Asymtotic Analysis

Data Science
Gianmaria Silvello

An algorithm which is asymptotically better than another will perform better for all but very small inputs $\rightarrow n \geq n_0$ where n_0 is a small ad-hoc selected value.

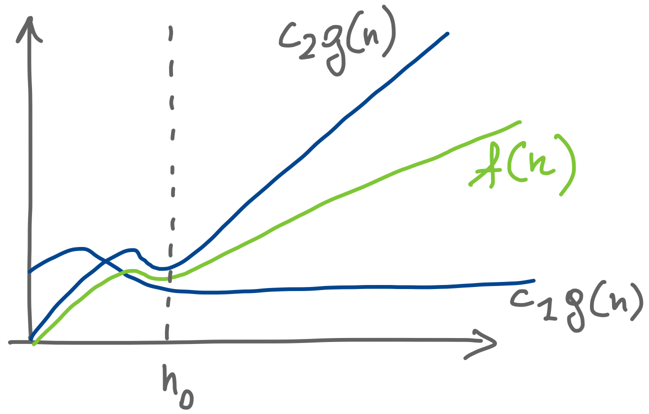
The asymptotic running time is defined in terms of functions $f(n)$ and $g(n)$ whose domains are in the set of naturals $\mathbb{N}_0 = \{0, 1, \dots\}$

We always consider the worst case running time $T(n)$

Big-theta Θ notation

$\Theta(g(n)) = \{ f(n)$: there exists positive constants c_1, c_2 and n_0 such that $\emptyset \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0 \}$

Alternative definition: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$, where $0 < c < +\infty$, if such c does exist, then $f(n) \in \Theta(g(n))$



we say that $f(n)$ can be sandwiched between $c_1 g(n)$ and $c_2 g(n)$

$g(n)$ defines lower and upper bounds of $f(n)$

↳ we say that $f(n)$ IS IN $g(n)$

NOT $f(n) = \Theta(g(n))$

allowed notation ABUSE

Exercise

Let us show that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

We have to show that $\emptyset \leq c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$ for $n \geq n_0$

we divide by $n^2 \Rightarrow$

$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$, for $n \geq 1$, $\frac{1}{2} - \frac{3}{n} \geq c_2 \Rightarrow n \to \infty \Rightarrow \frac{1}{2} \leq c_2$

$n = 7$ $\quad c_1 \leq \frac{1}{2} - \frac{3}{7} \Rightarrow c_1 \leq \frac{7-6}{14} \Rightarrow c_1 \leq \frac{1}{14}$

$\Rightarrow c_1 = \frac{1}{14}$, $c_2 = \frac{1}{2}$ and $n_0 = 7$ the bound is verified

other choices of constants exist, but we just need to find one.

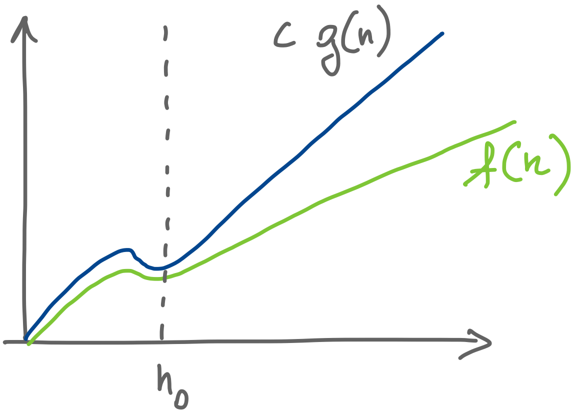
we can also prove the bound by contraddiction.
ab absurdo suppose that c_2 and n_0 exist such that $n^3 \le c_2 n^2$
but dividing by n^2 leads to $n \le c_2$ which cannot hold
since c_2 is a constant

Intuitively, we can say that for tight bounds (Θ), low order terms can be dropped because they are insignificant for arbitrarly large n ($n \ge n_0$) $\Rightarrow$

$$T(n) = \sum_{i=0}^{d} a_i n^i = \Theta(n^d)$$

Θ defines tight bounds, when we have only an upper bound for $f(n)$
we write $f(n) \in O(g(n))$ ⇝ O ⇝ big-oh

Definition: $O(g(n)) = \{f(n)$: there exists positive constants c and n_0 such that $0 \le f(n) \le c\,g(n)$, $\forall\ n \ge n_0\}$



$\Theta(g(n)) \subseteq O(g(n))$

if $f(n) \in \Theta(g(n))$ then also $f(n) \in O(g(n))$ but not vice versa.

