To fix this problem, we need to rewrite our number in binary. Because we also want to comment on the speed of this algorithm, let's go through how we would do this. (We'll do base "B").

Given $B \geq 2$; we want to write any integer x as

$$x = a_n B^n + a_{n-1} B^{n-1} + \dots + a_1 B + a_0 \quad \text{where}$$

$a_i \in \{0, \dots, B-1\}$. To do this:

- Find n such that $B^n \leq x < B^{n+1}$, from this we know the base B expansion will have $n+1$ digits.
- Choose a_n to be the largest element of $\{1, 2, \dots, B-1\}$ such that $a_n B^n \leq x < (a_n+1) B^n$
- Repeat previous ~~[illegible]~~ steps on $x - a_n B^n$ to find a_{n-1}, then repeat on $x - a_n B^n - a_{n-1} B^{n-1}$, etc.

Here is how we can use this to take modular exponents quickly: Let's work out an example.

Ex: Compute $107^{101} \bmod 113$.

Solution: The trick is to first write 101 in binary.

Since $2^6 = 64 \leq 101 < 128 = 2^7$, we know we will need 7 digits. The first one has to be "1".

So

$$101 = 1 \cdot 64 + ?$$

Now $101 - 64 = 37$. Since $2^5 = 32 \leq 37 < 64 = 2^6$, we know the second digit is also a "1":

$$101 = 1 \cdot 64 + 1 \cdot 32 + ?$$

Now $37 - 32 = 5$. Since $5 < 2^4 = 16$ and $5 < 2^3 = 8$, the next two digits are zero:

$$101 = 1 \cdot 64 + 1 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + ?$$

Last, $2^2 \leq 5 < 2^3$ so we get a "1", and finishing the whole thing off:

$$101 = 1\ 1\ 0\ 0\ 1\ 0\ 1$$

Next, compute 107^{2^k} for $k = 0, 1, 2, 3, 4, 5, 6$. We can do this quickly using repeated squaring:

k	2^k	$m^{2^k} \bmod 113$
0	1	$107^1 = 107 \bmod 113$
1	2	$107 \cdot 107 = 36 \bmod 113$
2	4	$36 \cdot 36 = 53 \bmod 113$
3	8	$53 \cdot 53 = 97 \bmod 113$
4	16	$97 \cdot 97 = 30 \bmod 113$
5	32	$30 \cdot 30 = 109 \bmod 113$
6	64	$109 \cdot 109 = 16 \bmod 113$

Having constructed the table, we use the binary expansion of e to write:

$$107^{101} = 107^{64+32+4+1} = \underbrace{107^{64} \cdot 107^{32} \cdot 107^{4} \cdot 107}_{\text{a product of table entries!}} \quad \text{mod } 113.$$

$$= 16 \cdot 109 \cdot 53 \cdot 107 \text{ mod } 113$$

$$= 12 \text{ mod } 113$$

How fast is this?

- To make the table, we need to do one multiplication for each digit in the binary expansion of the exponent.
- To compute the final product, we need to do at most one multiplication for each digit in the binary expansion of the exponent.

<u>This means</u>: If e is the exponent, the number of digits in the binary expansion of e is

$$\lceil \log_2 e \rceil$$

where $\lceil x \rceil$ is the "ceiling" of the number, ie the ~~grea~~ least integer larger than $\log_2 e$. So total number of multiplications is

$$2 \lceil \log_2 e \rceil \quad \text{as opposed to "e"!}$$

$\Longrightarrow$ ENORMOUS SAVINGS!

§7.5 and 7.6 A primer on primes and Fermat's little theorem.

First, recall that the primes are the positive numbers that have exactly two (positive) divisors (p prime iff the divisors of p are 1 and p).

We are going to need primes (big ones) to make RSA work. To this end, it's useful to know a couple facts:

(1) There are infinitely many primes.

Proof: Suppose there are finitely many, write them in a list.

$$2, 3, 5, 7, \dots, p_L$$

where p_L is the "last prime". Set

$$S = 2 \cdot 3 \cdot 5 \cdot 7 \cdot \dots \cdot p_L + 1 \quad \text{their product } + 1.$$

First, note that $S > p_L$, so S cannot be prime. This means that S is composite, ie. it is a product of primes.

