Factoring Integers

Franklin

Outline

Introduction

Problem

Given a positive integer n, find a nontrivial factor.

A *nontrivial factor* is a factor other than 1 or n.

Introduction

Problem

Given a positive integer n, find a nontrivial factor.

A *nontrivial factor* is a factor other than 1 or n.

If the input size is the number of bits b, there is no known algorithm that runs in polynomial time.

Introduction

Problem

Given a positive integer n, find a nontrivial factor.

A *nontrivial factor* is a factor other than 1 or n.

If the input size is the number of bits b, there is no known algorithm that runs in polynomial time.

For simplicity, we'll often assume that n is composite.

- Checking whether n is prime is easier, and can be done in polynomial time (AKS primality test).

Introduction

Problem

Given a positive integer n, find a nontrivial factor.

A *nontrivial factor* is a factor other than 1 or n.

If the input size is the number of bits b, there is no known algorithm that runs in polynomial time.

For simplicity, we'll often assume that n is composite.

- Checking whether n is prime is easier, and can be done in polynomial time (AKS primality test).

Important for RSA encryption, whose security depends on the difficulty of factoring the product of two primes (a *semiprime*).

Trial Division

The brute force approach: Simply try all integers less than n, checking if each is a factor.

We only need to check up to $\sqrt{n}$; if we haven't found a factor by then, n is prime.

Trial Division

The brute force approach: Simply try all integers less than n, checking if each is a factor.

We only need to check up to $\sqrt{n}$; if we haven't found a factor by then, n is prime.

Unfortunately, $O(\sqrt{n}) = O(\sqrt{2^b}) = O(2^{b/2})$, so the time complexity is still exponential in the number of bits.

Trial Division

The brute force approach: Simply try all integers less than n, checking if each is a factor.

We only need to check up to $\sqrt{n}$; if we haven't found a factor by then, n is prime.

Unfortunately, $O(\sqrt{n}) = O(\sqrt{2^b}) = O(2^{b/2})$, so the time complexity is still exponential in the number of bits.

Considering that there's no known polynomial algorithm, this is not all that bad. But can we do better?

Fermat's Algorithm

Suppose $n = a \cdot b$. Let's assume n is odd, so a and b are odd.

Fermat's Algorithm

Suppose $n = a \cdot b$. Let's assume n is odd, so a and b are odd.

Then write $a = x + y$, $b = x - y$ for some integers x, y, so

$$n = ab = x^2 - y^2 \implies x^2 - n = y^2.$$

Fermat's Algorithm

Suppose $n = a \cdot b$. Let's assume n is odd, so a and b are odd.

Then write $a = x + y$, $b = x - y$ for some integers x, y, so

$$n = ab = x^2 - y^2 \implies x^2 - n = y^2.$$

Now try values for x, starting from $x = \lceil \sqrt{n} \rceil$. For each x, compute $x^2 - n$, and see if the result is a perfect square.

Example

Consider $n = 1081$. Then $\lceil \sqrt{1081} \rceil = 33$, so we try:

Example

Consider $n = 1081$. Then $\lceil \sqrt{1081} \rceil = 33$, so we try:

x	x^2	$y^2 = x^2 - n$
33	1089	8
34	1156	75
35	1225	144

Example

Consider $n = 1081$. Then $\lceil \sqrt{1081} \rceil = 33$, so we try:

x	x^2	$y^2 = x^2 - n$
33	1089	8
34	1156	75
35	1225	144

We find 144 is a perfect square, so $x = 35, y = 12$ works.

Example

Consider $n = 1081$. Then $\lceil \sqrt{1081} \rceil = 33$, so we try:

x	x^2	$y^2 = x^2 - n$
33	1089	8
34	1156	75
35	1225	144

We find 144 is a perfect square, so $x = 35, y = 12$ works.

Then we can recover a, b:

$$a = x + y = 47, \quad b = x - y = 23.$$

Example

Consider $n = 1081$. Then $\lceil \sqrt{1081} \rceil = 33$, so we try:

x	x^2	$y^2 = x^2 - n$
33	1089	8
34	1156	75
35	1225	144

We find 144 is a perfect square, so $x = 35, y = 12$ works.

Then we can recover a, b:

$$a = x + y = 47, \quad b = x - y = 23.$$

As expected, $47 \cdot 23 = 1081$.

