# Algorithms: Review from last time

**DEFINITION 1** Let $f$ and $g$ be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O(g(x))$ if there are constants $C$ and $k$ such that

$$|f(x)| \leq C|g(x)|$$

whenever $x > k$. [This is read as "$f(x)$ is big-oh of $g(x)$."]

1. For what values of C and k (if any) is it the case that $x^2=O(100x^2+4)$?
2. For what values of C and k (if any) is it the case that $x^2=O(x^3+4)$?
3. Is $\log(x)=O(x)$?
4. Is $2x^2+x=O(x^2)$?
5. Is $x^2=O(2x^2+x)$?

## New stuff: Big Theta and Big Omega

**DEFINITION 2** Let $f$ and $g$ be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Omega(g(x))$ if there are positive constants $C$ and $k$ such that

$$|f(x)| \geq C|g(x)|$$

whenever $x > k$. [This is read as "$f(x)$ is big-Omega of $g(x)$."]

**DEFINITION 3** Let $f$ and $g$ be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $\Theta(g(x))$ if $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$. When $f(x)$ is $\Theta(g(x))$ we say that $f$ is big-Theta of $g(x)$, that $f(x)$ is of *order* $g(x)$, and that $f(x)$ and $g(x)$ are of the *same order*.

Basically if f(x)=O(g(x)) then g(x)=Ω(f(x)). And if f(x) is both big-O of g(x) ***and*** big-Omega of g(x) it is also "of the order of" g(x) and f(x)=Θ(g(x))

## Algorithm complexity (3.3)

We will generally use the big-Θ notation to describe the runtimes of algorithms in terms of their inputs.

Provide a reasonable big-Θ estimate for the run time of the following algorithms.

```
t := 0
for i := 1 to n
  for j := 1 to n
    t := t + i + j
```

```
i := 1
t := 0
while i ≤ n
    t := t + i
    i := 2i
```

```
m := 0
for i := 1 to n
  for j := i + 1 to n
      m := max(a_i a_j, m)
```

```
procedure closest-pair((x_1, y_1), (x_2, y_2), ..., (x_n, y_n): pairs of real numbers)
min= ∞
for i := 2 to n
      for j := 1 to i − 1
            if (x_j − x_i)^2 + (y_j − y_i)^2 < min then
                  min := (x_j − x_i)^2 + (y_j − y_i)^2
                  closest pair := ((x_i, y_i), (x_j, y_j))
return closest pair
```

Why do you think we tend to use big-O rather than big-Θ when discussing algorithms?

# Other things from section 3.3

## Terminology

It turns out people rarely say things like Θ(n). Instead they will say "its runtime is linear" or some such. Below is a list of terms you might hear when discussing complexity.

**TABLE 1 Commonly Used Terminology for the Complexity of Algorithms.**

| *Complexity* | *Terminology* |
|---|---|
| $\Theta(1)$ | ★ Constant complexity |
| $\Theta(\log n)$ | ★ Logarithmic complexity |
| $\Theta(n)$ | ★ Linear complexity |
| $\Theta(n \log n)$ | Linearithmic complexity |
| $\Theta(n^b)$ | Polynomial complexity |
| $\Theta(b^n)$, where $b > 1$ | ★ Exponential complexity |
| $\Theta(n!)$ | Factorial complexity |

The ones with ★ next to them are terms you should know by name. In addition you should be aware that $\Theta(n^2)$ is often just referred to as "quadratic complexity".

## Tractability

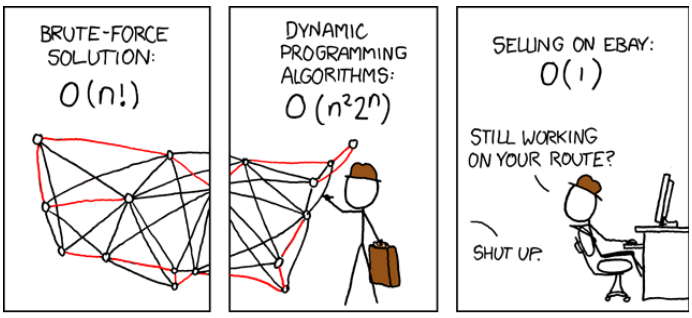
In general, we want to run algorithms that can actually solve problems. In section 3.2 the text showed that there exist problems which cannot be solved. But we also may encounter problems that are solvable, but aren't able to be solved in a reasonable period of time over a reasonable input set (yes, that was really vague on purpose).

