$ tests, total $n^{2+o(1)}$ bit ops. Fermat test, then Lucas test (as in 1980 Baillie–Wagstaff, 1980 Pomerance–Selfridge–Wagstaff), then cubic test (1995 Atkin), etc.; or some elliptic-curve tests.

Most iterations are much simpler: Fermat test rejects p. Fast reject by trial division/ECM? Still $n^{3+o(1)}$ bit ops per prime.

New: $n^{2.5+o(1)}$ bit ops per prime to generate $2^{n^{0.5+o(1)}}$ primes.

New: $n^{2.5+o(1)}$ bit ops per prime to generate $2^{n^{0.5+o(1)}}$ primes.

Recall:



New: $n^{2.5+o(1)}$ bit ops per prime to generate $2^{n^{0.5+o(1)}}$ primes.

Recall:

many n-bit integers, total $\geq y$ bits

↓ batch smoothness detection: $n(\lg y)^{2+o(1)}$ bit ops per integer

largest y-smooth divisor of each integer

Apply batch smoothness detection for $y = 2^{2^0}$, then $y = 2^{2^1}$, then $y = 2^{2^2}$, ..., then $y \approx 2^{n^{0.5+o(1)}}$.