Note that if ~~p, q are~~ p is a primes, then

$$S = pn \Rightarrow S \equiv 0 \bmod p,$$

ie if p divides n it's going to give remainder 0.
However from our construction, S satisfies:

$$S \equiv 1 \bmod 2, \quad S \equiv 1 \bmod 3, \quad S \equiv 1 \bmod 5, \dots$$

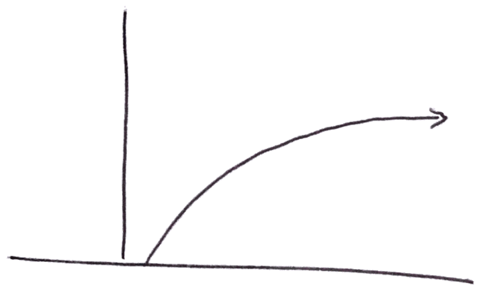
and in general $s \equiv 1 \bmod p$ for every prime p.
This means s has no prime divisors, a contradiction

(2) Primes become more a more rare as numbers get larger. I.e. between 0 and 100 there will be many more primes than between 10^{10} and $10^{10}+100$.
To make this explicit:

Set $\pi(x) = \#\{p \leq x \mid p \text{ is prime}\}$, the number of primes $\leq x$.

From (1), we know $\lim\limits_{x \to \infty} \pi(x) = \infty$. But how does it go to infinity? Is it like an exponential?

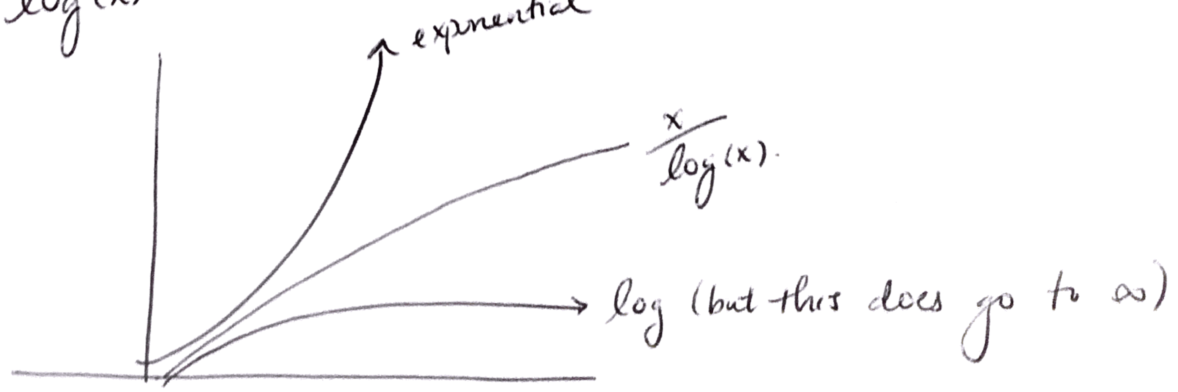
or like a log?



<u>Ans</u>: $\lim\limits_{x\to\infty} \pi(x) \Big/ \left(x \big/ \log(x)\right) = 1$, meaning it's

natural log "ln"

like $x \big/ \log(x)$, which is in between..



§7.6 Fermat's little Theorem

This is the last ingredient needed for RSA.

(FLT): Let p be a prime number, and suppose a is not "a" multiple of p. Then

$$a^{p-1} \equiv 1 \mod p.$$

E.g. Suppose $p=7$ and $a=3$. What is $3^6 \mod 7$?

Well, $6 = 4+2 = 110$ in binary. Make a table:

k	2^k	$3^{2^k} \mod 7$
0	1	$3 \mod 7$
1	2	$3 \cdot 3 = 2 \mod 7$
2	4	$2 \cdot 2 = 4 \mod 7$.

Then $3^6 \mod 7 = 3^{4+2} = 4 \cdot 2 = 8 = 1 \mod 7$. So it works!.

On the other hand, it's actually useful when the theorem fails. E.g.

Consider $3^{11} \mod 12$. As before, write

$11 = 2^3 + 2 + 1 = 1011$. Make a table:

k	2^k	$3^{2^k} \mod 12$
0	1	$3 \mod 12$
1	2	$9 \mod 12$
2	4	$9 \cdot 9 = 81 = 9 \mod 12$
3	8	$9 \cdot 9 = 81 = 9 \mod 12$.

