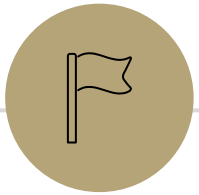
Wrap-up Number Theory

CSE 311 Autumn 2025
Lecture 14

Today

How does RSA work, though?



Number Theory Wrap-up

Plan For Number Theory Wrap up

We don't expect you to fully absorb the content in this section of the slides.

Our goals are:

1. See some definitions (these we do expect you to understand!)
2. Introduce an algorithm
3. See that number theory results can make code faster in unexpected ways.
4. See a bit of code analysis (a preview of 332).
5. Hopefully say "oh neat, I understand a *little bit* about how secure online communication works"

GCD and LCM

Greatest Common Divisor

The Greatest Common Divisor of a and b (gcd(a,b)) is the largest integer c such that $c|a$ and $c|b$

Least Common Multiple

The Least Common Multiple of a and b (lcm(a,b)) is the smallest positive integer c such that $a|c$ and $b|c$.

Try a few values…

gcd(100,125)

gcd(17,49)

gcd(17,34)

gcd(13,0)

lcm(7,11)

lcm(6,10)

How do you calculate a gcd?

You could:

Find the prime factorization of each

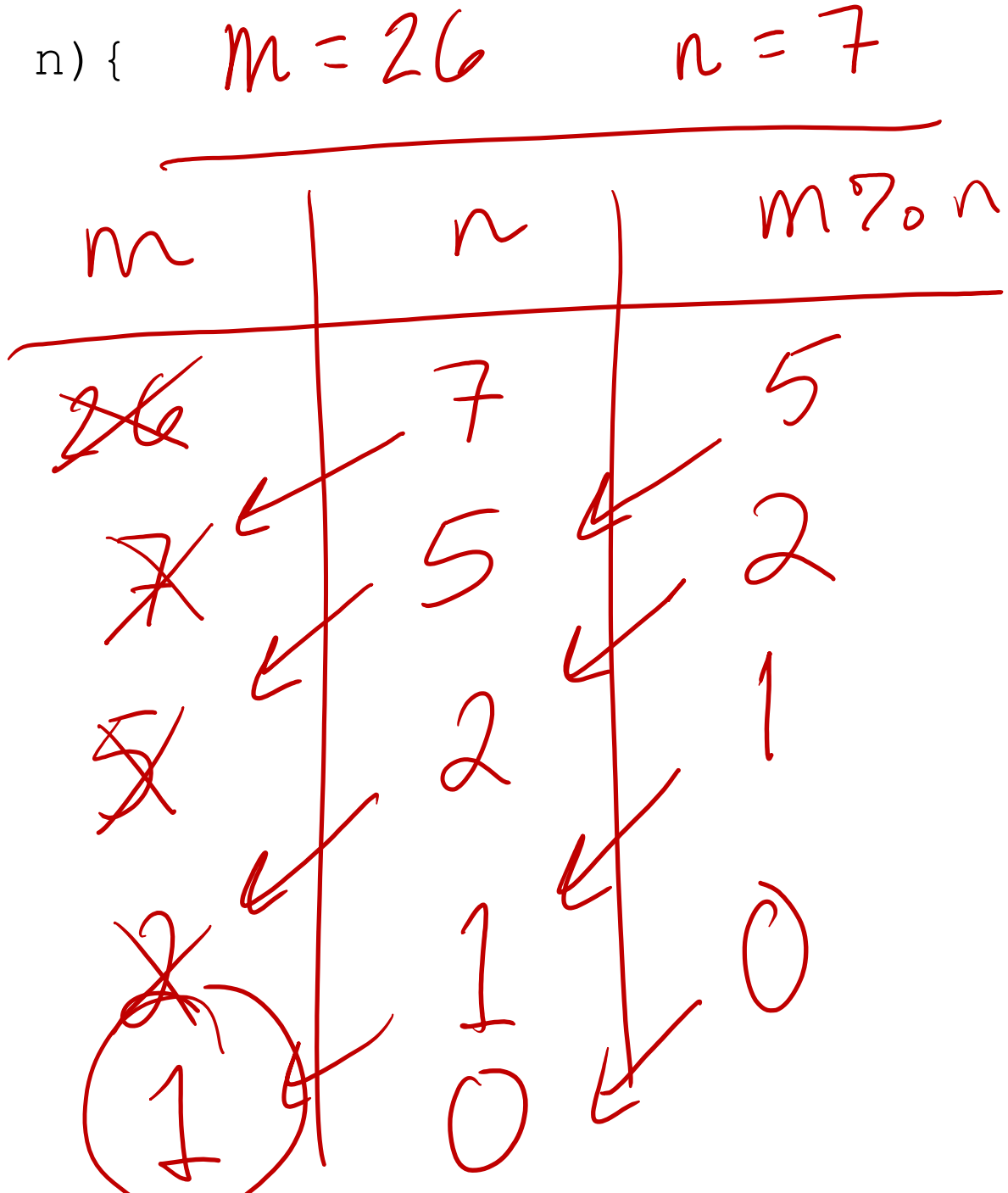
Take all the common ones. E.g.