Fermat's Algorithm: Analysis

If a factor is near $\sqrt{n}$, the algorithm is fast – x will not be far from $\sqrt{n}$.

Fermat's Algorithm: Analysis

If a factor is near $\sqrt{n}$, the algorithm is fast – x will not be far from $\sqrt{n}$.

But if the factors are far apart (or worse, if n is prime), we could be looking from $\sqrt{n}$ to near $n/2$. In the worst case, that takes $O(n)$ time – even worse than trial division!

Fermat's Algorithm: Analysis

If a factor is near $\sqrt{n}$, the algorithm is fast – x will not be far from $\sqrt{n}$.

But if the factors are far apart (or worse, if n is prime), we could be looking from $\sqrt{n}$ to near $n/2$. In the worst case, that takes $O(n)$ time – even worse than trial division!

We can make some optimizations to skip x that won't work, guess approximately where the primes are and start looking for x around there, or use Fermat's algorithm to narrow the search space for trial division.

Fermat's Algorithm: Analysis

If a factor is near $\sqrt{n}$, the algorithm is fast – x will not be far from $\sqrt{n}$.

But if the factors are far apart (or worse, if n is prime), we could be looking from $\sqrt{n}$ to near $n/2$. In the worst case, that takes $O(n)$ time – even worse than trial division!

We can make some optimizations to skip x that won't work, guess approximately where the primes are and start looking for x around there, or use Fermat's algorithm to narrow the search space for trial division.

But there's much better.

Section 2

Dixon's Algorithm

Modulo

Let's loosen our conditions a bit, and instead look for x, y such that $x^2 - y^2$ is a multiple of n. In other words,

$$x^2 - y^2 \equiv 0 \pmod{n} \implies x^2 \equiv y^2 \pmod{n}.$$

Modulo

Let's loosen our conditions a bit, and instead look for x, y such that $x^2 - y^2$ is a multiple of n. In other words,

$$x^2 - y^2 \equiv 0 \pmod{n} \implies x^2 \equiv y^2 \pmod{n}.$$

We can try to just look for pairs (x, y) satisfying this, but there are a lot of different possible values of x^2 mod n (these remainders are called *quadratic residues*). Specifically, if n is the product of just two primes, there are around $n/4$ quadratic residues mod n.

Modulo

Let's loosen our conditions a bit, and instead look for x, y such that $x^2 - y^2$ is a multiple of n. In other words,

$$x^2 - y^2 \equiv 0 \pmod{n} \implies x^2 \equiv y^2 \pmod{n}.$$

We can try to just look for pairs (x, y) satisfying this, but there are a lot of different possible values of x^2 mod n (these remainders are called *quadratic residues*). Specifically, if n is the product of just two primes, there are around $n/4$ quadratic residues mod n.

We can also try various x, reduce x^2 to its remainder mod n, and see if this result ($< n$) is a perfect square, like in Fermat's algorithm. Unfortunately, there are only $\sqrt{n}$ perfect squares less than n, so it will take a long time to land on one.

Modulo

Let's loosen our conditions a bit, and instead look for x, y such that $x^2 - y^2$ is a multiple of n. In other words,

$$x^2 - y^2 \equiv 0 \pmod{n} \implies x^2 \equiv y^2 \pmod{n}.$$

We can try to just look for pairs (x, y) satisfying this, but there are a lot of different possible values of x^2 mod n (these remainders are called *quadratic residues*). Specifically, if n is the product of just two primes, there are around $n/4$ quadratic residues mod n.

We can also try various x, reduce x^2 to its remainder mod n, and see if this result $(< n)$ is a perfect square, like in Fermat's algorithm. Unfortunately, there are only $\sqrt{n}$ perfect squares less than n, so it will take a long time to land on one.

Combinations

Landing the quadratic residue x^2 mod n on a perfect square is quite hard.
But we don't need to.

Combinations

Landing the quadratic residue $x^2 \bmod n$ on a perfect square is quite hard. But we don't need to.

Consider $n = 84923$. Then, here are some x we might try at some point, with the remainders factorized into primes:

$$513^2 \equiv 8400 = 2^4 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$
$$537^2 \equiv 33600 = 2^6 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$

Combinations

Landing the quadratic residue $x^2 \bmod n$ on a perfect square is quite hard. But we don't need to.