So then

$3^{11} = 3^{8+2+1} = 9 \cdot 9 \cdot 3 = 3 \bmod 12$, so the theorem's result fails <u>because 12 is not prime</u>!

So not only do we have a nice fact about primes, we also have a trick to show p is not prime by doing repeated multiplication (instead of repeated division — we could just try all factors to test primeness).

<u>Proof of Fermat's little theorem</u>:

Suppose $p > 2$ is prime, and that a is an integer with $\gcd(a,p) = 1$. Then

$$a,\ 2a,\ 3a,\ \dots,\ (p-1)a$$

are all distinct mod p (Question 2 of Assignment 2 — Proof is also in the text on page 199)

This means that

$$a \cdot 2a \cdot 3a \cdot \dots \cdot (p-a)a \bmod p$$

must be the same as

$$1 \cdot 2 \cdot 3 \cdot \dots \cdot (p-1) \bmod p$$

Since it's the same collection of numbers in each product, but possibly in a different order. The second quantity is equal to $(p-1)!$, the first is $a^{p-1}(p-1)!$

So $a^{p-1}(p-1)! = (p-1)! \bmod p$.

But now $(p-1)!$ is relatively prime to p, so it has an inverse mod p (ie there's $r \in \{0, \dots, p-1\}$ with $r \cdot (p-1)! = 1 \bmod p$.

Again this is from assignment 2 Question 2.

Then $\cancel{r\cdot(p-1)!}\, a^{p-1} \equiv \cancel{r\cdot(p-1)!} \mod p$

$a^{p-1} \equiv 1 \mod p$. So the theorem is true.

Finding primes with FLT:

We already saw $3^{11} \equiv 3 \mod 12$, since 12 is not prime.
We can use computations like this to check for primality of absurdly large (100's of digits) numbers, much faster than "divide by everything".

FLT primality test

Is q prime?

1. Pick a random number a with $1 < a < q$.
2. Compute $a^{q-1} \mod q$.
3. If $a^{q-1} \not\equiv 1 \mod q$, then q is not prime.
4. If $a^{q-1} \equiv 1 \mod q$, q might be prime.

But how likely is it that q is prime?

To facilitate our discussion, let's introduce a new term: having fixed q, a number "a" will be called a Fermat witness of q if $a^{q-1} \not\equiv 1 \mod q$, ie if it witnesses the fact that q is not prime.

Q: When q is composite, how any numbers a with $1 < a < q$ are Fermat witnesses for q?

A: It turns out there are two cases.

Case 1 : q has no Fermat witnesses whatsoever, so the FLT primality test is doomed to fail.
Fortunately these numbers are rare. They're called Carmichael numbers. (Very rare as you look at larger and larger numbers - a random 100-digit number has less than $\frac{1}{10^{32}}$ chance of being a Carmichael number).

Case 2 : Roughly 50% of the numbers a with $1 < a < q$ and $\gcd(a, q) = 1$ are Fermat witnesses.
This means if we choose a with $1 < a < q$, then
① Compute $\gcd(a, q)$. If it's not 1 stop, because q is not prime
② If it is 1 then compute a^{q-1}. If $a^{q-1} \neq 1 \bmod q$, stop because q is composite.
If these two steps fail to yield an answer, then you must've chosen a from the 50% of numbers $1 < a < q$ with $\gcd(a, q) = 1$ that are not witnesses. If the two steps above fail for a second choice a', then there's a

$$\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$$ chance that q is composite. Continuing,

you can eventually be "fairly certain" that q is prime (E.g. $\frac{1}{2^{32}}$ or something similarly enormous in only 32 tests).

<u>Example</u>: Suppose you want a 100-digit prime. Since $\pi(x) \sim x/\log(x)$, we can use this to estimate that roughly 0.5% of the 100-digit numbers are prime. So after 200 attempts, we'll probably hit on a prime.

For each attempt, we'll want to check a bunch of Fermat witnesses, say we check 50 of them. Each requires only $2\lceil \log x \rceil$ multiplications. Overall, checking 50 witnesses for 200 numbers is computationally feasible. In the end, we'll have a number that is almost definitely a prime ($\sim \frac{1}{2^{50}}$ chance it's not).

Chapter 8 RSA.

