Tuesday 17th March • 6 pm Lecture

THE UNIVERSITY OF AUCKLAND
FACULTY OF SCIENCE

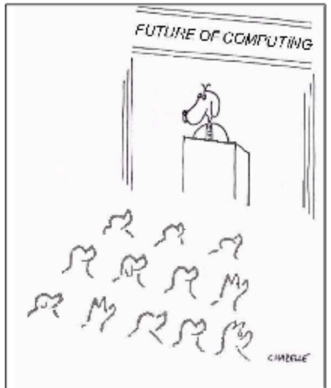
What an iPod, a Flock of Birds, and DNA have in common...

Professor Bernard Chazelle

Princeton University

Moore's Law holds that computing power doubles every 18 months. Most of the wonders of the computer age can be attributed directly to this law. Alas, its days are numbered. What then?

In this talk I will argue that the years ahead will usher in the era of the "Algorithm", a notion that might prove just as disruptive in this century as the revolution in the physical sciences was in the last century. I will discuss why algorithms are even more powerful than customarily believed but why they will not unleash their true potential until they become fully-fledged scientific tools and not just problem-solvers.



"Soon, my friends, you will look at a child's homework — and see nothing to eat."

Bernard Chazelle is Eugene Higgins Professor of Computer Science at Princeton University.

New Zealand Institute of Mathematics & its Applications

SLT1 Lecture Theatre • Ground Floor Building 303 • 38 Princes Street (by Albert Park)

CS 220
Complexity Measures

Notation Conventions

- **T(*n*)**: the "running time" for an algorithm having an ***n***-sized input, e.g., ***T(14) = 33, T(n) = 2n + 5***
- ***g(n)***: the formula for computing the running time for an algorithm having ***n***-sized input, e.g., ***g(n) = 2n + 5***
- ***O(n)***: a set of formulas that bound the running time for an algorithm as ***n*** gets very large
- ***f(n)***: the formula that represents how quickly the running time grows as ***n*** gets very large