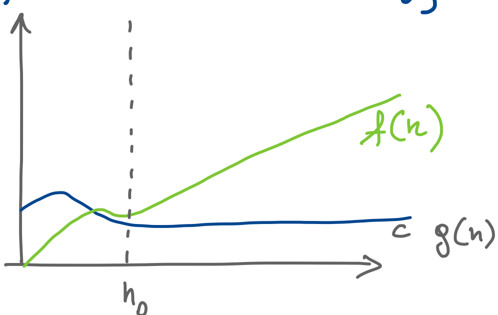
$f(n) = n + 4 \in O(n^2)$

true since $f(n) \subseteq O(n^2)$

Since O defines an upper bound when we use it to define the running time of an algorithm for the worst-case, we have the upper bound valid for all cases...

As O defines an upper bound of the running time, Ω (big-omega) defines a lower bound aka an asymptotic lower bound.

$\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le c\,g(n) \le f(n)$ for all $n > n_0\}$.



Theorem

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

As an example the running time of insertion sort belongs to $O(n^2)$ and $\Omega(n)$. These bound are as tight as possible since $\Omega(n^2)$ is not true for insertion sort. Nevertheless, we can say that the worst case of insertion sort belongs to $\Omega(n^2)$.

little-oh (o) notation

An asymptotic upper bound O might be tight, e.g. $2n^2 = O(n^2)$ or not tight, e.g. $2n = O(n^2)$.
We use the little-oh notation to indicate asymptotic upper bounds that are not tight.

$$o(g(n)) = \{f(n) : \text{for any positive constant } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\}.$$

Big-oh and little-oh definitions are similar, the difference is that in big-oh $f(n) = O(g(n))$ the bound $0 \le f(n) \le cg(n)$ holds <u>for all</u> constants $c > 0$, but for little-oh it holds <u>for SOME</u> constants $c > 0$.

little omega ω is defined likewise.

Exercise

Show that for any real constants a and b, where $b > 0$
$(n+a)^b = \Theta(n^b)$.

We need to show that $0 \le c_1 n^b \le (n+a)^b \le c_2 n^b$ for all $n \ge n_0$
we have to find $\boxed{\text{any}}$ c_1, c_2, n_0 for which the inequality holds

Note that $n + a \le n + |a| \le 2n$ when $|a| \le n$
and
$n + a \ge n - |a| \ge \frac{1}{2} n$ when $|a| \le \frac{1}{2}$

Thus, when $n \ge 2|a|$

$0 \le \frac{1}{2} n \le n + a \le 2n$ Since $b > 0$ the inequality still holds when all parts are raised to the power b:

$$0 \le \left(\frac{1}{2} n\right)^b \le (n+a)^b \le (2n)^b \qquad 0 \le \left(\frac{1}{2}\right)^b n^b \le (n+a)^b \le 2^b n^b$$

Thus, $c_1 = \left(\frac{1}{2}\right)^b$, $c_2 = 2^b$ and $n_0 = 2|a|$ satisfy the definition.

Exercise

Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

Intuitively, we can answer (1) Yes and (2) No. because $T(n) = \sum_{i=1}^{d} n^i = O(n^d)$

To prove the first bound we have to find a $c>0$ and a $n_0>0$ such that

$$2^{n+1} \leq c\, 2^n \quad \forall n \geq n_0$$

$$2^{n+1} = 2 \cdot 2^n \Rightarrow 2 \leq c \Rightarrow c = 2 \quad \forall n$$, which holds for $n_0 = 1$

The second bound is

$$2^{2n} \leq c\, 2^n$$

$$2^n \cdot 2^n \leq c\, 2^n \Rightarrow \underline{2^n \leq c}$$

↳ There is no possible constant greater than 2^n $\forall n$. → contradiction.

Exercise

Indicate, for each pair of expressions (A, B), whether A is $O, o, \Omega, \omega, \Theta$ of B. Assume that $k \geq 1$, $\epsilon > 0$ and $c > 1$ are constants.

	A	B	O	o	Ω	ω	Θ
a.	$\lg^k n$	n^{ϵ}	Yes	yes	no	no	no
b.	n^k	c^n	yes	yes	no	no	no
c.	$\sqrt{n}$	$n^{\sin n}$	no	no	no	no	no
d.	2^n	$2^{n/2}$	no	no	yes	yes	no