Fast Exponentiation Algorithm

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
2221240479274737794080665351419597459856902143413

=

3347807169895689878604416984821269081770479498371376856891243
1388982883793878002287614711652531743087737814467999489

×

3674604366679959042824463379962795263227915816434308764267603
2283815739666511279233373417143396810270092798736308917

How do we accomplish those steps?

That fact? You can prove it in the extra credit problem on HW5. It's a nice combination of lots of things we've done with modular arithmetic.

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;
}
```

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

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

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.

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

Find $4^{11}\%10$

Step 1: Write e in binary.

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

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

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 $\boldsymbol{a^e}$ by multiplying $\boldsymbol{a^c}$ for all $\boldsymbol{c}$ where binary expansion of $\boldsymbol{e}$ had a $\mathbf{1}$.

$4^1\%10 = 4$

$4^2\%10 = 6$

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

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

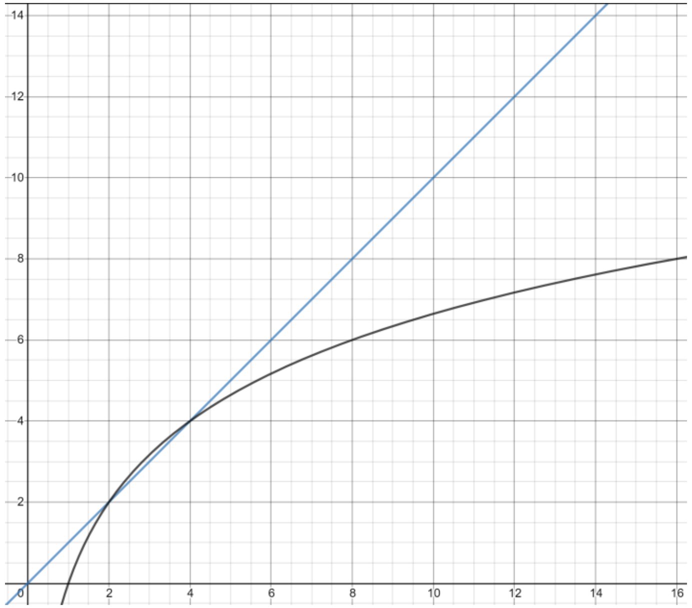
$4^{11}\%10 = 4^{8+2+1}\%10 =$
$[(4^8\%10)\cdot(4^2\%10)\cdot(4\%10)]\%10 = (6\cdot 6\cdot 4)\%10$
$= (36\%10\cdot 4)\%10 = (6\cdot 4)\%10 = 24\%10 = 4.$

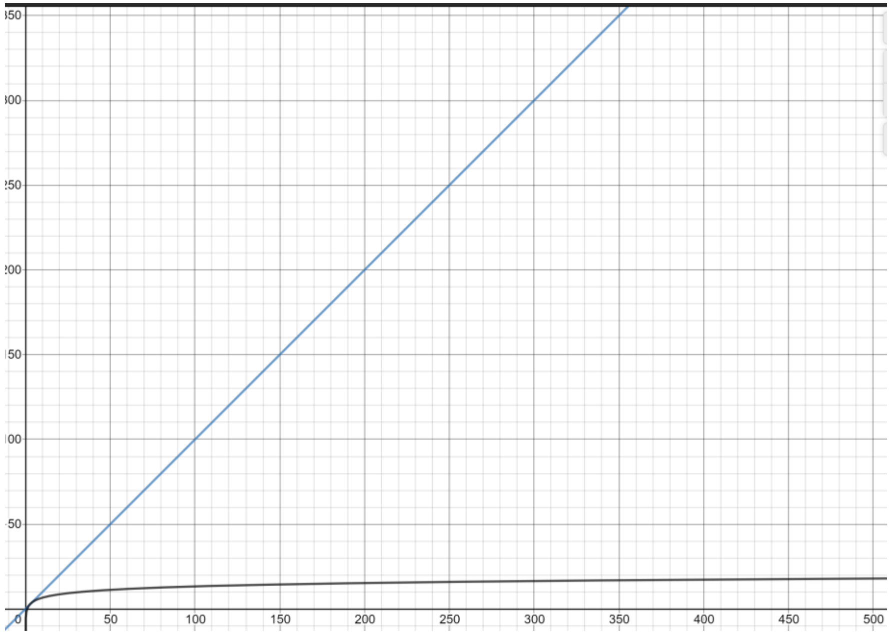
Fast Exponentiation Algorithm

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

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

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

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

$3^1\%7 = 3$

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

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

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

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

One More Example for Reference

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 a 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

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 $q \nmid a$