Finally, with everything in place we develop RSA. (Named after creators Rivest, Shamir, Adleman).

Recall that this is a public-key cryptosystem, so the security of it will be based on a "hard" mathematical problem that is easy to solve if you have a bit of secret info.

E.g. KidsRSA was based on the idea that a public key (e,n) with corresponding private key (d,n) (with $ed \equiv 1 \mod n$) did not allow you to compute the private key "d" on your own without great effort.

That this assumption proved false (we can use the Extended Euclidean Algorithm to solve

$ex \equiv 1 \mod n$)

was the key to cracking KidsRSA.

"Real" RSA is based on the difficulty of factoring large integers. E.g. If p, q are prime, it is very difficult to factor $n = pq$ any faster than trial and error. E.g. how would one factor

$\lfloor \pi \times 10^{100} \rfloor$? (it's an integer...)

Before describing RSA, we need some notation. The <u>Euler Totient function</u> $\varphi(n)$ counts the number of positive integers relatively prime to n and less than n. I.e.

$$\varphi(n) = \#\{m \mid 1 \leq m \leq n \text{ and } \gcd(m,n) = 1\}.$$

<u>Example</u>: If p is prime, then every m with $1 \leq m < p$ also satisfies $\gcd(m,p) = 1$. So

$$\varphi(p) = \#\{1, 2, 3, \ldots, p-1\}$$

note that $\gcd(p,p) = p$ so it's not in here.

If p and q are distinct primes, then $\varphi(pq) = (p-1)(q-1)$, here's why:

If $1 \leq n \leq pq$ and $\gcd(n, pq) \neq 1$, then n must be a multiple of p or a multiple of q. Therefore

$$\varphi(pq) = \# \underbrace{\left(\{1, 2, \ldots, pq\} \setminus \left\{\begin{matrix}\text{multiples of } p \text{ and} \\ \text{multiples of } q\end{matrix}\right\}\right)}_{\text{how to determine the size of this set?}}.$$

Well, $\{1, 2, \ldots, pq\}$ contains $p, 2p, 3p, \ldots, qp$ for a total of q multiples of p. It also contains $q, 2q, 3q, \ldots, pq$ for a total of q multiples of p.

So we have

$$\underbrace{\{1, 2, \ldots, pq\}}_{pq \text{ elements}} - \underbrace{\{\text{multiples of } p\}}_{q \text{ elements}} - \underbrace{\{\text{multiples of } q\}}_{p \text{ elements}}$$

So we have $pq-p-q$ elements... but we took away $\underline{pq \text{ twice}}$ so a proper count of elements is

$$\underline{\underline{\varphi(pq) = pq-p-q+1}} = (p-1)(q-1)$$

At last, here is <u>how</u> RSA works (not why, though)

Alice wants to send a message to Bob. Bob prepares keys as follows:

(1) Choose primes p and q

(2) Set $n = pq$, compute $\varphi(n) = (p-1)(q-1)$.

(3) Choose e with with $\gcd(e, \varphi(n)) = 1$ and calculate d with $de \equiv 1 \bmod \varphi(n)$.

<u>Remark</u>: These first steps use the primality testing that we covered (Fermat witnesses, etc) and also the extended Euclidean Algorithm. (to find e and d)

(4) Bob shares (e,n), his public key, and keeps (d,n), his private keys to himself.

__Note__: He also has to keep $\varphi(n)$ secret, as this can be used to compute d from e by the Euclidean Alg.

Alice wants to send Bob a message, say m where $1 \leq m \leq n$ and $\gcd(m,n)=1$. She computes

$$m^e \bmod n$$

and sends this number to Bob. Bob computes

$$(m^e)^d \bmod n$$ and miraculously gets m, Alice's message.

__Remark__: It is <u>not</u> supposed to make sense at this point why $(m^e)^d \bmod n = m$. We will prove this later.

__Example__: Bob needs to use huge numbers (hundreds of digits) for this to be secure, but for now we do a baby example.

Bob chooses primes $p=7$, $q=11$ (eleven). So $n=pq=77$, he computes $\varphi(n)=(7-1)(11-1)$
$=60$.

