## 3 Asymptotic Growth

When we discussed Insertion Sort, we did a precise analysis of the running time and found that the worst-case is $k_3 n^2 + k_4 n - k_5$. The effort to compute all terms and the constants in front of the terms is not really worth it, because for large input the running time is dominated by the term $n^2$. Another good reason for not caring about constants and lower order terms is that the RAM model is not completely realistic anyway (not all operations cost the same).

$$k_3 n^2 + k_4 n - k_5 \sim n^2$$

Basically, we look at the running time of an algorithm when the input size $n$ is large enough so that constants and lower-order terms do not matter. This is called **aymptotic analysis of algorithms**.

Now we would like to formalize this idea (It is easy to see that $n+2 \sim n$, or that $4n^2+3n+10 \sim n^2$. But how about more complicated functions? say $n^n + n! + n^{\log\log n} + n^{1/logn}$).

$\Downarrow$

- We want to express **rate of growth** of a function:
  - the dominant term with respect to $n$
  - ignoring constants in front of it

| |
|---|
| $k_1 n + k_2 \sim n$ |
| $k_1 n \log n \sim n \log n$ |
| $k_1 n^2 + k_2 n + k_3 \sim n^2$ |

- We also want to formalize that a e.g. $n \log n$ algorithm is better than a $n^2$ algorithm.

$\Downarrow$

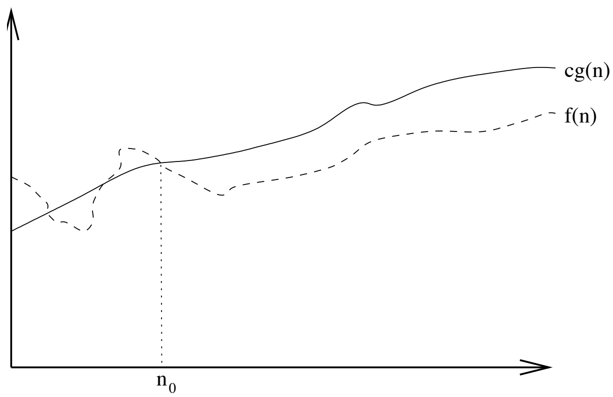
- $O$-notation (Big-$O$)
- $\Omega$-notation
- $\Theta$-notation
- you have probably seen it intuitively defined but we will now define it more carefully.

### 3.1 $O$-notation (Big-$O$)

$O(g(n)) = \{f(n) : \exists\ c, n_0 > 0 \text{ such that } f(n) \leq cg(n)\ \forall n \geq n_0\}$

- $O(\cdot)$ is used to asymptotically *upper bound* a function.

We think of $f(n) \in O(g(n))$ as corresponding to $f(n) \leq g(n)$.




Examples:

- $1/3n^2 - 3n \in O(n^2)$ because $1/3n^2 - 3n \leq cn^2$ if $c \geq 1/3 - 3/n$ which holds for $c = 1/3$ and $n > 1$.
- $k_1n^2 + k_2n + k_3 \in O(n^2)$ because $k_1n^2 + k_2n + k_3 < (k_1 + |k_2| + |k_3|)n^2$ and for $c > k_1 + |k_2| + |k_3|$ and $n \geq 1$, $k_1n^2 + k_2n + k_3 < cn^2$.
- $k_1n^2 + k_2n + k_3 \in O(n^3)$ as $k_1n^2 + k_2n + k_3 < (k_1 + k_2 + k_3)n^3$
- $f(n) = n^2/3 - 3n$, $g(n) = n^2$
  - $f(n) \in O(g(n))$
  - $g(n) \in O(f(n))$
- $f(n) = an^2 + bn + c$, $g(n) = n^2$
  - $f(n) \in O(g(n))$
  - $g(n) \in O(f(n))$
- $f(n) = 100n^2$, $g(n) = n^2$
  - $f(n) \in O(g(n))$
  - $g(n) \in O(f(n))$
- $f(n) = n, g(n) = n^2$
  - $f(n) \in O(g(n))$

Note: $O(\cdot)$ gives an upper bound of $f$, but not necessarilly tight:

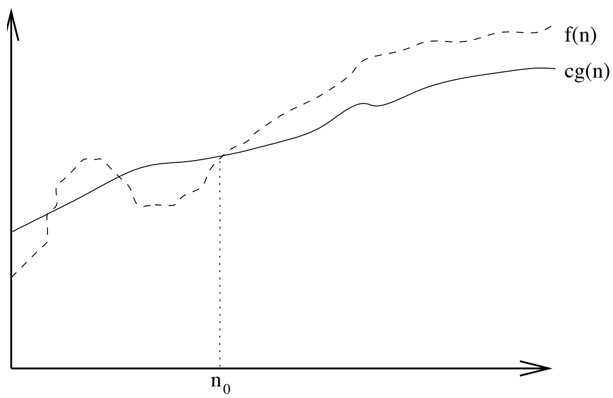
- $n \in O(n)$, $n \in O(n^2)$, $n \in O(n^3)$, $n \in O(n^{100})$

### 3.2 $\Omega$-notation (big-Omega)

$\Omega(g(n)) = \{f(n) : \exists\ c, n_0 > 0 \text{ such that } cg(n) \leq f(n)\ \forall n \geq n_0\}$

- $\Omega(\cdot)$ is used to asymptotically *lower bound* a function.