Consider $n = 84923$. Then, here are some x we might try at some point, with the remainders factorized into primes:

$$513^2 \equiv 8400 = 2^4 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$
$$537^2 \equiv 33600 = 2^6 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$

Neither of the residues are perfect squares. However, when multiplied:

$$8400 \cdot 33600 = 2^{10} \cdot 3^2 \cdot 5^4 \cdot 7^2 = (2^5 \cdot 3 \cdot 5^2 \cdot 7)^2 = 16800^2.$$

Combinations

Landing the quadratic residue $x^2 \bmod n$ on a perfect square is quite hard. But we don't need to.

Consider $n = 84923$. Then, here are some x we might try at some point, with the remainders factorized into primes:

$$513^2 \equiv 8400 = 2^4 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$
$$537^2 \equiv 33600 = 2^6 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$

Neither of the residues are perfect squares. However, when multiplied:

$$8400 \cdot 33600 = 2^{10} \cdot 3^2 \cdot 5^4 \cdot 7^2 = (2^5 \cdot 3 \cdot 5^2 \cdot 7)^2 = 16800^2.$$

So we can multiply remainders to get a perfect square!

Combinations

We now have $(513 \cdot 537)^2 \equiv 16800^2 \pmod{84923}$. To recover the original primes, note that $513 \cdot 537 = 275481 \equiv 20712 \pmod{84923}$, so

Combinations

We now have $(513 \cdot 537)^2 \equiv 16800^2 \pmod{84923}$. To recover the original primes, note that $513 \cdot 537 = 275481 \equiv 20712 \pmod{84923}$, so

$$\begin{aligned}(513 \cdot 537)^2 - 16800^2 &\equiv 20712^2 - 16800^2 \\ &\equiv (20712 - 16800)(20712 + 16800) \\ &\equiv 0 \pmod{84923}.\end{aligned}$$

So some divisor of $20712 - 16800$ is a factor of 84923, and same for $20712 + 16800$.

Combinations

We now have $(513 \cdot 537)^2 \equiv 16800^2 \pmod{84923}$. To recover the original primes, note that $513 \cdot 537 = 275481 \equiv 20712 \pmod{84923}$, so

$$\begin{aligned}(513 \cdot 537)^2 - 16800^2 &\equiv 20712^2 - 16800^2 \\ &\equiv (20712 - 16800)(20712 + 16800) \\ &\equiv 0 \pmod{84923}.\end{aligned}$$

So some divisor of $20712 - 16800$ is a factor of 84923, and same for $20712 + 16800$.

Our factors are then

$$\gcd(20712 - 16800, 84923) = 163, \quad \gcd(20712 + 16800, 84923) = 521.$$

Taking this Further

The key idea is that we don't need to hit a quadratic residue that's a perfect square – instead, we can find several residues whose product is a perfect square.

Taking this Further

The key idea is that we don't need to hit a quadratic residue that's a perfect square – instead, we can find several residues whose product is a perfect square.

We could start just squaring numbers above $\sqrt{n}$ and tracking the quadratic residues...

Taking this Further

The key idea is that we don't need to hit a quadratic residue that's a perfect square – instead, we can find several residues whose product is a perfect square.

We could start just squaring numbers above $\sqrt{n}$ and tracking the quadratic residues...

- ...but it's unrealistic to systematically check all the combinations of the residues.

Prime Factorization

Let's return to our earlier example of $n = 84923$. We had:

$$513^2 \equiv 8400 = 2^4 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$
$$537^2 \equiv 33600 = 2^6 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$

Prime Factorization

Let's return to our earlier example of $n = 84923$. We had:

$$513^2 \equiv 8400 = 2^4 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$
$$537^2 \equiv 33600 = 2^6 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$

From the prime factorizations, we can tell the product of these residues will give a square because the resulting exponents will be even.

Prime Factorization

Let's return to our earlier example of $n = 84923$. We had:

$$513^2 \equiv 8400 = 2^4 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$
$$537^2 \equiv 33600 = 2^6 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$

From the prime factorizations, we can tell the product of these residues will give a square because the resulting exponents will be even.