Then he chooses $e=13$, since e is prime we know $\gcd(13,60)=1$. Use the Euclidean algorithm to find its inverse mod 60,

(We'll write it in shorthand)

$60 = 4 \cdot 13 + 8$

$13 = 1 \cdot 8 + 5$

$8 = 1 \cdot 5 + 3$

$5 = 1 \cdot 3 + 2$

$3 = 1 \cdot 2 + \boxed{1}$

$\uparrow \gcd(13, 60).$

In reverse now:

$1 = 3 - 2 = 3 - (5 - 3)$

$1 = 2 \cdot 3 - 5 = 2 \cdot (8 - 5) - 5$

$1 = 2 \cdot 8 - 3 \cdot 5 = 2 \cdot 8 - 3 \cdot (13 - 8)$

$1 = 5 \cdot 8 - 3 \cdot 13 = 5 \cdot (60 - 4 \cdot 13) - 3 \cdot 13$

$1 = 5 \cdot 60 - \boxed{23}\, 13.$

Now note: This gives $-23 \cdot 13 = 1 - 5 \cdot 60$

or $-23 \cdot 13 = 1 \mod 60.$

So the integer "d" with $de = 1 \mod 60$ is $-23 = 37 \mod 60$. So Bob's private key is $(37, 77)$ and Alice will use his public key $(13, 77)$.

Alice decides to send $m = 17$. So she computes $17^{13} \mod 77$, done as follows:

$13 = 2^3 + 2^2 + 2^0$ (ie 1101 in binary)

So you make a table

k	2^k	$17^{2^k} \mod 77$
0	1	17
1	2	
2	4	
3	8	

mod 77

$17^2 = 58 \mod 77$

$58^2 = 53 \mod 77$

$53^2 = 37 \mod 77$

So then $17^{13} = 17^{8+4+1} = 37 \cdot 53 \cdot 17 = 73 \bmod 77$.

Bob receives the message "73". To decrypt, he computes $73^{37} = 73^{2^5+2^2+1} = 17 \bmod 77$.

It works, but also shows that sometimes things can behave strangely!

§8.2 RSA and symmetric encryption.

One drawback of RSA is that it is extremely slow to encrypt large messages. Here is why:

Suppose we've chosen primes p, q and computed n and $\varphi(n)$, and it happens that $\varphi(n)$ has 400 digits. Then we can send a message m using RSA that consists of at most 400 digits, and we've carefully discussed how much computation this requires – it's a fair amount, but not impossible for modern computers.

However most messages will be longer than 400 digits. E.g. the Wikipedia article about the coronavirus pandemic is nearly 200 000 characters, so we'd find ourselves doing hundreds (or thousands) of encryptions to send this single website in chunks.

The solution is to encrypt your message with a secure symmetric encryption scheme (such as AES, not covered in class) and then use RSA to send the key.

Example: Suppose Bob's public key is (13, 77) as in the previous example. Alice could send Bob the message (publicly, no less!)

Hi Bob,

I've used the LFSR $c_6 = c_4' + c_2' + c_1'$ to generate a pseudo-random stream and used it to encrypt a message to you. The encrypted message is

0 1 1 0 1 1 1 0 0 0

The seed I used for the LFSR, when changed from binary to decimal and encrypted using your RSA public key, is 17.

Bob receives this message from Alice, and using his private key of (37, 77) he computes

$$17^{37} \bmod 77 = 52 \bmod 77.$$

Writing 52 in binary, he gets a seed:

$$52 = 32 + 16 + 4$$
$$= 110100$$

Then he computes:

step	string					
0	1	1	0	1	0	0
1	0	1	1	0	1	0
2	0	0	1	1	0	1
3	0	0	0	1	1	0
4	1	0	0	0	1	1
5	0	1	0	0	0	1
6		0	1	0	0	0
7			0	1	0	0
8				0	1	0
9					0	1
10						0

stream.

Then he XORs:

$$\begin{array}{rcc} \text{ciphertext} \rightarrow & 0\ 1\ 1\ 0\ 1 & 1\ 1\ 0\ 0\ 0 \\ \text{keystream} \rightarrow & 0\ 0\ 1\ 0\ 1 & 1\ 0\ 0\ 0\ 1 \\ \hline & 0\ 1\ 0\ 0\ 0 & 0\ 1\ 0\ 0\ 1 \end{array} \leftarrow \text{message!}$$

He recognizes these as the abbreviated ASCII code for the letters "H I". (Abbreviated as outlined on assignment 3).