We think of $f(n) \in \Omega(g(n))$ as corresponding to $f(n) \geq g(n)$.




Examples:

- $1/3n^2 - 3n \in \Omega(n^2)$ because $1/3n^2 - 3n \geq cn^2$ if $c \leq 1/3 - 3/n$ which is true if $c = 1/6$ and $n > 18$.
- $k_1 n^2 + k_2 n + k_3 \in \Omega(n^2)$.
- $k_1 n^2 + k_2 n + k_3 \in \Omega(n)$ (lower bound!)
- $f(n) = n^2/3 - 3n$, $g(n) = n^2$
  - $f(n) \in \Omega(g(n))$
  - $g(n) \in \Omega(f(n))$
- $f(n) = an^2 + bn + c$, $g(n) = n^2$
  - $f(n) \in \Omega(g(n))$
  - $g(n) \in \Omega(f(n))$
- $f(n) = 100n^2$, $g(n) = n^2$
  - $f(n) \in \Omega(g(n))$
  - $g(n) \in \Omega(f(n))$
- $f(n) = n, g(n) = n^2$
  - $g(n) \in \Omega(f(n))$

Note: $\Omega(\cdot)$ gives a lower bound of $f$, but not necessarilly tight:

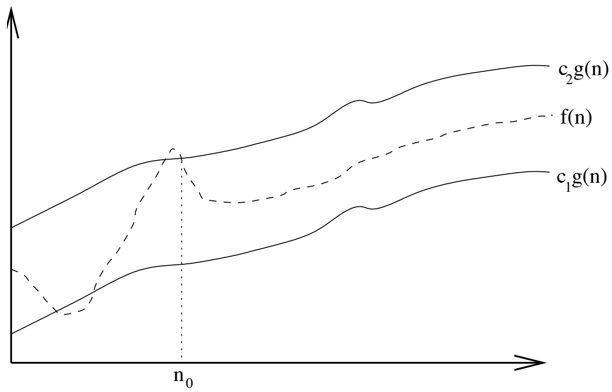
- $n \in \Omega(n)$, $n^2 \in \Omega(n)$, $n^3 \in \Omega(n)$, $n^{100} \in \Omega(n)$

### 3.3 $\Theta$-notation (Big-Theta)

$\Theta(g(n)) = \{f(n) : \exists\ c_1, c_2, n_0 > 0 \text{ such that } c_1 g(n) \le f(n) \le c_2 g(n)\ \forall n \ge n_0\}$

- $\Theta(\cdot)$ is used to asymptotically *tight bound* a function.

We think of $f(n) \in \Theta(g(n))$ as corresponding to $f(n) = g(n)$.




$\boxed{f(n) = \Theta(g(n)) \textit{ if and only if } f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))}$

It is easy to see (try it!) that:

**Theorem 1** *If $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$ then $f(n) \in \Theta(g(n))$.*

Examples:

- $k_1 n^2 + k_2 n + k_3 \in \Theta(n^2)$
- *worst case* running time of insertion-sort is $\Theta(n^2)$
- $6n \log n + \sqrt{n} \log^2 n \in \Theta(n \log n)$:
  - We need to find $n_0, c_1, c_2$ such that $c_1 n \log n \le 6n \log n + \sqrt{n} \log^2 n \le c_2 n \log n$ for $n > n_0$ $c_1 n \log n \le 6n \log n + \sqrt{n} \log^2 n \Rightarrow c_1 \le 6 + \frac{\log n}{\sqrt{n}}$. Ok if we choose $c_1 = 6$ and $n_0 = 1$. $6n \log n + \sqrt{n} \log^2 n \le c_2 n \log n \Rightarrow 6 + \frac{\log n}{\sqrt{n}} \le c_2$. Is it ok to choose $c_2 = 7$? Yes, $\log n \le \sqrt{n}$ if $n \ge 2$.
  - So $c_1 = 6$, $c_2 = 7$ and $n_0 = 2$ works.
- $n^2/3 - 3n \in O(n^2), n^2/3 - 3n \in \Omega(n^2) \longrightarrow n^2/3 - 3n \in \Theta(n^2)$
- $an^2 + bn + c \in O(n^2), an^2 + bn + c \in \Omega(n^2) \longrightarrow an^2 + bn + c \in \Theta(n^2)$
- $n \ne \Theta(n^2)$
- $f(n) = 6n \lg n + \sqrt{n} \lg^n$, $g(n) = n \lg n$

# 4 Growth Rate of Standard Functions

- Polynomial of degree $d$:

$$p(n) = \sum_{i=1}^{d} a_i \cdot n^i = \Theta(n^d)$$

  where $a_1, a_2, \ldots, a_d$ are constants (and $a_d > 0$).