gcd(24,20)=gcd($2^3 \cdot 3, 2^2 \cdot 5$) = 2^{min(2,3)} = 2^2 = 4.

(lcm has a similar algorithm – take the maximum number of copies of everything)

But that's….really expensive. Mystery finds gcd.

```
public int Mystery(int m, int n){
    if(m<n){
        int temp = m;
        m=n;
        n=temp;
    }
    while(n != 0) {
        int rem = m % n;
        m=n;
        n=rem;
    }
    return m;
}
```



GCD facts

1. gcd(a,0)=a

Pf: a is a common divisor ($a = 1 \cdot a$; $0 = 0 \cdot a$); larger numbers don't divide a (for positive numbers, if $x|y$ then $x \leq y$)

2. If a and b are positive integers, then gcd(a,b) = gcd(b, a % b)

Why is 2 true? The proof isn't easy, it's at the end of this deck.

Why should you care?

So…what's it good for?

Remember everything we're learning contributes to us eventually understanding RSA. This is a key step in generating keys.

Suppose I want to solve $7x \equiv 3 (mod\ n)$

Just multiply both sides by $\frac{1}{7}$…

Oh wait. We want a number to multiply by 7 to get 1.

What number can we pick?

The next two slides are going to get more abstract…we're listing out the facts we need to solve that equation.

Bézout's Theorem

Bézout's Theorem

If a and b are positive integers, then there exist integers s and t such that
gcd(a,b)= $sa + tb$

We're not going to prove this theorem…

But it turns out `Mystery` can be extended to find them.

We'll discuss that in the optional video.

So…what's it good for? With Bézout's

Suppose I want to solve $7x \equiv 3(mod\ n)$

Just multiply both sides by $\frac{1}{7}$…

Oh wait. We want a number to multiply by 7 to get 1.

If the gcd(7,n) = 1

Then $s \cdot 7 + tn = 1$, so $7s - 1 = -tn$ i.e. $n|(7s - 1)$ so $7s \equiv 1(mod\ n)$.

So the s from Bézout's Theorem is what we should multiply by!

Ok…how am I supposed to find s, t?

It turns out that while you're calculating the gcd (using the Mystery algorithm), you can keep some extra information recorded, and end up with the s, t

This is called the "extended Euclidian algorithm"

Examples in these slides.

Try it (set up)

Solve the equation $7y \equiv 3(mod\ 26)$

What do we need to find?

The multiplicative inverse of $7(\mathbf{mod}\ 26)$

Finding the inverse… (1)

gcd(26,7) = gcd(7, 26%7) = gcd(7,5)

= gcd(5, 7%5) = gcd(5,2)

= gcd(2, 5%2) = gcd(2, 1)

= gcd(1, 2%1) = gcd(1,0)= 1.

$26 = 3 \cdot 7 + 5$; $5 = 26 - 3 \cdot 7$

$7 = 5 \cdot 1 + 2$; $2 = 7 - 5 \cdot 1$

$5 = 2 \cdot 2 + 1$; $1 = 5 - 2 \cdot 2$

Finding the inverse… (2)