So we can try:

1. Square a bunch of numbers to get some residues.
2. Find the prime factorization of the residues.

Prime Factorization

Let's return to our earlier example of $n = 84923$. We had:

$$513^2 \equiv 8400 = 2^4 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$
$$537^2 \equiv 33600 = 2^6 \cdot 3 \cdot 5^2 \cdot 7 \pmod{84923}$$

From the prime factorizations, we can tell the product of these residues will give a square because the resulting exponents will be even.

So we can try:

1. Square a bunch of numbers to get some residues.
2. Find the prime factorization of the residues.
3. Find some way to multiply them so that the sum of the exponents of each prime in the result is even.

Factor Base

Unfortunately, factoring all the residues and then dealing with all the different primes that can appear in the factorizations is too much.

Factor Base

Unfortunately, factoring all the residues and then dealing with all the different primes that can appear in the factorizations is too much.

Instead:

1. Choose a few small primes up to some bound B – a **factor base**.

Factor Base

Unfortunately, factoring all the residues and then dealing with all the different primes that can appear in the factorizations is too much.

Instead:

1. Choose a few small primes up to some bound B – a **factor base**.
2. Square a bunch of numbers to get some residues.
3. Attempt to factor the residues over the factor base, throwing out any that contain other prime factors.

Factor Base

Unfortunately, factoring all the residues and then dealing with all the different primes that can appear in the factorizations is too much.

Instead:

1. Choose a few small primes up to some bound B – a **factor base**.
2. Square a bunch of numbers to get some residues.
3. Attempt to factor the residues over the factor base, throwing out any that contain other prime factors.

The prime factors of the remaining residues are less than B – we call these residues B**-smooth**. We can then try to combine the residues so that the resulting exponents in the prime factorization are even, which would give us a perfect square.

Example

Suppose $n = 4183$. We'll choose $B = 11$, so our factor base is $\{2, 3, 5, 7, 11\}$. Trying some random numbers not less than $\lceil \sqrt{n} \rceil = 65$, generate 6 B-smooth numbers:

Example

Suppose $n = 4183$. We'll choose $B = 11$, so our factor base is $\{2, 3, 5, 7, 11\}$. Trying some random numbers not less than $\lceil\sqrt{n}\rceil = 65$, generate 6 B-smooth numbers:

x	x^2	$y = x^2 \pmod{n}$	Prime factorization
65	4225	42	$2 \cdot 3 \cdot 7$
82	6724	2541	$3 \cdot 7 \cdot 11$
92	8464	98	$2 \cdot 7^2$
104	10816	2450	$2 \cdot 5^2 \cdot 7^2$
113	12769	220	$2^2 \cdot 5 \cdot 11$
118	13924	1375	$5^3 \cdot 11$

Example

Suppose $n = 4183$. We'll choose $B = 11$, so our factor base is $\{2, 3, 5, 7, 11\}$. Trying some random numbers not less than $\lceil\sqrt{n}\rceil = 65$, generate 6 B-smooth numbers:

x	x^2	$y = x^2 \pmod{n}$	Prime factorization
65	4225	42	$2 \cdot 3 \cdot 7$
82	6724	2541	$3 \cdot 7 \cdot 11$
92	8464	98	$2 \cdot 7^2$
104	10816	2450	$2 \cdot 5^2 \cdot 7^2$
113	12769	220	$2^2 \cdot 5 \cdot 11$
118	13924	1375	$5^3 \cdot 11$

$$\begin{pmatrix} 1 \\ 0 \\ 2 \\ 2 \\ 0 \end{pmatrix}$$

Write the prime factorizations as column vectors, storing the exponent for each prime in the factor base. The vector above corresponds to $y = 2450$.

Example

Multiplying residues is equivalent to adding the exponents in their prime factorizations. So given the 6 column vectors,

$$\begin{pmatrix}1\\1\\0\\1\\0\end{pmatrix} \quad \begin{pmatrix}0\\1\\0\\1\\1\end{pmatrix} \quad \begin{pmatrix}1\\0\\2\\0\\0\end{pmatrix} \quad \begin{pmatrix}1\\0\\2\\2\\0\end{pmatrix} \quad \begin{pmatrix}2\\1\\0\\0\\1\end{pmatrix} \quad \begin{pmatrix}0\\0\\3\\0\\1\end{pmatrix}$$

Example

Multiplying residues is equivalent to adding the exponents in their prime factorizations. So given the 6 column vectors,

$$\begin{pmatrix}1\\1\\0\\1\\0\end{pmatrix} \quad \begin{pmatrix}0\\1\\0\\1\\1\end{pmatrix} \quad \begin{pmatrix}1\\0\\2\\0\\0\end{pmatrix} \quad \begin{pmatrix}1\\0\\2\\2\\0\end{pmatrix} \quad \begin{pmatrix}2\\1\\0\\0\\1\end{pmatrix} \quad \begin{pmatrix}0\\0\\3\\0\\1\end{pmatrix}$$

our goal is to sum some of them to get a vector where every entry is even – in other words, 0 (mod 2).

