Basics of Algorithms Analysis

CS 3AC3

Ryszard Janicki

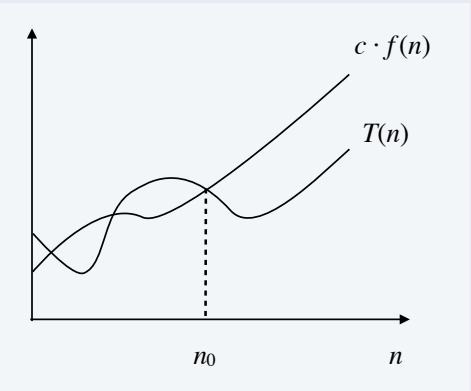
Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Acknowledgments: Material based on *Algorithm Design* by Jon Kleinberg and Éva Tardos (Chapter 2)

Big-Oh notation

Definition (Upper bounds)

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



Example

$T(n) = 32n^2 + 17n + 1$.

- $T(n)$ is $O(n^2)$. $\leftarrow$ choose $c = 50, n_0 = 1$
- $T(n)$ is also $O(n^3)$.
- $T(n)$ is neither $O(n)$ nor $O(n \log n)$.

Typical usage. Insertion makes $O(n^2)$ compares to sort n elements.

Notational abuses

- **Equals sign.** $O(f(n))$ is a set of functions, but computer scientists often write $T(n) = O(f(n))$ instead of $T(n) \in O(f(n))$.

Example

Consider $f(n) = 5n^3$ and $g(n) = 3n^2$.

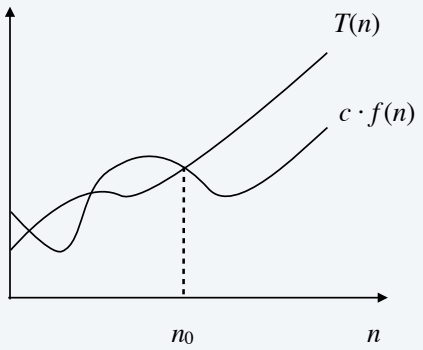
- We have $f(n) = O(n^3) = g(n)$.
- Thus, $f(n) = g(n)$.

- **Domain.** The domain of $f(n)$ is typically the natural numbers $\{0, 1, 2, \}$.
  - Sometimes we restrict to a subset of the natural numbers. Other times we extend to the reals.
- **Nonnegative functions.** When using big-Oh notation, we assume that the functions involved are (asymptotically) nonnegative.
- **Bottom line.** OK to abuse notation; not OK to misuse it.

Big-Omega notation

Definition (Lower bounds)

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



Example

$T(n) = 32n^2 + 17n + 1$.

- $T(n)$ is both $\Omega(n^2)$ and $\Omega(n)$. $\leftarrow$ choose $c = 32, n_0 = 1$
- $T(n)$ is also $O(n^3)$.
- $T(n)$ is neither $\Omega(n^3)$ nor $\Omega(n^3 \log n)$.

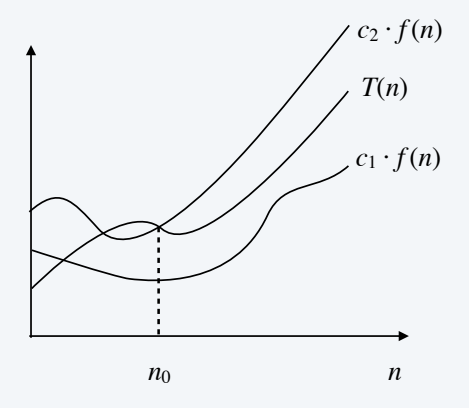
Typical usage. Any compare-based sorting algorithm requires $\Omega(n \log n)$ compares in the worst case.

Meaningless statement. Any compare-based sorting algorithm requires at least $O(n \log n)$ compares in the worst case.

Big-Theta notation

Definition (Tight bounds)

$T(n)$ is $\Theta(f(n))$ if there exist constants $c_1 > 0$, $c_2 > 0$ and $n_0 \geq 0$ such that $c_1 \cdot f(c) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$.



Example

$T(n) = 32n^2 + 17n + 1$.

- $T(n)$ is $\Theta(n^2)$. $\leftarrow$ choose $c_1 = 32, c_2 = 50, n_0 = 1$
- $T(n)$ is neither $\Theta(n)$ nor $\Theta(n^3)$.

Typical usage. Mergesort makes $\Omega(n \log n)$ compares to sort n elements.

Useful facts

Proposition

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$, *then* $f(n)$ *is* $\Theta(g(n))$.

Proof.

By definition of the limit, there exists n_0 such such that for all $n \geq n_0$

$$\frac{1}{2}c < \frac{f(n)}{g(n)} < 2c$$

- Thus, $f(n) \leq 2cg(n)$ for all $n \geq n_0$, which implies $f(n)$ is $O(g(n))$.
- $f(n) \geq \frac{1}{2}cg(n)$ for all $n \geq n_0$, which implies $f(n)$ is $\Omega(g(n))$. □

Proposition

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, *then* $f(n)$ *is* $O(g(n))$.

Asymptotic bounds for some common functions

- **Polynomials.** Let $T(n) = a_0 + a_1 n + \ + a_d n^d$ with $a_d > 0$. Then, $T(n)$ is $\Theta(n^d)$.

  Proof. $\lim_{n\to\infty} \dfrac{a_0 + a_1 n + \ + a_d n^d}{n^d} = a_d > 0.$

- **Logarithms.** *Theta*$(\log_a n)$ is $\Theta(\log_b n)$ for any constants $a, b > 0$.

  Proof. Since $\log_a n = \dfrac{\log_n n}{\log_b a}$.

- **Exponentials and polynomials.** For every $r > 1$ and every $d > 0$, n^d is $O(r^n)$.

  Proof. Since $\lim_{n\to\infty} \dfrac{n^d}{r^d} = 0.$

Big-Oh notation with multiple variables

Definition (Upper bounds)

$T(m,n)$ is $O(f(m,n))$ if there exist constants $c > 0$, $m^0 \geq 0$, and $n_0 \geq 0$ such that $T(m,n) \leq c \cdot f(m,n)$ for all $n \geq n_0$ and $m \geq m_0$.

Example

$T(m,n) = 32mn^2 + 17mn + 32n^3$.

- $T(m,n)$ is both $O(mn^2 + n^3)$ and $O(mn^3)$.
- $T(m,n)$ is neither $O(n^3)$ nor $O(mn^2)$.

Typical usage. Breadth-first search takes $O(m+n)$ time to find the shortest path from s to t in a digraph (*recall CS/SE 2C03*).

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