$\gcd(a,b) = sa + tb$

$1 = \_ \cdot 7 + \_ \cdot 26$

$\gcd(26,7) = \gcd(7, 26\%7) = \gcd(7,5)$

$= \gcd(5, 7\%5) = \gcd(5,2)$

$= \gcd(2, 5\%2) = \gcd(2, 1)$

$= \gcd(1, 2\%1) = \gcd(1,0) = 1.$

$$1 = 5 - 2 \cdot 2$$
$$= 5 - 2(7 - 5 \cdot 1)$$
$$= 3 \cdot 5 - 2 \cdot 7$$
$$= 3 \cdot (26 - 3 \cdot 7) - 2 \cdot 7$$
$$= 3 \cdot 26 - 11 \cdot 7$$

$26 = 3 \cdot 7 + 5$; $5 = 26 - 3 \cdot 7$

$7 = 5 \cdot 1 + 2$; $2 = 7 - 5 \cdot 1$

$5 = 2 \cdot 2 + 1$; $1 = 5 - 2 \cdot 2$

-11 is a multiplicative inverse of 7 for (mod 26) arithmetic!

We'll write that as 15, since we're working mod 26.

Try it (complete)

Solve the equation $7y \equiv 3(mod\ 26)$

What do we need to find?

The multiplicative inverse of $7\ (mod\ 26)$. We found it's 15.

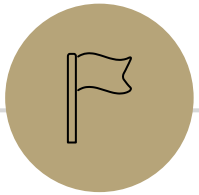
$15 \cdot 7 \cdot y \equiv 15 \cdot 3(mod\ 26)$

$y \equiv 45(mod\ 26)$

Or $y \equiv 19(mod\ 26)$

So $26|19 - y$, i.e. $26k = 19 - y$ (for $k \in \mathbb{Z}$) i.e. $y = 19 - 26 \cdot k$ for any $k \in \mathbb{Z}$

Solutions: $\{\ldots, -7, 19, 45, \ldots 19 + 26k, \ldots\}$ i.e. $\{x: x = 19 + 26k$ for some $k \in \mathbb{Z}\}$



RSA Encryption

Key Steps in RSA

Given two numbers, we can find their gcd quickly.

If we have an equation

$ax \equiv b (\mathrm{mod}\, n)$

And $\gcd(a, n) = 1$ then we can quickly find a number to multiply the equation by to solve for x.

Framing Device (key generation 1)

We're going to give you enough background to (mostly) understand the RSA encryption system.

Key generation [edit]

The keys for the RSA algorithm are generated in the following way:

1. Choose two distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random and should be similar in magnitude but differ in length by a few digits to make factoring harder.[2] Prime integers can be efficiently found using a primality test.
   - p and q are kept secret.
2. Compute $n = pq$.
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
   - n is released as part of the public key.
3. Compute $\lambda(n)$, where λ is Carmichael's totient function. Since $n = pq$, $\lambda(n) = \operatorname{lcm}(\lambda(p), \lambda(q))$, and since p and q are prime, $\lambda(p) = \varphi(p) = p - 1$, and likewise $\lambda(q) = q - 1$. Hence $\lambda(n) = \operatorname{lcm}(p - 1, q - 1)$.
   - $\lambda(n)$ is kept secret.
   - The lcm may be calculated through the Euclidean algorithm, since $\operatorname{lcm}(a, b) = |ab|/\gcd(a, b)$.
4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; that is, e and $\lambda(n)$ are coprime.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – the most commonly chosen value for e is $2^{16} + 1 = 65\,537$. The smallest (and fastest) possible value for e is 3, but such a small value for e has been shown to be less secure in some settings.[15]
   - e is released as part of the public key.
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; that is, d is the modular multiplicative inverse of e modulo $\lambda(n)$.
   - This means: solve for d the equation $d \cdot e \equiv 1 \pmod{\lambda(n)}$; d can be computed efficiently by using the extended Euclidean algorithm, since, thanks to e and $\lambda(n)$ being coprime, said equation is a form of Bézout's identity, where d is one of the coefficients.
   - d is kept secret as the *private key exponent*.

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the private (or decryption) exponent d, which must be kept secret. p, q, and $\lambda(n)$ must also be kept secret because they can be used to calculate d. In fact, they can all be discarded after d has been computed.[16]

Framing Device (key generation 2)

We're going to give you enough background to (mostly) understand the RSA encryption system.

Key generation [edit]

Prime Numbers

The keys for the RSA algorithm are genera

1. Choose two distinct prime numbers p and q.
   - For security purposes, the integers p and q should be chosen at random and should be similar in magnitude but differ in length by a few digits to make factoring harder.[2] Prime integers can be efficiently found using a primality test.
   - p and q are kept secret.