Linear System

Reduce the entries of the vectors mod 2. Then, we want to find $x_1, \dots, x_6 \in \{0,1\}$ (not all zero) such that, mod 2,

$$x_1 \begin{pmatrix}1\\1\\0\\1\\0\end{pmatrix} + x_2 \begin{pmatrix}0\\1\\0\\1\\1\end{pmatrix} + x_3 \begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix} + x_4 \begin{pmatrix}1\\0\\0\\0\\0\end{pmatrix} + x_5 \begin{pmatrix}0\\1\\0\\0\\1\end{pmatrix} + x_6 \begin{pmatrix}0\\0\\1\\0\\1\end{pmatrix} \equiv \begin{pmatrix}0\\0\\0\\0\\0\end{pmatrix}.$$

Linear System

Reduce the entries of the vectors mod 2. Then, we want to find $x_1, \ldots, x_6 \in \{0, 1\}$ (not all zero) such that, mod 2,

$$x_1 \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} + x_3 \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + x_4 \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + x_5 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} + x_6 \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

From linear algebra, this always has a solution. There are 5 equations and 6 variables – the vectors are *linearly dependent*, meaning there is some solution $(x_1, x_2, x_3, x_4, x_5, x_6) \neq (0, 0, 0, 0, 0, 0)$.

Linear System

Reduce the entries of the vectors mod 2. Then, we want to find $x_1, \ldots, x_6 \in \{0, 1\}$ (not all zero) such that, mod 2,

$$x_1 \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} + x_3 \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + x_4 \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + x_5 \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} + x_6 \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

From linear algebra, this always has a solution. There are 5 equations and 6 variables – the vectors are *linearly dependent*, meaning there is some solution $(x_1, x_2, x_3, x_4, x_5, x_6) \neq (0, 0, 0, 0, 0, 0)$.

In this case, $(0, 0, 1, 1, 0, 0)$ works.

Combining the Residues

Our solution $(0, 0, 1, 1, 0, 0)$ corresponds to the 3rd and 4th rows of our table:

x	x^2	$y = x^2 \pmod{n}$	Prime factorization
92	8464	98	$2 \cdot 7^2$
104	10816	2450	$2 \cdot 5^2 \cdot 7^2$

Combining the Residues

Our solution $(0, 0, 1, 1, 0, 0)$ corresponds to the 3rd and 4th rows of our table:

x	x^2	$y = x^2 \pmod{n}$	Prime factorization
92	8464	98	$2 \cdot 7^2$
104	10816	2450	$2 \cdot 5^2 \cdot 7^2$

Let's multiply these. We have $92 \cdot 104 \equiv 1202 \pmod{4183}$, and

$$92^2 \cdot 104^2 \equiv 98 \cdot 2450 \equiv (2 \cdot 5 \cdot 7^2)^2 \equiv 490^2 \pmod{4183}$$
$$0 \equiv (92 \cdot 104)^2 - 490^2 \equiv (1202 - 490)(1202 + 490) \pmod{4183}$$

Our factors are then

$$a = \gcd(1202 - 490, 4183) = 89, \quad b = \gcd(1202 + 490, 4183) = 41.$$

Dixon's Algorithm

Given an input n:

1. Choose a factor base of size k, containing all primes up to some bound B.

Dixon's Algorithm

Given an input n:

1. Choose a factor base of size k, containing all primes up to some bound B.
2. Generate residues (at least $k + 1$) that factor over the factor base.
   - Repeatedly choose (random) numbers between $\sqrt{n}$ and n and square them.