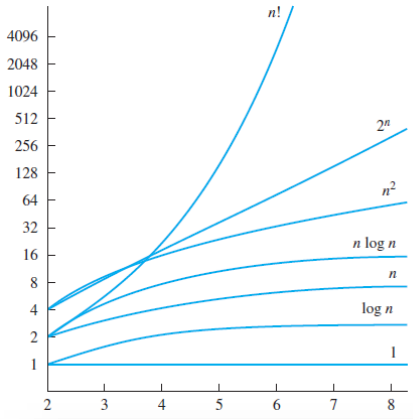
As a pretty arbitrary line, CS folks have chosen to call anything that can't be solved in polynomial time or less "intractable" meaning that it may be solvable but we can't really solve it in the general case. Again, this is pretty arbitrary. While $2^n$ certainly grows faster than $n^{100}$, it is actually smaller for quite a long time (~n=1000) and well past when we'd have a chance of doing the computation (at n=1000 both are around 10E300).

That said, we rarely encounter polynomial complexity algorithms where the exponent is much greater than 4 or 5. While we do hit exponential algorithms on a regular basis.

The "P=NP" question asks if a certain class of important algorithms (including things like minesweeper!) have a polynomial complexity or not. That one is still out…

# Algorithms Review—what we've done




1. Defined what made an algorithm.
    a. A finite sequence of precise instructions for doing something.
    b. Did a bit with the book's pseudo-code.
    c. Wrote one function, read a few.
2. Learned about Big-O notation.
    a. How it is defined.
    b. Why it's useful.
    c. Also learned Big-Θ and Big-Ω
3. Applied these notions to looking at real functions.

**procedure** $max(a_1, a_2, \ldots, a_n$: integers)
$max := a_1$
**for** $i := 2$ **to** $n$
  **if** $max < a_i$ **then** $max := a_i$
**return** $max$ {$max$ is the largest element}

Let $f$ and $g$ be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O(g(x))$ if there are constants $C$ and $k$ such that

$$|f(x)| \leq C|g(x)|$$

whenever $x > k$. [This is read as "$f(x)$ is big-oh of $g(x)$."]

One clarification. Our book prefers to use the language "$x^2+4$ is $O(x^2)$" rather than using the equal sign. That's a bit non-standard but not unheard of and probably clearer. Others also think of big-O as a set and might write "$x^2+4 \in O(x^2)$" though that is really non-standard from what I can find.[1]

## Questions/Review

1. Is the algorithm max (found above) O(n)? $O(n^2)$? O(log(n))?
2. Using the big-Θ notation, what order is the function max?
3. Describe why big-O/big-Ω is useful to computer scientists.

[1] **From Wikipedia:**
The statement "f(x) is O(g(x))" as defined above is usually written as f(x) = O(g(x)). Some consider this to be an abuse of notation, since the use of the equals sign could be misleading as it suggests a symmetry that this statement does not have. As de Bruijn says, O(x) = $O(x^2)$ is true but $O(x^2)$ = O(x) is not. Knuth describes such statements as "one-way equalities", since if the sides could be reversed, "we could deduce ridiculous things like n = $n^2$ from the identities n = $O(n^2)$ and $n^2$ = $O(n^2)$." For these reasons, it would be more precise to use set notation and write f(x) ∈ O(g(x)), thinking of O(g(x)) as the class of all functions h(x) such that |h(x)| ≤ C|g(x)| for some constant C. However, the use of the equals sign is customary. Knuth pointed out that "mathematicians customarily use the = sign as they use the word 'is' in English: Aristotle is a man, but a man isn't necessarily Aristotle.

# Number theory (Chapter 4)

The part of mathematics devoted to the study of the set of integers and their properties is known as number theory. We will hit a number of topics including cryptography and pseudorandom number generation.

