ALGORITHM ANALYSIS

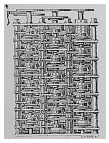
Computational Tractability

"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)

Define Algorithm Efficiency

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

How many possible solutions are there in the stable matching problem?
(In other words, how many possible perfect matchings are there? We're not worried about stability right now.)

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 "Exponential"
- Example: Stable matching: n! with n men and n women
  - If n increases by 1, what happens to the running time?

Polynomial-Time

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.

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

choose $C = 2^d$

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

10

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

What are alternatives to worst-case analysis?

11

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

12

Worst-Case Polynomial-Time

Def. An algorithm is **efficient** if its running time is *polynomial*

Justification: It really works in practice!

- In practice, poly-time algorithms that people develop almost always have low constants and low exponents
- Although $6.02 \times 10^{23} \times N^{20}$ is technically poly-time, it would be useless in practice
- 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

13

Why It Matters

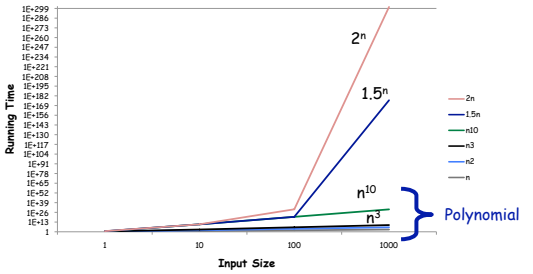
Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

Input Size	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

Polynomial

14

More Running Times

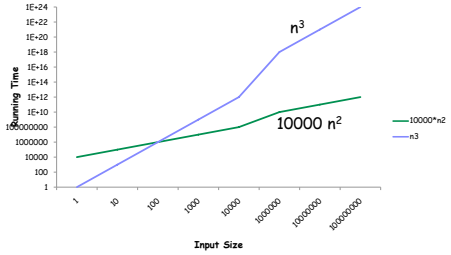


- Huge difference from polynomial to not polynomial
- Differences in runtime matter more as input size increases

Jan 19, 2009

15

More Running Times



As input size increases, n^3 dominates large constant * n^2
➔Care about running time as input size approaches infinity
➔Only care about highest-order term

Asymptotic Order of Growth: Upper Bounds

T(n) is the worst case running time of an algorithm

We say that T(n) is **O(f(n))**

- "order f(n)"

if there exist constants $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$

- i.e., sufficiently large n, c cannot depend on n

we have $T(n) \leq c \cdot f(n)$

- i.e., T(n) is bounded above by a constant multiple of f(n)

➢T is ***asymptotically upperbounded*** by f

Example: Upper Bound

$T(n) = pn^2 + qn + r$

- p, q, r are positive constants

For all $n \geq 1$,

$T(n) = pn^2 + qn + r \leq pn^2 + qn^2 + rn^2 = (p+q+r)\, n^2$

➔$T(n) \leq cn^2$, where $c = p+q+r$

➔$T(n) = O(n^2)$

Also correct to say that $T(n) = O(n^3)$

Asymptotic Order of Growth: Lower Bounds

Complementary to upper bound.

T(n) is **Ω(f(n))**

if there exist constants $\varepsilon > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$

- i.e., sufficiently large n, ε cannot depend on n

we have $T(n) \geq \varepsilon \cdot f(n)$

- i.e., T(n) is bounded below by a constant multiple of f(n)

➢T is ***asymptotically lowerbounded*** by f

Example: Lower Bound

$T(n) = pn^2 + qn + r$

- p, q, r are positive constants

Idea: Need to *deflate* the terms rather than inflate

For all $n \geq 0$,

$T(n) = pn^2 + qn + r \geq pn^2$

➔$T(n) \geq cn^2$, where $\varepsilon = p$

➔$T(n) = \Omega(n^2)$

Also correct to say that $T(n) = \Omega(n)$

Asymptotic Order of Growth

Tight bounds. T(n) is **Θ(f(n))** if T(n) is both O(f(n)) and Ω(f(n))

- The "right" bound

Practice: Asymptotic Order of Growth

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

- What are the upper bound, lower bound, and tight bound on T(n)?

22

Practice: Asymptotic Order of Growth

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

- $T(n)$ 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 $\Theta(n^3)$

23

Notation

Slight abuse of notation. $T(n) = O(f(n))$

- Asymmetric:
  - $f(n) = 5n^3$; $g(n) = 3n^2$
  - $f(n) = O(n^3) = g(n)$
  - but $f(n) \neq g(n)$.
- Better notation: $T(n) \in O(f(n))$

Meaningless statement. Any comparison-based sorting algorithm requires *at least* O(n log n) comparisons

- Use Ω for lower bounds

24

Properties

Transitivity

- If $f = O(g)$ and $g = O(h)$ then $f = O(h)$
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$

Proofs in book

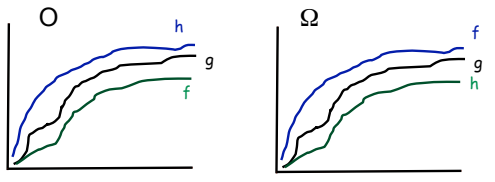
25

Properties

Transitivity

- If $f = O(g)$ and $g = O(h)$ then $f = O(h)$
- If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$
- If $f = \Theta(g)$ and $g = \Theta(h)$ then $f = \Theta(h)$

Proofs in book



26

Properties

Additivity

- If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$
- If $f = \Omega(h)$ and $g = \Omega(h)$ then $f + g = \Omega(h)$
- If $f = \Theta(h)$ and $g = O(h)$ then $f + g = \Theta(h)$

Proofs in book

27

Properties

Additivity

- If f = O(h) and g = O(h) then f + g = O(h)
- If f = Ω(h) and g = Ω(h) then f + g = Ω(h)
- If f = Θ(h) and g = O(h) then f + g = Θ(h)

Proofs in book

Sketch proof for O

- $f \leq c \cdot h$
- $g \leq d \cdot h$
- $f + g \leq c \cdot h + d \cdot h = (c + d)\, h = c' \cdot h$

28