Dixon's Algorithm

Given an input n:

1. Choose a factor base of size k, containing all primes up to some bound B.
2. Generate residues (at least $k+1$) that factor over the factor base.
   - Repeatedly choose (random) numbers between $\sqrt{n}$ and n and square them.
3. From the prime factorizations, use the exponents to create a system of linear equations mod 2.
4. Find a solution to the system. (e.g., with Gaussian elimination)
5. Multiply the corresponding residues and recover the factors of n.

Analysis

Dixon's algorithm is dominated mainly by:

- Generation of residues (repeated squaring), most of which will be discarded due to not factoring over the factor base
- Attempting to factor the residues over the factor base

Analysis

Dixon's algorithm is dominated mainly by:

- Generation of residues (repeated squaring), most of which will be discarded due to not factoring over the factor base
- Attempting to factor the residues over the factor base

Choosing the correct upper bound B is important – it turns out to be about

$$\exp\left(\sqrt{\log n}\sqrt{\log\log n}/2\right).$$

Analysis

Dixon's algorithm is dominated mainly by:

- Generation of residues (repeated squaring), most of which will be discarded due to not factoring over the factor base
- Attempting to factor the residues over the factor base

Choosing the correct upper bound B is important – it turns out to be about

$$\exp\left(\sqrt{\log n}\sqrt{\log\log n}/2\right).$$

The (expected) runtime of Dixon's algorithm ends up as roughly

$$O\left(\exp\left(2\sqrt{2}\sqrt{\log n}\sqrt{\log\log n}\right)\right).$$

Analysis

Dixon's algorithm is dominated mainly by:

- Generation of residues (repeated squaring), most of which will be discarded due to not factoring over the factor base
- Attempting to factor the residues over the factor base

Choosing the correct upper bound B is important – it turns out to be about

$$\exp\left(\sqrt{\log n}\sqrt{\log\log n}/2\right).$$

The (expected) runtime of Dixon's algorithm ends up as roughly

$$O\left(\exp\left(2\sqrt{2}\sqrt{\log n}\sqrt{\log\log n}\right)\right).$$

This is actually sub-exponential in $b = \log n$, thanks to the square roots.

Section 3

Quadratic Sieve

Improving on Dixon's

Ideas:

1. When generating residues, square values near $\sqrt{n}$. This way, the residue will be small:

$$r(x) = x^2 - n \ll n.$$

Improving on Dixon's

Ideas:

1. When generating residues, square values near $\sqrt{n}$. This way, the residue will be small:

$$r(x) = x^2 - n \ll n.$$

2. Choose the primes in the factor base wisely. In particular, choose p such that

$$r(x) = x^2 - n \equiv 0 \pmod{p}$$

has solutions. We'll see why shortly.

The Sieve

Suppose we have one solution x to $x^2 - n \equiv 0 \pmod{p}$.

The Sieve

Suppose we have one solution x to $x^2 - n \equiv 0 \pmod{p}$. Then note

$$
\begin{aligned}
r(x+kp) &\equiv (x+kp)^2 - n \\
&\equiv x^2 + 2kpx + (kp)^2 - n \\
&\equiv x^2 - n \\
&\equiv 0 \pmod{p}.
\end{aligned}
$$

The Sieve

Suppose we have one solution x to $x^2 - n \equiv 0 \pmod{p}$. Then note

$$\begin{aligned} r(x+kp) &\equiv (x+kp)^2 - n \\ &\equiv x^2 + 2kpx + (kp)^2 - n \\ &\equiv x^2 - n \\ &\equiv 0 \pmod{p}. \end{aligned}$$

Therefore, if x is a solution, $x+kp, x+2kp, \ldots$ are solutions. Each of these give residues that are multiples of p.

The Sieve

Suppose we have one solution x to $x^2 - n \equiv 0 \pmod{p}$. Then note

$$\begin{aligned} r(x + kp) &\equiv (x + kp)^2 - n \\ &\equiv x^2 + 2kpx + (kp)^2 - n \\ &\equiv x^2 - n \\ &\equiv 0 \pmod{p}. \end{aligned}$$

Therefore, if x is a solution, $x + kp, x + 2kp, \ldots$ are solutions. Each of these give residues that are multiples of p.

So by solving $x^2 - n \equiv 0 \pmod{p}$, we get many x that give residues that are multiples of p at once!