## Divisibility and Modular Arithmetic (4.1)

When one integer is divided by a (non-zero) integer the result may be an integer. For example 7/2 is 3.5 while 8/2 is 4. We say "$a$ divides $b$" if $b$ divided by $a$ results in an integer. There are other terms such as "factor", "divisor" and "multiple" that are often used in this context. Our text uses the following:

> If $a$ and $b$ are integers with $a \neq 0$, we say that $a$ *divides* $b$ if there is an integer $c$ such that $b = ac$, or equivalently, if $\frac{b}{a}$ is an integer. When $a$ divides $b$ we say that $a$ is a *factor* or *divisor* of $b$, and that $b$ is a *multiple* of $a$. The notation $a \mid b$ denotes that $a$ divides $b$. We write $a \nmid b$ when $a$ does not divide $b$.

### Questions

1. Is a|b the same as $\exists c(ac = b)$ assuming $a, b, c \in \mathbb{Z}$?
2. Does 3|12?
3. Does 12|3?
4. Does 0|4?
5. Does 4|0?

### Divisibility and addition

$a, b, c, n, m \in \mathbb{Z}$ where $a \neq 0$, $\forall m, n\ (\ (a|b) \wedge (a|c) \rightarrow a|(mb + nc)$

What does the above mean? If a=4, b=8 and c=12, what do we know?

We'll prove that later…

## The Division "Algorithm"

**THE DIVISION ALGORITHM** Let $a$ be an integer and $d$ a positive integer. Then there are unique integers $q$ and $r$, with $0 \leq r < d$, such that $a = dq + r$.

The key notion here is that there is a *unique* value for r and q as long as $0 \leq r < d$. The function which generates q is called "**div**". The function which generates r is called **mod**.

### Questions

1. If a=20 and d=3, what are q and r? (same questions: What is 20 div 3 and what is 20 mod 3?)
2. What is -20 mod 3?
3. What is -20 div 3?
4. Why did I put "algorithm" in quotes?

## Modular arithmetic

We introduce the notion of "congruent" numbers. a is congruent to b modulo m if a mod m=b mod m. We (re)use the symbol ≡ to indicate congruency. So we'd write a≡b (mod m).

Huh?

10 mod 3=1 and 4 mod 3=1. Thus we could say that 10 is congruent to 4 mod 3. Or $10 \equiv 4 \pmod 3$

### Questions

1. Is $10 \equiv 4 \pmod 4$?
2. Is $10 \equiv 4 \pmod 2$?
3. For what positive values of m is $10 \equiv 4 \pmod m$?

The text uses an alternative (but equivalent) definition of congruence.

> If $a$ and $b$ are integers and $m$ is a positive integer, then $a$ is *congruent to $b$ modulo $m$* if $m$ divides $a - b$. We use the notation $a \equiv b \pmod m$ to indicate that $a$ is congruent to $b$ modulo $m$. We say that $a \equiv b \pmod m$ is a **congruence** and that $m$ is its **modulus** (plural **moduli**). If $a$ and $b$ are not congruent modulo $m$, we write $a \not\equiv b \pmod m$.

And that results in the following:

> Let $m$ be a positive integer. The integers $a$ and $b$ are congruent modulo $m$ if and only if there is an integer $k$ such that $a = b + km$.

Let's prove that.

And let's prove this:

Let $m$ be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then

$$a + c \equiv b + d \pmod{m} \quad \text{and} \quad ac \equiv bd \pmod{m}.$$

And now this:

Let $m$ be a positive integer and let $a$ and $b$ be integers. Then

$$(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$$

and

$$ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m.$$

## Representation of Integers (4.2)

This section mainly covers material we've already touched on—the idea that we can work in different bases.

**TABLE 1** Hexadecimal, Octal, and Binary Representation of the Integers 0 through 15.

| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hexadecimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| Octal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| Binary | 0 | 1 | 10 | 11 | 100 | 101 | 110 | 111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

1. Convert $10101_2$ (that subscript means "base 2") to base 10.
2. Convert $10F_{16}$ to base 10 (base 16 is often called "hexadecimal" or "hex")
3. Convert 120 to base 2 (notice we generally don't state the base when working in base 10)
4. Convert $101101101_2$ to base 8. (hint: there is an easy way to do this!)
5. Add $1110_2$ and $1011_2$
6. Add $1010_2$ and $0111_2$

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1} \\ 1\ 1\ 1\ 0 \\ +\ 1\ 0\ 1\ 1 \\ \hline 1\ 1\ 0\ 0\ 1 \end{array}$$

