Two examples: factoring and GCDs

The classical computers that exist today are incredibly fast, and their speed seems to be ever increasing. For this reason, some might be inclined to believe that computers are so fast that no computational problem is beyond their reach.

This belief is false. Some computational problems are so inherently complex that, although there exist algorithms to solve them, no computer on the planet Earth today is fast enough to run these algorithms to completion on even moderately sized inputs within the lifetime of a human — or even within the lifetime of the Earth itself.

To explain further, let's introduce the *integer factorization* problem.

> **Integer factorization**
> Input: an integer $N \geq 2$
> Output: the prime factorization of N

By the *prime factorization* of N we mean a list of the prime factors of N and the powers to which they must be raised to obtain N by multiplication. For example, the prime factors of 12 are 2 and 3, and to obtain 12 we must take the product of 2 to the power 2 and 3 to the power 1.

$$12 = 2^2 \cdot 3$$

Up to the ordering of the prime factors, there is only one prime factorization for each positive integer $N \geq 2$, which is a fact known as the *fundamental theorem of arithmetic*.

A few simple code demonstrations in Python will be helpful for further explaining integer factorization and other concepts that relate to this discussion. The following imports are needed for these demonstrations.

```
import math
from sympy.ntheory import factorint
```

The `factorint` function from the `SymPy` symbolic mathematics package for Python solves the integer factorization problem for whatever input N we choose. For example, we can obtain the prime factorization for 12, which naturally agrees with the factorization above.

```
N = 12
print(factorint(N))
```

Output:

```
{2: 2, 3: 1}
```

Factoring small numbers like 12 is easy, but when the number N to be factored gets larger, the problem becomes more difficult. For example, running `factorint` on a significantly larger number causes a short but noticeable delay on a typical personal computer.

```
N = 340282366920938463463374074317682310984309834
print(factorint(N))
```

Output:

```
{3: 2, 74519450661011221: 1, 507372928070793263124358
```

For even larger values of N, things become impossibly difficult, at least as far as we know. For example, the *RSA Factoring Challenge*, which was run by RSA Laboratories from 1991 to 2007, offered a cash prize of $100,000 to factor the following number, which has 309 decimal digits (or 1024 bits when written in binary). The prize for this number was never collected and its prime factors remain unknown.

```
RSA1024 = 135066410865995223349603216278805969938
print(RSA1024)
```

Output:

```
135066410865995223349603216278805969938881475605667027
```

We need not bother running `factorint` on RSA1024, it wouldn't finish within our lifetimes.

The fastest known algorithm for factoring large integers is known as the *number field sieve*. As an example of this algorithm's use, the RSA challenge number RSA250, which has 250 decimal digits (or 829 bits when written in binary), was factored using the number field sieve in 2020. The computation required thousands of CPU core-years, distributed across tens of thousands of machines around the world. Here we can appreciate this effort by checking the solution.

```
RSA250 = 2140324650240744961264423072839333563008
p = 6413528947707158027879019017057738908482501474294
q = 3337202759497815655622601060535511422794076034476
print(RSA250 == p * q)
```

Output:

```
True
```

The security of the RSA public-key cryptosystem is based on the computational difficulty of integer factoring, in the sense that an efficient algorithm for integer factoring would break it.

Next let's consider a related but very different problem, which is computing the greatest common divisor (or GCD) of two integers.

> **Greatest common divisor (GCD)**
> Input: nonnegative integers N and M, at least one of which is positive
> Output: the greatest common divisor of N and M

The greatest common divisor of two numbers is the largest integer that evenly divides both of them.

This problem is easy to solve with a computer — it has roughly the same computational cost as multiplying the two input numbers together. The `gcd` function from the Python `math` module computes the greatest common divisor of numbers that are considerably larger than RSA1024 in the blink of an eye. (In fact, RSA1024 is the GCD of the two numbers in this example.)

```
N = 463675969018391834968223957323668663263635331
M = 505671487480487786422516484397774937475102137
print(math.gcd(N, M))
```

Output:

```
135066410865995223349603216278805969938881475605667027
```

This is possible because we have very efficient algorithms for computing GCDs, the most well-known of which is *Euclid's algorithm*, discovered over 2,000 years ago.

Could there be a fast algorithm for integer factorization that we just haven't discovered yet, allowing large numbers like RSA1024 to be factored in the blink of an eye? The answer is yes. Although we might expect that an efficient algorithm for factoring as simple and elegant as Euclid's algorithm for computing GCDs would have been discovered by now, there is nothing that rules out the existence of a very fast classical algorithm for integer factorization, beyond the fact that we've failed to find one thus far. One could be discovered tomorrow — but don't hold your breath. Generations of mathematicians and computer scientists have searched, and factoring numbers like RSA1024 remains beyond our reach.

Was this page helpful?

Yes No

Report a bug, typo, or request content on GitHub ↗.

Previous page Next page