Modular Arithmetic

2. Compute $n = pq$.
   - n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
   - n is released as part of the public key.
3. Compute $\lambda(n)$, where λ is Carmichael's totient function. Since $n = pq$, $\lambda(n) = \operatorname{lcm}(\lambda(p), \lambda(q))$, and since p and q are prime, $\lambda(p) = \varphi(p) = p - 1$, and likewise $\lambda(q) = q - 1$. Hence $\lambda(n) = \operatorname{lcm}(p - 1, q - 1)$.
   - $\lambda(n)$ is kept secret.
   - The lcm may be calculated through the Euclidean algorithm, since $\operatorname{lcm}(a, b)$

Modular Multiplicative Inverse

4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; that is, e and $\lambda(n)$ are co
   - e having a short bit-length and small Hamming weight results in more efficient enc ... he most commonly chosen value for e is $2^{16} + 1 = 65\,537$. The smallest (and fastest) possible value for e is 3, but such a small value for e has been shown to be less secure in some settings.[15]
   - e is released as part of the public key.

Bezout's Theorem

5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; that is, d is the modular multiplicative inverse of e modulo $\lambda(n)$.
   - This means: solve for d the equation $d \cdot e \equiv 1 \pmod{\lambda(n)}$; d can be computed efficiently by using the extended Euclidean algorithm, since, thanks to e and $\lambda(n)$ being coprime, said equation is a form of Bézout's identity, where d is one of the coefficients.
   - d is kept secret as the *private key exponent*.

Extended Euclidian Algorithm

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the private (or decryp ... t also be kept secret because they can be used to calculate d. In fact, they can all be discarded after d has been computed.[16]

Framing Device (encryption, decryption 1)

We're going to give you enough background to (mostly) understand the RSA encryption system.

Encryption [edit]

After Bob obtains Alice's public key, he can send a message M to Alice.

To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c, using Alice's public key e, corresponding to

$$c \equiv m^e \pmod{n}.$$

This can be done reasonably quickly, even for very large numbers, using modular exponentiation. Bob then transmits c to Alice. Note that at least nine values of m will yield a ciphertext c equal to m,[22] but this is very unlikely to occur in practice.

Decryption [edit]

Alice can recover m from c by using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

Given m, she can recover the original message M by reversing the padding scheme.

Framing Device (encryption, decryption 2)

We're going to give you enough background to (mostly) understand the RSA encryption system.

Encryption [edit]

After Bob obtains Alice's public key, he can send a message M to Alice.

To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c, using Alice's public key e, corresponding to

$$c \equiv m^e \pmod{n}.$$

This can be done reasonably quickly, even for very large numbers, using modular exponentiation. Bob then transmits c to Alice. Note that at least nine values of m will yield a ciphertext c equal to m,[22] but this is very unlikely to occur in practice.

Modular Exponentiation

Decryption [edit]

Alice can recover m from c by using her private key exponent d by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

Given m, she can recover the original message M by reversing the padding scheme.

Imagine this scenario

- You want to send your credit card information to buy some dog treats for some very good boys  (Rufus and Gumball)
- The internet is a scary place – without encryption, we have to assume everything we send is visible to anyone who looks 
- When I send my credit card information to Amazon, I wanted it to be encrypted so  can't steal it
  - I want **encryption** and **decryption** to be fast (so Amazon and I can communicate efficiently)
  - I want  's attempts to **decrypt** the information to be so slow that it's infeasible

An application of all of this modular arithmetic

Amazon chooses random 512-bit (or 1024-bit) prime numbers p, q and an exponent e (often about 60,000).

Amazon calculates $\mathrm{n} = pq$. They tell your computer (n, e) (not p, q)

You want to send Amazon your credit card number a.

You compute $C = a^e \% n$ and send Amazon C.

Amazon computes d, the multiplicative inverse of e $(mod\ [p-1][q-1])$

Amazon finds $C^d \% n$

Fact: $a = C^d \% n$ as long as $0 < a < n$ and $p \nmid a$ and $q \nmid a$

How big are those numbers?

