Quadratic Sieve Algorithm

a.k.a. the Second Fastest Algorithm in the West

Nils Olsson

May 5th, 2021

The quadratic sieve (QS) algorithm…

- Invented by Carl Pomerance in 1981 (improvement to Schroeppel’s linear sieve)
- In practice is the second fastest integer factorization algorithm (after the general number field sieve)
- Still fastest for integers over 100 decimal digits
- **Fundamentally works like (random) square factoring**

(Random) Square Factoring

Fix n an integer, and suppose we have integers x and y such that:

$$x \not\equiv \pm y \pmod{n}, \quad \text{but } x^2 \equiv y^2 \pmod{n}.$$

Then n divides $(x-y)(x+y) = x^2 - y^2$, but not $(x-y)$ or $(x+y)$ alone.

This means $\mathbf{gcd}(x \pm y, n)$ gives“non-trivial” (not ± 1 or $\pm n$) factors of n.

(This is known as Fermat’s factorization method.)

Fix n an integer, and suppose we have integers x and y such that:

$$x \not\equiv \pm y \pmod{n}, \quad \text{but } x^2 \equiv y^2 \pmod{n}.$$

Then n divides $(x-y)(x+y) = x^2 - y^2$, but not $(x-y)$ or $(x+y)$ alone.

This means $\gcd(x \pm y, n)$ gives"non-trivial" (not ± 1 or $\pm n$) factors of n.

(This is known as Fermat's factorization method.)

The question: *how to we find x and y?*

(Random) Square Factoring

(Attempt 0) **Pick randomly (a.k.a. cross your fingers)**

1. Pick $x \in \mathbb{Z}$ (randomly).
2. If $x^2 \pmod{n}$ is a perfect square, 🤙

(Random) Square Factoring

(Attempt 0) **Pick randomly (a.k.a. cross your fingers)**

1. Pick $x \in \mathbb{Z}$ (randomly).
2. If $x^2 \pmod{n}$ is a perfect square, 🤙

(e.g.) With $n = 5959$, we randomly 😉 picked $x = 80$. We find

- $80^2 = 6400 \equiv 441 = 21^2 \pmod{5959}$
- $\gcd(80 \pm 21, 5959) = 59$ and 101, two non-trivial factors of 5959.

(Random) Square Factoring

(Attempt 0) **Pick randomly (a.k.a. cross your fingers)**

1. Pick $x \in \mathbb{Z}$ (randomly).
2. If $x^2 \pmod{n}$ is a perfect square, 🤙

(e.g.) With $n = 5959$, we randomly 😉 picked $x = 80$. We find

- $80^2 = 6400 \equiv 441 = 21^2 \pmod{5959}$
- $\gcd(80 \pm 21, 5959) = 59$ and 101, two non-trivial factors of 5959.

(This might be slower than trial division)

For $x = 2, \ldots, \sqrt{n}$, check if x divides n (keep dividing, collect powers, etc.)

(Random) Square Factoring

(Attempt 1) **Dixon's method**

Select a subset of primes $\mathcal{B} = \{p_1, p_2, \ldots, p_t\}$ called your *factor base*.

Any integer x_i that can be written as $x_i = \prod_j p_j^{e_{ij}}$ for $p_i \in \mathcal{B}$ (or at least with $\max(e_{ij}) \leq \max(p_j)$) is called p_t-smooth.

(Attempt 1) **Dixon's method**

Select a subset of primes $\mathcal{B} = \{p_1, p_2, \ldots, p_t\}$ called your *factor base*.

Any integer x_i that can be written as $x_i = \prod_j p_j^{e_{ij}}$ for $p_i \in \mathcal{B}$ (or at least with $\max(e_{ij}) \leq \max(p_j)$) is called p_t-smooth.

We want to find pairs (x_i, y_i) satisfying two conditions:

1. $x_i^2 \equiv y_i \pmod{n}$ (*quadratic residue*), and
2. $y_i = \prod_j p_j^{e_{ij}}$ (y_i is p_t-smooth)

(Random) Square Factoring

(Attempt 1) **Dixon's method** (continued)

Let $\vec{e}_i = (e_{i1}, e_{i2}, \ldots)$ and $\vec{v}_i = (e_{i1} \bmod 2, \ldots)$ be vectors of its exponents.

If a sum of some of these binary vectors is the zero vector, then from their corresponding x_i and y_i's we can construct an x and a p_t-smooth y such that

$$x \not\equiv y \pmod{n}, \quad \text{but } x^2 \not\equiv y^2 \pmod{n}. \quad ✅$$

Review (Quadratic Residue)

Fix $n \in \mathbb{Z}$ and $a \in \mathbb{Z}_n^*$.

If there exists exists an $x \in \mathbb{Z}_n^*$ such that $x^2 \equiv a \pmod{n}$

- a is a *quadratic residue/square* modulo n; if there is no such x, then
- a is a *quadratic non-residue* modulo n.

(e.g.) fix $a = 2$

With $n = 5$

$$2^2 = 4 \qquad \not\equiv 2 \pmod{5}$$
$$3^2 = 9 \equiv 4 \not\equiv 2 \pmod{5}$$
$$4^2 = 16 \equiv 1 \not\equiv 2 \pmod{5}$$

2 is *not* a non-residue modulo 5.

With $n = 7$

$$2^2 = 4 \not\equiv 2 \pmod{7}$$
$$3^2 = 9 \equiv 2 \pmod{7}$$

2 *is* a quadratic residue modulo 7.

(Attempt 1) **Dixon's method** (continued)

If we construct a matrix V from the v_i's, we're looking for linearly dependent row-index subsets.

(Note we have a column for each p_j of the factor base $\mathcal{B}$.)

