Review from last time
- Polynomial time means $O(n^c)$ where $n =$ # bits in input = $\lceil \log_2(\text{input}) \rceil$, $c =$ const (2, 3, ...)
- So an algorithm that takes input $x$ and runs in time $O(x)$ is not polynomial.
- Addition/subtraction (right-to-left with carry/borrow) is $O(n)$.
- Elementary-school multiplication are $O(n^2)$ but we can improve:

Karatsuba $(x, y)$:
- multiply $x \times y$, return product.

$n = \max \# \text{bits in } x \text{ or } y$

if $n \leq 32$, return $x \times y$.

$nn = \lceil n/2 \rceil$  

- base case of recursion
- uses built-in machine arithmetic instructions

$p = \text{Karatsuba}(x_{h}, y_{h})$
$q = \text{Karatsuba}(x_{l}, y_{l})$
$r = \text{Karatsuba}(x_{h} + x_{l}, y_{h} + y_{l})$

return $p \times (nn + nn) + (r - p - q) \times nn + q$

Analysis:
- master theorem

$T(n) = 3T(n/2) + O(n)$

$= O(n^{\log_3 3})$

$\approx O(n^{1.58})$

$= O(n^{3/2})$
More polynomial time arithmetic problems

- Test if $x$ is prime (complicated, high exponent)
- Division, modulo: same time as multiply
- Faster multiplication: almost but not quite $O(n \log n)$
- Greatest common divisors (Euclid)

$$\gcd(x, y):$$
- if $x = 0$: return $y$
- if $y = 0$: return $x$
- base case
- return $\gcd(y, x \mod y)$

Binary numbers (shifts and subtractions, no multiplies)

- Modular inverse
  - given $x, y$: find $z$ such that $zx \equiv 1 \pmod{y}$
  - equivalently: $z = \frac{1}{x} \pmod{y}$
  - only possible when $\gcd(x, y) = 1$
  - Extended $\gcd \Rightarrow$ same run time

$$x \cdot x \mod y = 1$$
Exponentiation

Computing $x^y$ (or in Python notation: `x ** y`) cannot be a polynomial time operation.

Why? Answer is too big. It has $y \log_2 x$ bits. So just writing it down takes more than polynomial time.

Instead: $\text{powermod}(x, y, z) = \text{return } x^y \mod z$

Base cases:

- if $y == 0$: return 1
- if $y == 1$: return $x \% z$

$a = \text{powermod}(x, y/2, z)$

if $y$ is even:

- return $(a * a) \% z$
else:

- return $(a * a * x) \% z$

Each recursive call reduces #bits ($y$) by one.

So #levels of recursion $\leq n$

Time = $O(n \times \text{multiplication}) = \text{polynomial}$
Example: $2 \mod 7$
\[(64 \mod 7 = 1)\]

\[\text{powermod}(2, 6, 7)\]
\[a = \text{powermod}(2, 3, 7)\]
\[a = \text{powermod}(2, 1, 7)\]

Base case

\[\text{return: } 2 \mod 7\]

\[a = 2\]
\[y(3) \text{ is odd}\]

\[\text{return } 2 \ast 2 \ast 2 \mod 7 = 1\]

\[a = 1\]
\[y(6) \text{ is even}\]

\[\text{return } 1 \ast 1 \mod 7 = 1\]
Encryption

"Symmetric encryption": two people meet and share a secret key later use same key to encrypt & decrypt
(simple example: key = long seq. of random bits encryption = decryption = message xor key)
Flaw: have to meet ahead of time

Public-key cryptography:
instead of having separate encryption and decryption keys you publish encrypt key keep decrypt key secret
⇒ anyone else can send encrypted messages to you
RSA public key system

you: choose big random prime numbers p and q
choose random number e
publish: e, p*q
keep secret: p, q
d = \frac{1}{e} \mod ((p-1)*(q-1))

encrypt (m) = return m^e \mod (p*q)
decrypt (m) = return m^d \mod (p*q)

works for any message m that can be encoded as a binary number 0 ≤ m < p*q