1230186684530117755130494958384962720772853569595334792197322
4521517264005072636575187452021997864693899564749427740638459
2519255732630345373154826850791702612214291346167042921431160
22212404792747377940806653514195974598569021434 13

=

3347807169895689878604416984821269081770479498371376856891243
1388982883793878002287614711652531743087737814467999489

×

3674604366679959042824463379962795263227915816434308764267603
2283815739666511279233373417143396810270092798736308917

How do we accomplish those steps?

That fact? You can prove it! It's sometimes an extra credit problem on a future homework. It's a nice combination of lots of things we've done with modular arithmetic. (You'll need a few additional pieces like Fermat's Little Theorem

Let's talk about finding $C = a^e \% n$.

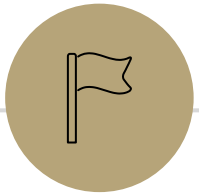
e is a BIG number (about 2^{16} is a common choice)

```
int total = 1;
for(int i = 0; i < e; i++){
    total = (a * total) % n;
}
```

We can do "%n" in these intermediate steps because of the identity

$(a \cdot b) \% n = (a \% n) \cdot (b \% n) \% n$

Try writing up a proof of this yourself!



Fast Exponentiation Algorithm

Let's build a faster algorithm.

Fast exponentiation – simple case. What if e is exactly 2^{16}?

```
int total = 1;
for(int i = 0; i < e; i++){
     total = a * total % n;
}
```

Instead:

```
int total = a;
for(int i = 0; i < log(e); i++){
     total = total^2 % n;
}
```

Fast Exponentiation Algorithm (steps)

What if e isn't exactly a power of 2?

Step 1: Write e in binary.

Step 2: Find $a^c \% n$ for c every power of 2 up to e.

Step 3: calculate a^e by multiplying a^c for all c where binary expansion of e had a 1.

Fast Exponentiation Algorithm (step 1)

Find $4^{11}\%10$ $e=11$

Step 1: Write e in binary.

Step 2: Find $a^c\%n$ for c every power of 2 up to e.

Step 3: calculate a^e by multiplying a^c for all c where binary expansion of e had a 1.

Start with largest power of 2 less than e (8). 8's place gets a 1. Subtract power

Go to next lower power of 2, if remainder of e is larger, place gets a 1, subtract power; else place gets a 0 (leave remainder alone).

11 = 1011_2

Fast Exponentiation Algorithm (step 2)

Find $4^{11}\%10$

Step 1: Write e in binary.

Step 2: Find $a^c\%n$ for c every power of 2 up to e.

Step 3: calculate a^e by multiplying a^c for all c where binary expansion of e had a 1.

$4^1\%10 = 4$

$4^2\%10 = 6$

$4^4\%10 = 6^2\%10 = 6$

$4^8\%10 = 6^2\%10 = 6$

Fast Exponentiation Algorithm (step 3)

Find $4^{11}\%10$

Step 1: Write e in binary.

Step 2: Find $a^c\%n$ for c every power of 2 up to e.

Step 3: calculate a^e by multiplying a^c for all c where binary expansion of e had a 1.

$4^1\%10 = 4$

$4^2\%10 = 6$

$4^4\%10 = 6^2\%10 = 6$

$4^8\%10 = 6^2\%10 = 6$

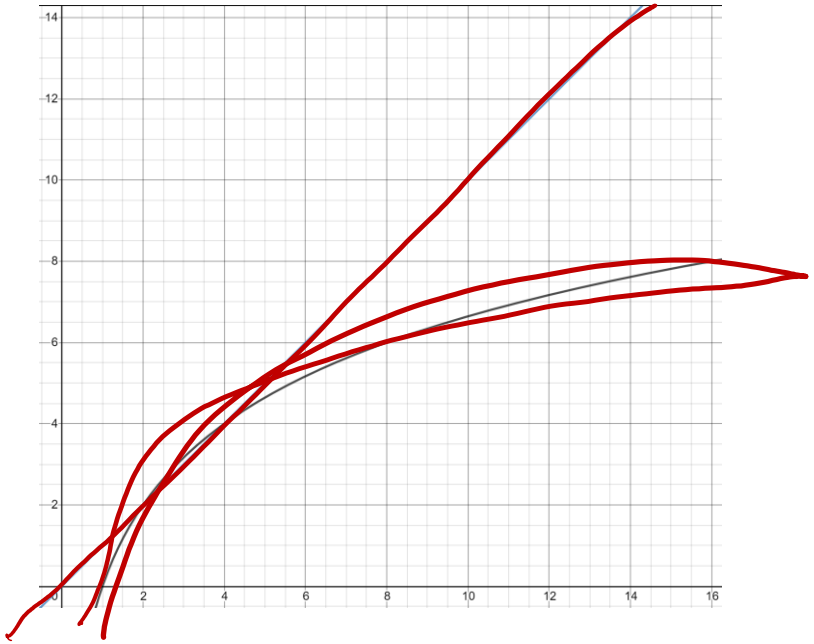
$$4^{11}\%10 = 4^{8+2+1}\%10 =$$
$$[(4^8\%10)\cdot(4^2\%10)\cdot(4\%10)]\%10 = (6\cdot 6\cdot 4)\%10$$
$$= (36\%10\cdot 4)\%10 = (6\cdot 4)\%10 = 24\%10 = 4.$$