## Modular Exponentiation

In cryptography it is important to be able to find $b^n$ **mod** $m$ efficiently, where $b$, $n$, and $m$ are large integers. It is impractical to first compute $b^n$ and then find its remainder when divided by $m$ because $b^n$ will be a huge number. Instead, we can use an algorithm that employs the binary expansion of the exponent $n$.[2]

OK, this gets tricky. What we are going to do is notice that if we raise some number b to the $n^{th}$ power, we can consider the binary representation of n as ($a_{k-1}$, … $a_1$, $a_0$). So if n=12 we could consider $1100_2$. Consider the claim that

$$b^n = b^{a_{k-1}\cdot 2^{k-1}+\cdots+a_1\cdot 2+a_0} = b^{a_{k-1}\cdot 2^{k-1}} \cdots b^{a_1\cdot 2} \cdot b^{a_0}.$$

In our case (n=12) we are saying that $b^{12}=b^8*b^4$ which is clearly true.

[2] Text from page 253 of Rosen

So what are going to do is take advantage of this

**procedure** *modular exponentiation*($b$: integer, $n = (a_{k-1}a_{k-2}\ldots a_1a_0)_2$, $m$: positive integers)
$x := 1$
$power := b$ **mod** $m$
**for** $i := 0$ **to** $k - 1$
  **if** $a_i = 1$ **then** $x := (x \cdot power)$ **mod** $m$
  $power := (power \cdot power)$ **mod** $m$
**return** $x$ {$x$ equals $b^n$ **mod** $m$}

**EXAMPLE 12** Use Algorithm 5 to find $3^{644}$ **mod** 645.

*Solution:* Algorithm 5 initially sets $x = 1$ and $power = 3$ **mod** 645 $= 3$. In the computation of $3^{644}$ **mod** 645, this algorithm determines $3^{2^j}$ **mod** 645 for $j = 1, 2, \ldots, 9$ by successively squaring and reducing modulo 645. If $a_j = 1$ (where $a_j$ is the bit in the $j$th position in the binary expansion of 644, which is $(1010000100)_2$), it multiplies the current value of $x$ by $3^{2^j}$ **mod** 645 and reduces the result modulo 645. Here are the steps used:

$i = 0$: Because $a_0 = 0$, we have $x = 1$ and $power = 3^2$ **mod** 645 $= 9$ **mod** 645 $= 9$;
$i = 1$: Because $a_1 = 0$, we have $x = 1$ and $power = 9^2$ **mod** 645 $= 81$ **mod** 645 $= 81$;
$i = 2$: Because $a_2 = 1$, we have $x = 1 \cdot 81$ **mod** 645 $= 81$ and $power = 81^2$ **mod** 645 $= 6561$ **mod** 645 $= 111$;
$i = 3$: Because $a_3 = 0$, we have $x = 81$ and $power = 111^2$ **mod** 645 $= 12{,}321$ **mod** 645 $= 66$;
$i = 4$: Because $a_4 = 0$, we have $x = 81$ and $power = 66^2$ **mod** 645 $= 4356$ **mod** 645 $= 486$;
$i = 5$: Because $a_5 = 0$, we have $x = 81$ and $power = 486^2$ **mod** 645 $= 236{,}196$ **mod** 645 $= 126$;
$i = 6$: Because $a_6 = 0$, we have $x = 81$ and $power = 126^2$ **mod** 645 $= 15{,}876$ **mod** 645 $= 396$;
$i = 7$: Because $a_7 = 1$, we find that $x = (81 \cdot 396)$ **mod** 645 $= 471$ and $power = 396^2$ **mod** 645 $= 156{,}816$ **mod** 645 $= 81$;
$i = 8$: Because $a_8 = 0$, we have $x = 471$ and $power = 81^2$ **mod** 645 $= 6561$ **mod** 645 $= 111$;
$i = 9$: Because $a_9 = 1$, we find that $x = (471 \cdot 111)$ **mod** 645 $= 36$.

This shows that following the steps of Algorithm 5 produces the result $3^{644}$ **mod** 645 $= 36$.

Let's see how we'd use this to find $5^{13}$ mod 3 (something a bit less painful).