The Sieve

Let $x_0 = \lceil \sqrt{n} \rceil$. Say we generate an array of residues of some size:

$$A = \begin{bmatrix} r(x_0) & r(x_0+1) & r(x_0+2) & r(x_0+3) & r(x_0+4) & r(x_0+5) & \cdots \end{bmatrix}$$

The Sieve

Let $x_0 = \lceil \sqrt{n} \rceil$. Say we generate an array of residues of some size:

$$A = \begin{bmatrix} r(x_0) & r(x_0+1) & r(x_0+2) & r(x_0+3) & r(x_0+4) & r(x_0+5) & \cdots \end{bmatrix}$$

Now consider $p = 2$. Find the solution of $r(x) = x^2 - n \equiv 0 \pmod 2$ closest to $\lceil \sqrt{n} \rceil$. Let's say it's x_0 (it's either that or $x_0 + 1$). Then we know 2 divides all of the following:

$$r(x_0), r(x_0+2), r(x_0+4), \ldots$$

The Sieve

Let $x_0 = \lceil\sqrt{n}\rceil$. Say we generate an array of residues of some size:

$$A = \begin{bmatrix} r(x_0) & r(x_0+1) & r(x_0+2) & r(x_0+3) & r(x_0+4) & r(x_0+5) & \cdots \end{bmatrix}$$

Now consider $p = 2$. Find the solution of $r(x) = x^2 - n \equiv 0 \pmod 2$ closest to $\lceil\sqrt{n}\rceil$. Let's say it's x_0 (it's either that or $x_0 + 1$). Then we know 2 divides all of the following:

$$r(x_0), r(x_0+2), r(x_0+4), \ldots$$

So we can update our array as follows:

$$A = \begin{bmatrix} \frac{r(x_0)}{2} & r(x_0+1) & \frac{r(x_0+2)}{2} & r(x_0+3) & \frac{r(x_0+4)}{2} & r(x_0+5) & \cdots \end{bmatrix}$$

The Sieve

Next, repeat for other primes p:

- Solve $x^2 - n \equiv 0 \pmod{p}$ to get a sequence of solutions $x, x+p, x+2p, x+3p, \ldots$.
- Divide the corresponding entries in the array A by p.

The Sieve

Next, repeat for other primes p:

- Solve $x^2 - n \equiv 0 \pmod{p}$ to get a sequence of solutions $x, x+p, x+2p, x+3p, \ldots$.
- Divide the corresponding entries in the array A by p.

This is the "sieve" in the quadratic sieve – since we divide every 2nd element by p, then every 3rd element, then every 5th element, etc. (reminiscent of the Sieve of Eratosthenes).

The Sieve

Next, repeat for other primes p:

- Solve $x^2 - n \equiv 0 \pmod{p}$ to get a sequence of solutions $x, x+p, x+2p, x+3p, \dots$.
- Divide the corresponding entries in the array A by p.

This is the "sieve" in the quadratic sieve – since we divide every 2nd element by p, then every 3rd element, then every 5th element, etc. (reminiscent of the Sieve of Eratosthenes).

Once an entry in A reaches 1, that means the corresponding residue fully factors over all primes p used in the sieving.

The Sieve

Next, repeat for other primes p:

- Solve $x^2 - n \equiv 0 \pmod{p}$ to get a sequence of solutions $x, x+p, x+2p, x+3p, \ldots$.
- Divide the corresponding entries in the array A by p.

This is the "sieve" in the quadratic sieve – since we divide every 2nd element by p, then every 3rd element, then every 5th element, etc. (reminiscent of the Sieve of Eratosthenes).

Once an entry in A reaches 1, that means the corresponding residue fully factors over all primes p used in the sieving.

A solution to $x^2 - n \equiv 0 \pmod{p}$ doesn't actually exist for all p, so we'll have to skip some primes p. The primes that we keep constitute our factor base.

Combining Residues

Once we have enough fully-factoring residues (likely one more than the size of the factor base), we can create the system of equations as in Dixon's algorithm to find the combination of residues that will result in a perfect square.

Combining Residues

Once we have enough fully-factoring residues (likely one more than the size of the factor base), we can create the system of equations as in Dixon's algorithm to find the combination of residues that will result in a perfect square.

Then we can apply difference of squares to get the factors of n, as seen earlier.