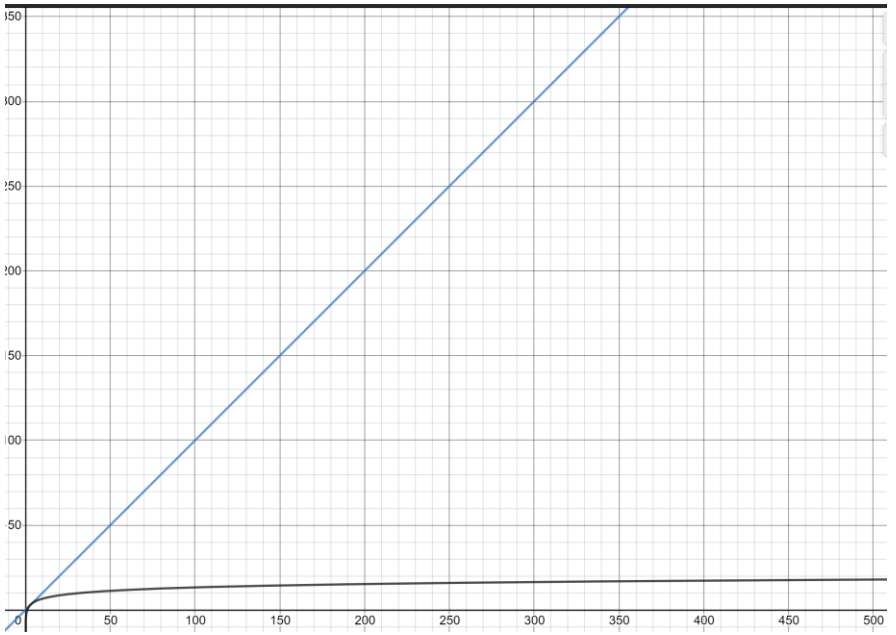
Fast Exponentiation Algorithm (comparison)

Is it…actually fast?

The number of multiplications is between $\log_2 e$ and $2\log_2 e$.

That's A LOT smaller than e





One More Example for Reference (1)

Find $3^{25}\%7$ using the fast exponentiation algorithm.

Find 25 in binary:

16 is the largest power of 2 smaller than 25. $(25-16)=9$ remaining

8 is smaller than 9. $(9-8)=1$ remaining.

4s place gets a 0.

2s place gets a 0

$1s$ place gets a 1

11001_2

One More Example for Reference (2)

Find $3^{25}\%7$ using the fast exponentiation algorithm.

Find $3^{2^i}\%7$:

$3^1\%7 = 3$

$3^2\%7 = 9\%7 = 2$

$3^4\%7 = (3^2 \cdot 3^2)\%7 = (2 \cdot 2)\%7 = 4$

$3^8\%7 = (3^4 \cdot 3^4)\%7 = (4 \cdot 4)\%7 = 2$

$3^{16}\%7 = \left(3^8 \cdot 3^8\right)\%7 = (2 \cdot 2)\%7 = 4$

One More Example for Reference (3)

Find $3^{25}\%7$ using the fast exponentiation algorithm.

