# 1 Factorization

- *Algorithm:* Multi Precision Quadratic Sieve (algo. 1)
- *Input:* A number $n$
- *Complexity:* $\mathcal{O}(e^{(1+o(1))\sqrt{\log n \log\log n}})$
- *Data structure compatibility:* N/A
- *Common applications:* Factorization finding, prime judging, cryptographic application (RSA algorithm)

**Problem.** Factorization
Given a large positive interger $n$, find two integers $a$ and $b$ which satisfy $n = ab$

## Description

Factorization is to decompose an interger into the product of two other intergers. This mathematical operatin is very important in pratice, and its usage is not just limited in the realm of mathematics. For example, factorization is tightly concerned with cryptography. The famous RSA cryptography algorithm is based on multiplication of large prime numbers, so the security of it is associated with the efficiency of factorization.

Since factorizing an even number by 2 is very intuitive, here only the case of factorizing an odd number is considered.

The most naive method of factorization can be the trial-division method, which test all prime numbers smaller than the square root of given $n$. However this can be time-consuming, not only because the number of primes to be tested, but also due to the time consumption in arithmetic operation when operands are large.

In $17_{th}$ century, the french mathematician Pierre de Fermat introduced a quadratic method which finds the following decomposition of a nubmer n

$$n = c^2 - d^2$$

Instead of direct factorization. It is obvious that if such $c, d$ exist, then $n$ can be represented as $(c+d)(c-d)$. Conversely, if factorization $n = ab$ exists, $n$ can also be written as

$$n = (\frac{a+b}{2})^2 - (\frac{a-b}{2})^2$$

Thus the correctness of Fermat's method is proved. However, due to the operations in finding a including calculating squares and detecting a square number, the worst case time complexity of Fermat's method is even larger than that of trial division.

So the quadratic sieving algorithm is introduce as an optimization of Fermat's method. The mechanism of traversing numbers $c$ and calculating $c^2 - n$ is the same as Fermat's method, but it does not search for a square result, and instead it detects if some previous results can have a product as a square number.

Taking factorizing 1649 as an example. In Fermat's method, it can be written as $1649 = 57^2 - 40^2$. Because the searching for $c$ always starts at $\lceil\sqrt{1649}\rceil = 41$, it means that 17 judges should be done before the true answer is found. But in quadratic siecing, only three needs to be calculated.

$$41^2 \equiv 32 \pmod{1649}$$

$$42^2 \equiv 115 \pmod{1649}$$

$$43^2 \equiv 200 \pmod{1649}$$

Because $32 \times 200 = 6400 = 80^2$, it can be seen that $(41 \times 43)^2 \equiv (114)^2 \equiv 80^2 \pmod{1649}$, then the factorization can be deduced.

Now the main strategy is set, the problem to solve is how to find a subset of the previous results which have a square product.

The finding process is associated with a crucial lemma, which states that if a sequence only contains integers which can be represented as product of the $k$ prime numbers and the length of the sequence is larger than $k$, then there must exist a subsequence whose product of elements is a square number.

So the strategy is to select a proper number (denoted as $k$) of primes (denoted as $p_1, p_2, ...p_k$) to which $c^2$ might be congruent. For numbers which are not very large, $k = 4$ is enough. In traversing of $c$ from $\lceil\sqrt{n}\rceil$, $(k+1)$ results which can be represented as product of $p_1, p_2, ...p_k$ will be stored, and now denote them as $c_1, c_2, ...c_{k+1}$. The position of these numbers can be predicted using quadratic congruent function $x^2 - n \equiv 0 \pmod{p_i}$, and it is in fact the "sieving" process.

The process of finding the subset is based on linear algebra. First construct a $k \times (k+1)$ matrix $A$, in which $A_{i,j}$ means the exponential order of $p_i$ in $c_j$. Then the equation system

$$A \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{k+1} \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \pmod{2}$$

is constructed.

When the equation system is solved, then the product $\prod_{i=1}^{k+1} c_i^{s_i}$ will be a proper $c$, and then the number $d$ can be found.

That is all things about quadratic sieving, but it is not enough for leading to a complete implementation. Nowadays, numbers that will be studied in factorization are very large now ( for example, the largest number solved in number factoring challenge issued by RSA consists of 232 decimal digits ). It means that the mathematical algorithm above should find a proper calculating environment which allows the arithmetic operations on such large numbers.

It is why multiprecision should be introduced. The word "multiprecision" means that the precision of arithmetic operation in a computer will only be limited by the available memory in the host system. It surely contradicts with the mechanism of the ALU in the processor which offers a fixed precision arithmetic, and several modern programming languages have added multiprecision as a built-in support or external library.

As for the complexity, for a certain integer $n$, if an optimal time needs to be achieved, the upper bound for the prime numbers should be around

$$exp((2^{-1/2} + o(1))(\log X \log\log X)^{1/2})$$

where $o(1)$ denotes an infinitely small value compared with constant. And considering the searching process, and the calculation, the total time complexity is $\mathcal{O}(e^{(1+o(1))\sqrt{\log n \log\log n}})$. The algorithm is often used when $n$ is about 100 digits long, in which it has the fastest performance.

**Algorithm 1:** Quadratic Sieve

```
Input  : An integer $n$
Output: Two integers $a, b$ which satisfy $n = ab$
k ← 4 ;                              /* set flexibly, needs to be larger if $n$ is large */
root ← $\lceil\sqrt{n}\rceil$;
A ← an $k \times (k+1)$ matrix filled with 0;
P ← {};
i ← 2;
while $len(P) \neq k$ do
    if $i$ is prime and $x^2 - n \equiv 0 \pmod{i}$ has solution then
        $P \leftarrow P + \{i\}$;
    i← i+1;
end while
C ← {};
R ← {};
x ← 0;
while $len(C) \neq k+1$ do
    y← $x^2 - n$;
    if $y$ is product of elements in $P$ then
        $C \leftarrow C + \{y\}$;
        $R \leftarrow R + \{x\}$;
        for $i \leftarrow 1$ to $k$ do
            $A_{i,len(C)} \leftarrow$ order of $p[i]$ in $y$;
        end for
    x← x+1;
end while
S ← solution for equation $AS \equiv \mathbf{0} \pmod 2$;
a ← $gcd(\prod_{i=1}^{k+1} R[i]^{S[i]} - \sqrt{\prod_{i=1}^{k+1} C[i]^{S[i]}}, n)$;
b ← $n/a$;
return a,b
```

## References

- Pomerance, Carl (1996), "A Tale of Two Sieves"
- Pomerance, Carl (1982), "Analysis and Comparison of Some Integer Factoring Algorithms"
- https://en.wikipedia.org/wiki/Quadratic_sieve#cite_ref-1
- http://rosettacode.org/wiki/Arbitrary-precision_integers_(included)