Quadratic Sieve Algorithm

Given an input n:

1. Initialize a large array A to hold information about residues for x near $\sqrt{n}$.

Quadratic Sieve Algorithm

Given an input n:

1. Initialize a large array A to hold information about residues for x near $\sqrt{n}$.
2. Generate the factor base and sieve:

Quadratic Sieve Algorithm

Given an input n:

1. Initialize a large array A to hold information about residues for x near $\sqrt{n}$.
2. Generate the factor base and sieve:
   - Solve $x^2 - n \equiv 0 \pmod{p}$ for increasing values of p.
   - Add the p to the factor base.
   - Update the entries in A corresponding to the solution.

Quadratic Sieve Algorithm

Given an input n:

1. Initialize a large array A to hold information about residues for x near $\sqrt{n}$.
2. Generate the factor base and sieve:
   - Solve $x^2 - n \equiv 0 \pmod{p}$ for increasing values of p.
   - Add the p to the factor base.
   - Update the entries in A corresponding to the solution.
3. From the prime factorizations, use the exponents to create a system of linear equations mod 2.
4. Find a solution to the system. (e.g., with Gaussian elimination)
5. Multiply the corresponding residues and recover the factors of n.

Analysis

With proper selection of the size of the factor base (too small and few residues factor over it; too large and we'll need a lot of residues to solve the system of equations), the expected runtime of the quadratic sieve is

$$O\left(\exp\left(\sqrt{9/8}\sqrt{\log n}\sqrt{\log\log n}\right)\right).$$

Analysis

With proper selection of the size of the factor base (too small and few residues factor over it; too large and we'll need a lot of residues to solve the system of equations), the expected runtime of the quadratic sieve is

$$O\left(\exp\left(\sqrt{9/8}\sqrt{\log n}\sqrt{\log\log n}\right)\right).$$

In particular, $\sqrt{9/8}$ is a smaller value than what appeared there in Dixon's algorithm.

In Practice

Currently, the quadratic sieve is the second-fastest known factoring algorithm, beaten only by the general number field sieve. For numbers under ≈ 100 digits, the quadratic sieve is still the fastest.

In Practice

Currently, the quadratic sieve is the second-fastest known factoring algorithm, beaten only by the general number field sieve. For numbers under ≈ 100 digits, the quadratic sieve is still the fastest.

The quadratic sieve was the first to factor RSA-129, a 129-digit semiprime (in 1994). The factor base used 524339 primes. The data collection (sieving) took over 5000 MIPS-years, distributed over 1600 computers. The data processing (solving the system) took another 45 hours on a supercomputer.

In Practice

Currently, the quadratic sieve is the second-fastest known factoring algorithm, beaten only by the general number field sieve. For numbers under ≈ 100 digits, the quadratic sieve is still the fastest.

The quadratic sieve was the first to factor RSA-129, a 129-digit semiprime (in 1994). The factor base used 524339 primes. The data collection (sieving) took over 5000 MIPS-years, distributed over 1600 computers. The data processing (solving the system) took another 45 hours on a supercomputer.

- We've come a long way since:

  > *In 2015, RSA-129 was factored in about one day, with the CADO-NFS open source implementation of number field sieve, using a commercial cloud computing service for about $30.*
  >
  > — Wikipedia

Questions?

Brainteaser

Assume 100 zombies are walking on a straight line, all moving with the same speed. Some are moving towards left, and some towards right. If a collision occurs between two zombies, they both reverse their direction. Initially all zombies are standing at 1 unit intervals. For every zombie, you can see whether it moves left or right, can you predict the number of collisions?

Bibliography I

Dixon's factorization method.

https://en.wikipedia.org/wiki/Dixon%27s_factorization_method.

Factoring algorithms lecture.

https://people.math.sc.edu/filaseta/gradcourses/Math788Lecture19.pdf.

Fermat's factorization method.

https://en.wikipedia.org/wiki/Fermat%27s_factorization_method.

Eric Landquist.

The quadratic sieve factoring algorithm.

https://www.cs.virginia.edu/crab/QFS_Simple.pdf, 2001.

Bibliography II

Vinnie Monaco.

Dixon’s algorithm and the quadratic sieve.

https://web.archive.org/web/20160205002504/https://vmonaco.com/dixons-algorithm-and-the-quadratic-sieve/, 2012.

Quadratic sieve.

https://en.wikipedia.org/wiki/Quadratic_sieve.