To guarantee linear dependence, we collect $|\mathcal{B}| + 1$ such (x_i, y_i) pairs. Using linear algebra we find those subsets.

(Random) Square Factoring

(Attempt 1) **Dixon's method** (continued)

If we construct a matrix V from the v_i's, we're looking for linearly dependent row-index subsets.

(Note we have a column for each p_j of the factor base $\mathcal{B}$.)

To guarantee linear dependence, we collect $|\mathcal{B}| + 1$ such (x_i, y_i) pairs. Using linear algebra we find those subsets.

If for our constructed x and y we have $x \not\equiv \pm y$, then for n we have found non-trivial factors $\mathbf{gcd}(x \pm y, n)$.

(Random) Square Factoring

(Attempt 1) **Dixon's method** (continued)

If we construct a matrix V from the v_i's, we're looking for linearly dependent row-index subsets.

(Note we have a column for each p_j of the factor base $\mathcal{B}$.)

To guarantee linear dependence, we collect $|\mathcal{B}| + 1$ such (x_i, y_i) pairs. Using linear algebra we find those subsets.

If for our constructed x and y we have $x \not\equiv \pm y$, then for n we have found non-trivial factors $\mathbf{gcd}(x \pm y, n)$.

But we can do better

(Attempt 2) **Quadratic sieve**

Fix n a composite that is *not a perfect-power*, $m = \lfloor \sqrt{n} \rfloor$, and consider the polynomial

$$\begin{aligned} f(x) &= x^2 - n \\ \Rightarrow f(x+kp) &= x^2 + 2xkp + (kp)^2 - n \\ &= f(x) + 2xkp + (kp)^2 \equiv f(x) \pmod{p}. \end{aligned}$$

(Attempt 2) **Quadratic sieve**

Fix n a composite that is *not a perfect-power*, $m = \lfloor \sqrt{n} \rfloor$, and consider the polynomial

$$\begin{aligned} f(x) &= x^2 - n \\ \Rightarrow f(x+kp) &= x^2 + 2xkp + (kp)^2 - n \\ &= f(x) + 2xkp + (kp)^2 \equiv f(x) \pmod{p}. \end{aligned}$$

We choose $x_i = (x+m)$ and check whether $y_i = f(x_i) = (x+m)^2 - n$ is p_t-smooth.

(Attempt 2) **Quadratic sieve** (continued)

Things to note:

- $x_i^2 = (x+m)^2 \equiv b_i \pmod{n}$,
- if p divides b_i, then $(x+m)^2 \equiv n \pmod{p}$,
- thus n is a modulo p.

(Attempt 2) **Quadratic sieve** (continued)

Things to note:

- $x_i^2 = (x+m)^2 \equiv b_i \pmod{n}$,
- if p divides b_i, then $(x+m)^2 \equiv n \pmod{p}$,
- thus n is a modulo p.

So $\mathcal{B}$ only needs to contain p_j's such that n is quadratic residue modulo p_j.

Square Factoring

(Attempt 2) **Quadratic sieve** (continued)

To check if $y_i = f(x_i)$ is p_t-smooth we *could* use trial division.

Instead we *sieve*

(Attempt 2) **Quadratic sieve** (continued)

To check if $y_i = f(x_i)$ is p_t-smooth we *could* use trial division.

Instead we *sieve*, by...

1. Constructing $Y = \{y_i = (x_i + m)^2 - n\}$ for each x_i in a *sieving interval*
2. Solving for roots (r_1, r_2) of n modulo $p_j \in \mathcal{B}$. If they exist
3. For each $y_i \equiv r \pmod{m}$, dividing y_i by p_j (until no-longer divisible).

(Attempt 2) **Quadratic sieve** (continued)

To check if $y_i = f(x_i)$ is p_t-smooth we *could* use trial division.

Instead we *sieve*, by...

1. Constructing $Y = \{y_i = (x_i + m)^2 - n\}$ for each x_i in a *sieving interval*
2. Solving for roots (r_1, r_2) of n modulo $p_j \in \mathcal{B}$. If they exist
3. For each $y_i \equiv r \pmod{m}$, dividing y_i by p_j (until no-longer divisible).

When complete, if $y_i = 1$ then y_i completely factorable over $\mathcal{B}$.

If we have at least $|\mathcal{B}|$ of these, then we proceed in the same way as Dixon's:

Using linear algebra to find linearly dependent y_i binary exponent vectors.

Implementation

Notes:

- Since n must be a non-perfect-power composite, QS factoring is merely one part of a complete factoring algorithm

Implementation

Notes:

- Since n must be a non-perfect-power composite, QS factoring is merely one part of a complete factoring algorithm
- *Haven't even considered how to choose $\mathcal{B}$, or the sieving interval!*

Notes:

- Since n must be a non-perfect-power composite, QS factoring is merely one part of a complete factoring algorithm
- *Haven't even considered how to choose $\mathcal{B}$, or the sieving interval!*

Notes (continued):

- Many parts that in theory take we for granted:
    - Factoring over the factor base
    - Solving for quadratic roots of n modulo p
    - Solving for linear dependencies
    - Primality testing
    - Perfect-power testing

Implementation

Notes (continued):

- Many parts that in theory take we for granted:
  - Factoring over the factor base
  - Solving for quadratic roots of n modulo p
  - Solving for linear dependencies
  - Primality testing
  - Perfect-power testing
- I used (i.e. implemented)
  - something very simple
  - the Tonelli-Shanks algorithm
  - augmented matrix$\rightarrow$elementary row operations$\rightarrow$ echelon form to find left-nullspace
  - Skipped this!
  - "Detecting Perfect Powers In Essentially Linear Time"

Implementation

(At least) Two implementations I've written in pure Python:

- qs-sieveless.py (the "instructional" version)
- qs-sieving.py

Also planning on implementing in Rust.