Typical Complexity Curves

$T(n) \propto c$ constant

$T(n) \propto \log n$ logarithmic

$T(n) \propto n$ linear

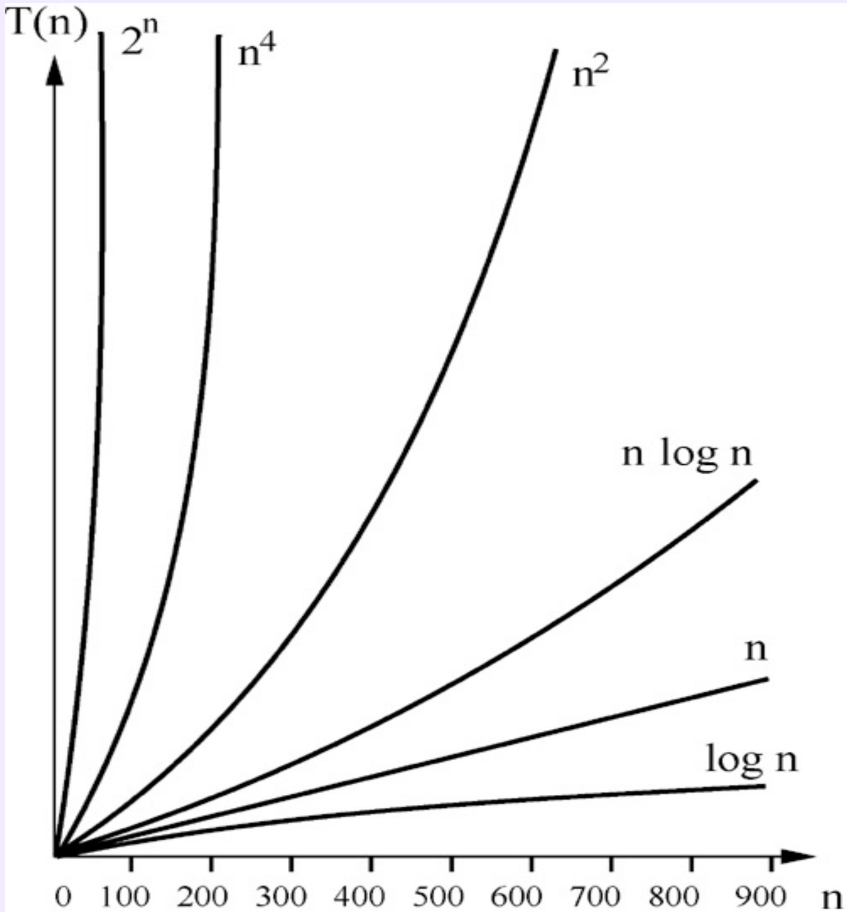
$T(n) \propto n \log n$

$T(n) \propto n^2$ quadratic

$T(n) \propto n^3$ cubic

$T(n) \propto n^k$ polynomial

$T(n) \propto 2^n$ exponential



Relative growth: $g(n) = f(n)/f(5)$

Function $f(n)$		Input size n: 5	25	125	625
Constant	1	1	1	1	1
Logarithm	$\log_5 n$	1	2	3	4
Linear	n	**1**	**5**	**25**	**125**
"n log n"	$n \log_5 n$	1	10	75	500
Quadratic	n^2	1	25 (5^2)	625 (5^4)	15,625 (5^6)
Cubic	n^3	1	125 (5^3)	15,625 (5^6)	1,953,125 (5^9)
Exponential	2^n	**1**	$\mathbf{2^{20} \approx 10^6}$	$\mathbf{2^{120} \approx 10^{36}}$	$\mathbf{2^{620} \approx 10^{187}}$

"Big-Oh" $O(\ldots)$: Formal Definition

Let $f(n)$ and $g(n)$ be nonnegative-valued functions defined on nonnegative integers n

The function $g(n)$ is $O(f(n))$ (read: $g(n)$ is **Big Oh** of $f(n)$) **iff** there exists a positive real constant c and a positive integer n_0 such that $g(n) \leq cf(n)$ for all $n > n_0$

– Notation "***iff***" is an abbreviation of "**if and only if**"

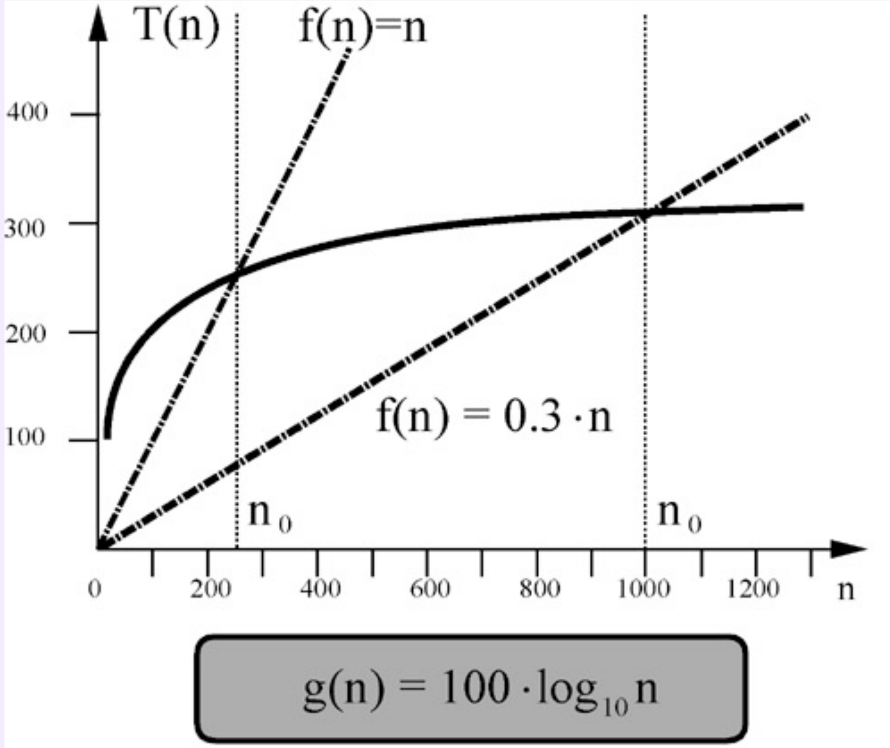
– *Example* 1.9 (p.15): $g(n) = 100\log_{10}n$ is $O(n)$

$\Leftarrow g(n) < n$ if $n > 238$ or $g(n) < 0.3n$ if $n > 1000$

$g(n)$ is $O(f(n))$, or $g(n) = O(f(n))$

$g(n)$ is $O(f(n))$ if:

- a constant $c > 0$ exists such that $cf(n)$ grows faster than $g(n)$ for all $n > n_0$
- To prove that some function $g(n)$ is $O(f(n))$ means to show for g and f such constants c and n_0 exist
- The constants c and n_0 are interdependent



$n > (n_0 = 238)$: $g(n) < (\ f(n) = n\ )$

$n > (n_0 = 1000)$: $g(n) < (\ f(n) = 0.3 \cdot n\ )$

“Big-Oh” $O(\ldots)$: Informal Meaning

- If $g(n)$ is $O(f(n))$, an algorithm with running time $g(n)$ runs **asymptotically** (i.e. for large n), **at most** as fast, to within a constant factor, as an algorithm with running time $f(n)$

$O(f(n))$ specifies an asymptotic upper bound, i.e. $g(n)$ for large n may approach closer and closer to $cf(n)$

Notation $g(n) = O(f(n))$ means actually $g(n) \in O(f(n))$, i.e. $g(n)$ is a member of the set $O(f(n))$ of functions increasing with the same or lesser rate as $n \to \infty$