$3^1\%7 = 3$

$3^2\%7 = 2$

$3^4\%7 = 4$

$3^8\%7 = 2$

$3^{16}\%7 = 4$

$3^{25}\%7 = 3^{16+8+1}\%7$
$= [(3^{16}\%7) \cdot (3^8\%7) \cdot (3^1\%7)]\%7$
$= [4 \cdot 2 \cdot 3]\%7$
$= (1 \cdot 3)\%7 = 3$

A Brief Concluding Remark

Why does RSA work? i.e. why is my credit card number "secret"?

Raising numbers to large exponents (in mod arithmetic) and finding multiplicative inverses in modular arithmetic are things computers can do quickly.

But factoring numbers (to find p, q to get d) or finding an "exponential inverse" (not the real term) directly are not things computers can do quickly. At least as far as we know.

An application of all of this modular arithmetic (reminder)

Amazon chooses random 512-bit (or 1024-bit) prime numbers p, q and an exponent e (often about 60,000).

Amazon calculates $\mathrm{n} = pq$. They tell your computer (n, e) (not p, q)

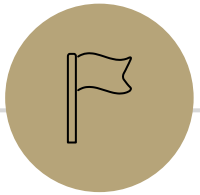
You want to send Amazon your credit card number a.

You compute $C = a^e \% n$ and send Amazon C.

Amazon computes d, the multiplicative inverse of e $(mod\ [p-1][q-1])$

Amazon finds $C^d \% n$

Fact: $a = C^d \% n$ as long as $0 < a < n$ and $p \nmid a$ and $q \nmid a$



Why does the Euclidian Algorithm Work?

Correctness of an algorithm

The key to the Euclidian Algorithm being correct is that each time through the loop, you don't change the gcd of the variables `m,n.`

To prove the code correct, you really want an induction proof (it's good practice to think about it!). The inductive step relies on the fact we stated but didn't prove:

```
gcd(a,b) = gcd(b, a%b).
```

Let's prove it!

GCD fact

If a and b are positive integers, then gcd(a,b) = gcd(b, a % b)

How do you show two gcds are equal?

Call $a = \gcd(w, x), b = \gcd(y, z)$

If $b|w$ and $b|x$ then b is a common divisor of w, x so $b \leq a$

If $a|y$ and $a|z$ then a is a common divisor of y, z, so $a \leq b$

If $a \leq b$ and $b \leq a$ then $a = b$

gcd(a,b) = gcd(b, a % b) (slide 1)

Let $\mathrm{x} = \gcd(a, b)$ and $y = \gcd(b, a\%b)$.

We show that y is a common divisor of a and b.

By definition of gcd, $y|b$ and $y|(a\%b)$. So it is enough to show that $y|a$.

Applying the definition of divides we get $b = yk$ for an integer k, and $(a\%b) = yj$ for an integer j.

By definition of mod, $a\%b$ is $a = qb + (a\%b)$ for an integer q.

Plugging in both of our other equations:

$a = qyk + yj = y(qk + j)$. Since q, k, and j are integers, $y|a$. Thus y is a common divisor of a, b and thus $y \leq x$.

gcd(a,b) = gcd(b, a % b) (slide 2)

Let $\mathrm{x} = \gcd(a, b)$ and $y = \gcd(b, a\%b)$.

We show that x is a common divisor of b and $\mathrm{a}\%b$.

By definition of gcd, $\mathrm{x}|b$ and $x|a$. So it is enough to show that $\mathrm{x}|(a\%b)$.

Applying the definition of divides we get $b = xk'$ for an integer k', and $\mathrm{a} = xj'$ for an integer j'.

By definition of mod, $a\%b$ is $a = qb + (a\%b)$ for an integer q

Plugging in both of our other equations:

$xj' = qxk' + a\%b$. Solving for $a\%b$, we have $a\%b = xj' - qxk' = x(j' - qk')$. So $x|(a\%b)$. Thus x is a common divisor of $b, a\%b$ and thus $x \leq y$.

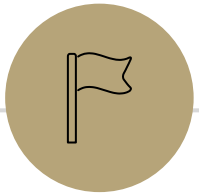
gcd(a,b) = gcd(b, a % b) (slide 3)

Let $\mathrm{x} = \gcd(a, b)$ and $y = \gcd(b, a\%b)$.

We show that x is a common divisor of b and $\mathrm{a}\%b$.

We have shown $x \leq y$ and $y \leq x$.

Thus $x = y$, and $\gcd(a, b) = \gcd(b, a\%b)$.



