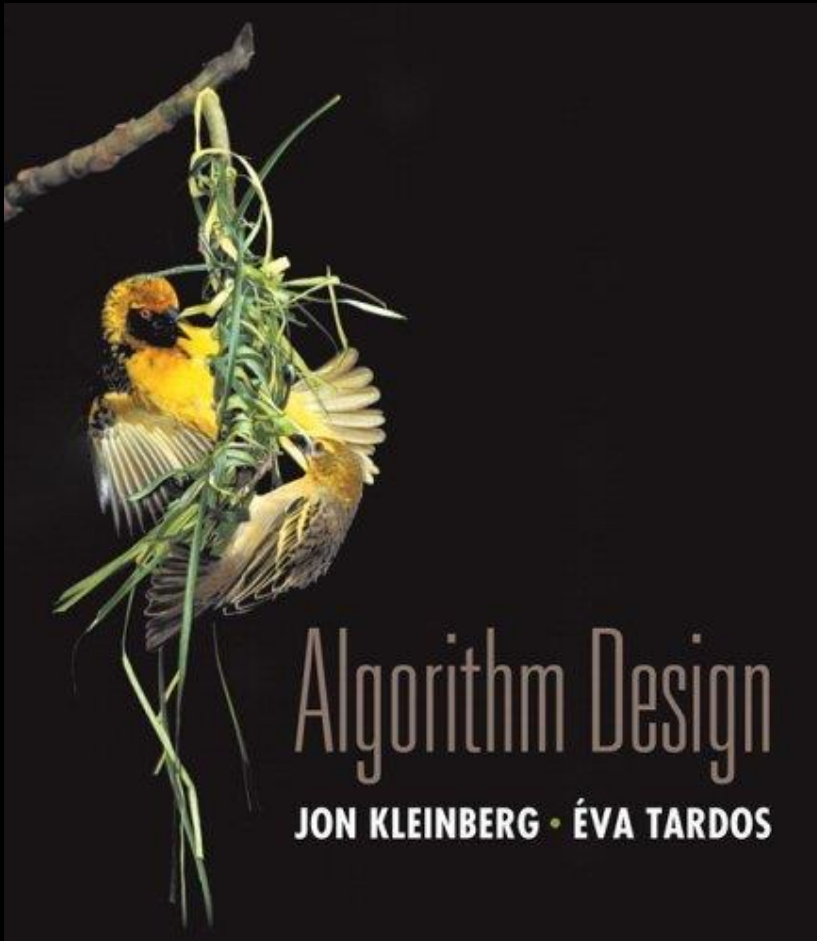


Chapter 2

Basics of Algorithm Analysis



Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

Announcements:

Office hours:

Prof. Brian Ziebart
1108 SEO
3-5pm Wednesdays

Wei Xing
1310 SEO
1:30-5:30pm Tuesdays

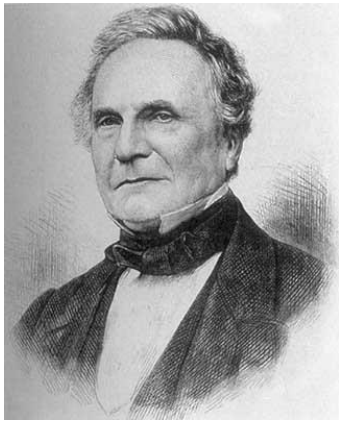
Enrollment: A few seats are now open. Please email me (bziebart@uic.edu) if you are still unable to register.

2.1 Computational Tractability

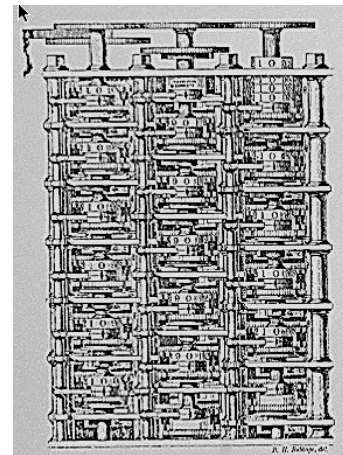
"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing." - *Francis Sullivan*

Computational Tractability

As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time? - *Charles Babbage*



Charles Babbage (1864)



Analytic Engine (schematic)

Computational Tractability

How fast is the algorithm?

Even for identical inputs, it will depend on:

- The clock speed of the processor
- The data structure implementation
- The instruction set of the processor / compiler
- The speed/size of the memory hierarchy
- Branch prediction in the processor
- And many other variables we'd rather not think about...

Can we characterize
algorithm speed without
the infrastructure
specifics?



[Non-]Polynomial-Time

Brute force. For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution.

- Typically takes 2^N time or worse for inputs of size N .
- Unacceptable in practice.

Desirable scaling property. When the input size doubles, the algorithm should only slow down by some constant factor.

There exists constants $c > 0$ and $d > 0$ such that on every input of size N , its running time is bounded by $c \underline{N^d}$ steps.

Def. An algorithm is **poly-time** if the above scaling property holds.

↖
choose $C = 2^d$

Different Types of Analysis

Worst case running time. Obtain bound on **largest possible** running time of algorithm on input of a given size N .

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Average case running time. Obtain bound on running time of algorithm on **random** input as a function of input size N .

- Hard (or impossible) to accurately model real instances by random distributions.
- Algorithm tuned for a certain distribution may perform poorly on other inputs.

Smoothed case running time. Obtain bound on **largest possible** running time of algorithm on **random** perturbation of any input as a function of input size N .

- Interpolates between the worst and average case
- Very hard to compute

Worst-Case Polynomial-Time

Def. Algorithm is **efficient** if its running time is polynomial.

Justification: **It really works in practice!**

- Although $6.02 \times 10^{23} \times N^{20}$ is technically poly-time, it would be useless in practice.
- In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions.

- Some poly-time algorithms do have high constants and/or exponents, and are useless in practice.
- Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare.

↑
simplex method
Unix grep

Why It Matters

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

2.2 Asymptotic Order of Growth

Asymptotic Order of Growth

Upper bounds. $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \leq c \cdot f(n)$.

Lower bounds. $T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \geq c \cdot f(n)$.

Tight bounds. $T(n)$ is $\Theta(f(n))$ if $T(n)$ is both $O(f(n))$ and $\Omega(f(n))$.

Ex: $T(n) = 32n^2 + 17n + 32$. $\leq c n^2 \quad n \geq n_0$

- $T(n)$ is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$.
- $T(n)$ is not $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$.

What can we say about $T(n) = 15n!$ order of growth?

What can we say about $T(n) = \log(n!)$?

$$O(n^{n-1})$$

$$\leq n \cdot n \cdot n \cdot \dots \cdot n$$