Big Omega $\Omega(\ldots)$

- The function $g(n)$ is $\Omega(f(n))$ **iff** there exists a positive real constant c and a positive integer n_0 such that $g(n) \geq cf(n)$ for all $n > n_0$

  $\Omega(\ldots)$ is opposite to $O(\ldots)$ and specifies an <u>asymptotic lower bound</u>: if $g(n)$ is $\Omega(f(n))$ then $f(n)$ is $O(g(n))$

  Example 1: $5n^2$ is $\Omega(n) \Leftarrow 5n^2 \geq 5n$ for $n \geq 1$

  Example 2: $0.01n$ is $\Omega(\log n) \Leftarrow 0.01n \geq 0.5\log_{10}n$ for $n \geq 100$

Big Theta Θ(...)

- The function $g(n)$ is $\Theta(f(n))$ **iff** there exists two positive real constants c_1 and c_2 and a positive integer n_0 such that $c_1 f(n) \leq g(n) \leq c_2 f(n)$ for all $n > n_0$

$g(n)$ is $\Theta(f(n)) \Rightarrow$

$g(n)$ is $O(f(n))$ AND $f(n)$ is $O(g(n))$

Ex.: the same rate of increase for $g(n) = n + 5n^{0.5}$ and $f(n) = n$

$\Rightarrow \; n \leq n + 5n^{0.5} \leq 6n$ for $n > 1$

Comparisons: Two Crucial Ideas

- Exact running time function is unimportant since it can be multiplied by an arbitrary positive constant.
- Two functions are compared ***asymptotically***, for large n, and not near the origin
  - If the constants c involved are very large, then the asymptotical behaviour is of no practical interest!
  - To prove that $g(n)$ is **not** $O(f(n))$, $\Omega(f(n))$, or $\Theta(f(n))$ we have to show that the desired constants do not exist, i.e. lead to a contradiction

Example 1.12, p.17

Linear function $g(n) = an + b$; $a > 0$, is $O(n)$

To prove, we form a chain of inequalities:

$$g(n) \le a\,n + |b| \le g(n) \le (a + |b|) \cdot n \text{ for all } n \ge 1$$

Do not write $O(2n)$ **or** $O(an + b)$ **as this means still** $O(n)$**!**

$O(n)$ - running time:

$T(n) = 3n + 1$ $\quad$ $T(n) = 10^8 + n$

$T(n) = 50 + 10^{-8}\,n$ $\quad$ $T(n) = 10^6\,n + 1$

Remember that "Big-Oh" describes an "asymptotic behaviour" for large problem sizes

Example 1.13, p.17

Polynomial $P_k(n) = a_k\, n^k + a_{k-1}n^{k\text{-}1} + \ldots + a_1 n + a_0;\ a_k > 0$,
is $\mathrm{O}(n^k) \Leftarrow P_k(n) \le (a_k + |a_{k-1}| + \ldots + |a_0|)\, n^k\ ;\ n \ge 1$

<u>Do not write</u> $\mathrm{O}(\mathrm{P}_k(n))$ **<u>as this means still</u>** $\mathrm{O}(n^k)$**!**

<u>$\mathrm{O}(n^k)$</u> - running time:

- $\mathrm{T}(n) = 3n^2 + 5n + 1$ is $\mathrm{O}(n^2)$ <u>Is it also $\mathrm{O}(n^3)$?</u>
- $\mathrm{T}(n) = 10^{-8}\, n^3 + 10^8\, n^2 + 30$ is $\mathrm{O}(n^3)$
- $\mathrm{T}(n) = 10^{-8}\, n^8 + 1000n + 1$ is $\mathrm{O}(n^8)$

$T(n) = P_k(n) \Rightarrow O(n^m),\ m \ge k;\ \ \Theta(n^k);\ \ \Omega(n^m);\ m \le k$

Example 1.14, p.17

Exponential $g(n) = 2^{n+k}$ is $O(2^n)$: $2^{n+k} = 2^k \cdot 2^n$ for all n

Exponential $g(n) = m^{n+k}$ is $O(l^n)$, $l \geq m > 1$:

$$m^{n+k} \leq l^{n+k} = l^k \cdot l^n \text{ for all } n, k$$

A “brute-force” search for the best combination of n interdependent binary decisions by exhausting all the 2^n possible combinations has exponential time complexity! Therefore, **try to find a more efficient way of solving the decision problem** with $n \geq 20 \ldots 30$

Example 1.15, p.17

- Logarithmic function $g(n) = \log_m n$ has the same rate of increase as $\log_2 n$ because

$$\log_m n = \log_m 2 \cdot \log_2 n \quad \text{for all } n, m > 0$$

Do not write $O(\log_m n)$ **as this means still** $O(\log n)$!

You will find later that the most efficient search for data in an ordered array has logarithmic time complexity