Another Extended Euclidian Algorithm Example

Extended Euclidian Algorithm (steps)

Step 1 compute gcd(a,b); keep tableau information.

Step 2 solve all equations for the remainder.

Step 3 substitute backward

gcd(35,27)

Extended Euclidian Algorithm (step 1)

Step 1 compute gcd(a,b); keep tableau information.

Step 2 solve all equations for the remainder.

Step 3 substitute backward

gcd(35,27) = gcd(27, 35%27) = gcd(27,8)
= gcd(8, 27%8) = gcd(8, 3)
= gcd(3, 8%3) = gcd(3, 2)
= gcd(2, 3%2) = gcd(2,1)
= gcd(1, 2%1) = gcd(1,0)

$$35 = 1 \cdot 27 + 8$$
$$27 = 3 \cdot 8 + 3$$
$$8 = 2 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$

Extended Euclidian Algorithm (step 2 p1)

Step 1 compute gcd(a,b); keep tableau information.

Step 2 solve all equations for the remainder.

Step 3 substitute backward

$$
\begin{aligned}
35 &= 1 \cdot 27 + 8 \\
27 &= 3 \cdot 8 + 3 \\
8 &= 2 \cdot 3 + 2 \\
3 &= 1 \cdot 2 + 1
\end{aligned}
$$

Extended Euclidian Algorithm (step 2 p2)

Step 1 compute gcd(a,b); keep tableau information.

Step 2 solve all equations for the remainder.

Step 3 substitute backward

$$
\begin{aligned}
35 &= 1 \cdot 27 + 8 \\
27 &= 3 \cdot 8 + 3 \\
8 &= 2 \cdot 3 + 2 \\
3 &= 1 \cdot 2 + 1
\end{aligned}
$$

$$
\begin{aligned}
8 &= 35 - 1 \cdot 27 \\
3 &= 27 - 3 \cdot 8 \\
2 &= 8 - 2 \cdot 3 \\
1 &= 3 - 1 \cdot 2
\end{aligned}
$$

Extended Euclidian Algorithm (step 3 p1)

Step 1 compute gcd(a,b); keep tableau information.

Step 2 solve all equations for the remainder.

Step 3 substitute backward

$$
\begin{aligned}
8 &= 35 - 1 \cdot 27 \\
3 &= 27 - 3 \cdot 8 \\
2 &= 8 - 2 \cdot 3 \\
1 &= 3 - 1 \cdot 2
\end{aligned}
$$

Extended Euclidian Algorithm (step 3 p2)

Step 1 compute gcd(a,b); keep tableau information.

Step 2 solve all equations for the remainder.

Step 3 substitute backward

$$
\begin{aligned}
8 &= 35 - 1 \cdot 27 \\
3 &= 27 - 3 \cdot 8 \\
2 &= 8 - 2 \cdot 3 \\
1 &= 3 - 1 \cdot 2
\end{aligned}
$$

$$
\begin{aligned}
1 &= 3 - 1 \cdot 2 \\
&= 3 - 1 \cdot (8 - 2 \cdot 3) \\
&= -1 \cdot 8 + 2 \cdot 3
\end{aligned}
$$

Extended Euclidian Algorithm (step 3 p3)

Step 1 compute gcd(a,b); keep tableau information.

Step 2 solve all equations for the remainder.

Step 3 substitute backward

$$
\begin{aligned}
8 &= 35 - 1 \cdot 27 \\
3 &= 27 - 3 \cdot 8 \\
2 &= 8 - 2 \cdot 3 \\
1 &= 3 - 1 \cdot 2
\end{aligned}
$$

gcd(27,35) = $13 \cdot 27 + (-10) \cdot 35$

$$
\begin{aligned}
1 &= 3 - 1 \cdot 2 \\
&= 3 - 1 \cdot (8 - 2 \cdot 3) \\
&= -1 \cdot 8 + 3 \cdot 3 \\
&= -1 \cdot 8 + 3(27 - 3 \cdot 8) \\
&= 3 \cdot 27 - 10 \cdot 8 \\
&= 3 \cdot 27 - 10(35 - 1 \cdot 27) \\
&= 13 \cdot 27 - 10 \cdot 35
\end{aligned}
$$

When substituting back, you keep the larger of m, n and the number you just substituted. Don't simplify further! (or you lose the form you need)