Week 4

Modular powers

We're about to do something cool, but we'll need to learn one more algorithm first: computing powers in modular arithmetic!

Remember, "a mod b" means "the remainder when you divide a by b".

Addition and multiplication work well with modular arithmetic: for example, $12 \times 13 = 156$ is the same as $2 \times 3 = 6$ modulo 10. Powers don't work well, though: for example, 12^{13} is 106993205379072, which is not the same as $2^3 = 8$ modulo 10.

What if you wanted to calculate 12^{13} mod 10 by hand? You could try to compute 12^{13} and then take the remainder modulo 10, but the numbers get really big and complicated. Instead, here's an alternative algorithm for computing powers.

To compute a^b mod q:

1. Take a new number T. It's going to start as 1, but we'll keep updating it throughout this algorithm.
2. Write b in binary, and read the digits from left to right.
   (a) If you see a 0, square T and take its remainder modulo q, and set T to be this new value.
   (b) If you see a 1, square T and take its remainder modulo q, then multiply by a and take the remainder modulo q again, and set T to be this value.
3. Keep doing this until you run out of digits of b. When this happens, the final result for T is the answer!

For example, let's use this to compute 12^{13} mod 10. We start with $T = 1$. In binary, 13 is 1101.

- The first digit is 1, so we square T ($1^2 = 1$) and take remainder modulo 10 (it's still just 1), then multiply it by 12 ($1 \times 12 = 12$) and take the remainder modulo 10 ($12 \bmod 10 = 2$).
- The next digit is 1 again, so we square T ($2^2 = 4$), take remainder modulo 10 ($4 \bmod 10 = 4$), multiply by 12 ($4 \times 12 = 48$) and take remainder modulo 10 ($48 \bmod 10 = 8$).
- The next digit is 0, so we square T ($8^2 = 64$) then take remainder modulo 10 (64 mod 10 is 4).
- Finally, the last digit is 1, so we square ($4^2 = 16$), reduce modulo 10 ($16 \bmod 10 = 6$), multiply by 12 ($6 \times 12 = 72$) and reduce modulo 10 again ($72 \bmod 10 = 2$).

We end up with 2 — which is the right answer!

Question 1. Use this method to compute:

(a) 4^5 mod 10. (5 is 101 in binary. Check this one by computing 4^5 by hand or with a calculator!)

(b) 3^{16} mod 7. (16 is 10000 in binary.)

(c) 6^{54} mod 25. (54 is 110110 in binary.)

(d) 25^{194} mod 30. (194 is 11000010 in binary.)

Question 2. Compute:

(a) $1^{13} \bmod 21$ (13 is 1101 in binary),

(b) $2^{13} \bmod 21$,

(c) $3^{13} \bmod 21$,

(d) $4^{13} \bmod 21$,

(e) $5^{13} \bmod 21$,

(f) $6^{13} \bmod 21$.

Can you guess what $7^{13} \bmod 21$ will be without calculating it…?

Question 3. *(Challenge.)* Why does this method work?

RSA encryption algorithm

We've finally reached the thing we've been building up towards — **RSA encryption**! It's a way to send coded information to someone, invented by Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman in 1977 (and also by Clifford Cocks in 1973, but he worked for the UK spy agency so his work was classified until 1997). It's used in many situations where people want to send secret information, like keeping your emails private and hiding your bank information when you buy things online.

This is a type of "public key encryption": you have some numbers, a "public key", that anyone can use to encrypt a message to you — but once it's encrypted, it can't be decrypted without using a "private key" that you keep secret.

Here's how it works! You can do the calculations by hand if you want, but I recommend using a calculator if you have one, or even writing some computer code if you know how.

Setup

- Choose two prime numbers, P and Q. **Keep P and Q secret!**

  If you're working by hand, I suggest picking numbers no bigger than about 50*; if you have a calculator, try numbers up to* 1000*; and if you're using a computer, the numbers can be as big as you want, as long as they're prime.*

- Set $N = P \times Q$. You don't need to keep N secret — feel free to shout it to your whole house or post it on Instagram or whatever.

- Set $\Phi = (P-1) \times (Q-1)$. (The "$\Phi$" symbol is the Greek letter "phi".) **Keep Φ secret!**

- Choose a number E that doesn't share any common factors with Φ, apart from 1 — check this with the Euclidean algorithm. You can pick something small, like 3 or 5 or 7. You can tell anyone what E is.

- Set D to be the multiplicative inverse of E modulo Φ — use the extended Euclidean algorithm for this. **Keep D secret!**

Encryption

The instructor will be sending a secret message to you.

- Tell me what N and E are. (This is your "public key", the information needed to encrypt the message. The other numbers are your secret "private key".)

- I'll pick a secret number M (the "M" stands for "Message") to send to you by code. The only restriction is that M needs to be less than N. (If you wanted to send a text message through RSA, you could turn it into a number with the rule A = 01, B = 02, ...)

- Next, I'll encrypt the message by calculating $C = M^E \bmod N$. The "C" stands for "Code": this number is the coded message that I'll send to you.

- Now it's your turn: to decrypt the message, calculate $C^D \bmod N$, using the power algorithm. If everything worked properly, this should be the secret message!

Question 4. Did it work?

Question 5. Send a message to me! My public key is:

$$N = 11303 \qquad \text{and} \qquad E = 5.$$

Why is RSA encryption secure?

The main idea behind the RSA algorithm is that the steps in the algorithm can be done quickly by a computer, but if you wanted to break the code, you'd have to factorise N, which takes a long time.

Question 6. Choose two secret prime numbers, P and Q. Calculate $N = P \times Q$, and type your N in the chat. Can you factorise anyone else's N (without asking a computer to do it for you)?

Here are the calculations involved in the RSA algorithm:

Finding some prime numbers	The fastest known method to check whether a number is prime takes approximately n^6 steps for an n-digit number.
Multiplication	The long-multiplication method that you probably learnt in elementary school takes approximately n^2 steps to multiply two n-digit numbers.
Subtracting 1	This takes n steps for an n-digit number in the worst-case scenario.
Checking that numbers don't share common factors besides 1	The Euclidean algorithm takes about n^2 steps for numbers with n digits.
Computing a multiplicative inverse	The extended Euclidean algorithm still takes about n^2 steps.
Calculating powers in modular arithmetic	The power algorithm we learnt today takes about n^3 steps.

Now, if you wanted to hack the algorithm, you'd need to figure out the secret numbers, by factorising N to figure out P and Q. Unfortunately for potential hackers, it's easy to multiply numbers together, but it's a lot harder to factorise them apart again, especially for really big numbers — the best known techniques for factorising a number with n digits take about 2^n steps.

Question 7. Plot the functions x, x^2, x^3, x^6 and 2^x on a graph, using `https://www.desmos.com/calculator`. Zoom out so you can see up to $x = 100$.

Which of these functions are bigger? Do you expect the smaller ones to catch up if you zoom out further? What does this tell you about RSA encryption? What else do you notice?

Try modifying these functions: for example, add "$\cdots + 7$" or "$3 \times \ldots$" to some of them. How do the functions change? What effect does this have on your answers to this question?

Question 8. *(Extra challenge.)* Why does the RSA algorithm work?

(Hint: Try calculating $M^\Phi \bmod N$ for different values of M, and look for a pattern. Explaining why this pattern happens is the hard part!)