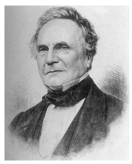
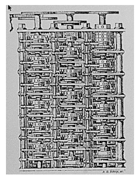
Computational Tractability

> 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)

1

What do we want from our performance measure?

- platform independent, implementation-detail independent → ignore constant factors, use big O notation when we talk about running time.
- instance independent → worst-case analysis (sometimes average case analysis)
- of predictive value with respect to increasing input size, tells us how algorithm scales → want to measure rate of growth of T(n) as function of n, the input size.

Asymptotic, worst – case analysis
Seek polynomial time algorithms

2

Worst-Case 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.

3

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.

n! for stable matching with n men and n women

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

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

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

choose $C = 2^d$

4

Worst-Case Polynomial-Time

Def. An 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.

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

Moore's Law

The prediction that transistor density and hence the speed of computers will double every 18 months or so.

- Based on observation of 1960-- 1965
- Has pretty much held for last 40 years

Does this provide disincentive to develop efficient (polynomial time) algorithms?

Moore's Law

Does Moore's Law provide disincentive to develop efficient (polynomial time) algorithms?

NO!!

Running time of alg	Max input size in time T	2x speedup	2^8x speedup

Moore's Law

Does Moore's Law provide disincentive to develop efficient (polynomial time) algorithms?

NO!!

Exponential algorithms make polynomially slow progress, while polynomial algorithms advance exponentially fast!

9

Asymptotic Analysis of Algorithms

In a nutshell:

- Suppresses constant factors (that are system dependent)
- Suppresses lower order terms (that are irrelevant for large inputs)

10

Asymptotic Order of Growth

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

Little oh. $T(n)$ is $o(f(n))$ if for all constants $c > 0$ there is $n_0 \geq 0$ such that for all $n \geq n_0$ we have $T(n) \leq c \cdot f(n)$.

Ex: $T(n) = 32n^2 + 17n + 32$.

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

11

Asymptotic Bounds for Some Common Functions

Polynomials. $a_0 + a_1 n + \dots + a_d n^d$ is $\Theta(n^d)$ if $a_d > 0$.

Polynomial time. Running time is $O(n^d)$ for some constant d independent of the input size n.

Logarithms. $O(\log_a n) = O(\log_b n)$ for any constants $a, b > 0$.

can avoid specifying the base

Logarithms. For every $x > 0$, $\log n = O(n^x)$.

log grows slower than every polynomial

Exponentials. For every $r > 1$ and every $d > 0$, $n^d = O(r^n)$.

every exponential grows faster than every polynomial

12