- Any polylog grows slower than any polynomial.

$$\log^a n = O(n^b), \forall a > 0$$

- Any polynomial grows slower than any exponential with base $c > 1$.

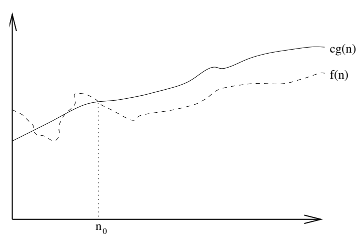
$$n^b = O(c^n), \forall b > 0, c > 1$$

Last time we looked at the problem of comparing functions (running times).

$$3n^2 \lg n + 2n + 1 \text{ vs. } 1000n \lg^{10} n + n \lg n + 5$$

Basically, we want to quantify how fast a function grows when $n \longrightarrow \infty$.
$\Downarrow$
**asymptotic** analysis of algorithms

More precisely, we want to compare 2 functions (running times) and tell which one is larger (grows faster) than the other. We defined $O, \Omega, \Theta$:




- $f$ is below $g \Leftrightarrow f \in O(g) \Leftrightarrow f \leq g$
- $f$ is above $g \Leftrightarrow f \in \Omega(g) \Leftrightarrow f \geq g$
- $f$ is both above and below $g \Leftrightarrow f \in \Theta(g) \Leftrightarrow f = g$

Example: Show that $2n^2 + 3n + 7 \in O(n^2)$

Upper and lower bounds are symmetrical: If $f$ is upper-bounded by $g$ then $g$ is lower-bounded by $f$ and we have:

$$f \in O(g) \Leftrightarrow g \in \Omega(f)$$

(Proof: $f \leq c \cdot g \Leftrightarrow g \geq \frac{1}{c} \cdot f$). Example: $n \in O(n^2)$ and $n^2 \in \Omega(n)$

An $O()$ upper bound is not a tight bound. Example:
$2n^2 + 3n + 5 \in O(n^{100})$
$2n^2 + 3n + 5 \in O(n^{50})$
$2n^2 + 3n + 5 \in O(n^3)$
$2n^2 + 3n + 5 \in O(n^2)$
Similarly, an $\Omega()$ lower bound is not a tight bound. Example:
$2n^2 + 3n + 5 \in \Omega(n^2)$
$2n^2 + 3n + 5 \in \Omega(n \log n)$
$2n^2 + 3n + 5 \in \Omega(n)$
$2n^2 + 3n + 5 \in \Omega(\lg n)$

An asymptotically **tight** bound for $f$ is a function $g$ that is equal to $f$ up to a constant factor: $c_1 g \leq f \leq c_2 g, \forall n \geq n_0$. That is, $f \in O(g)$ and $f \in \Omega(g)$.

Some properties:

- $f = O(g) \Leftrightarrow g = \Omega(f)$
- $f = \Theta(g) \Leftrightarrow g = \Theta(f)$
- reflexivity: $f = O(f), f = \Omega(f), f = \Theta(f)$
- transitivity: $f = O(g), g = O(h) \longrightarrow f = O(h)$

The growth of two functions $f$ and $g$ can be found by computing the limit $lim_{n \longrightarrow \infty} \frac{f(n)}{g(n)}$. Using the definition of $O, \Omega, \Theta$ it can be shown that :

- if $lim_{n \longrightarrow \infty} \frac{f(n)}{g(n)} = 0$: then intuitively $f < g \Longrightarrow f = O(g)$ and $f \neq \Theta(g)$.
- if $lim_{n \longrightarrow \infty} \frac{f(n)}{g(n)} = \infty$: then intuitively $f > g \Longrightarrow f = \Omega(g)$ and $f \neq \Theta(g)$.
- if $lim_{n \longrightarrow \infty} \frac{f(n)}{g(n)} = c, c > 0$: then intuitively $f = c \cdot g \Longrightarrow f = \Theta(g)$.

This property will be very useful when doing exercises.

## 5 Algorithms matter!

Sort 10 million integers on

- 1 GHZ computer (1000 million instructions per second) using $2n^2$ algorithm.
  - $\frac{2\cdot(10^7)^2 \ inst.}{10^9 \ inst.\ per\ second} = 200000$ seconds $\approx 55$ hours.
- 100 MHz computer (100 million instructions per second) using $50n \log n$ algorithm.
  - $\frac{50\cdot10^7\cdot\log 10^7 \ inst.}{10^8 \ inst.\ per\ second} < \frac{50\cdot10^7\cdot7\cdot3}{10^8} = 5\cdot7\cdot3 = 105$ seconds.

## 6 Comments

- The correct way to say is that $f(n) \in O(g(n))$. Abusing notation, people normally write $f(n) = O(g(n))$.

$$3n^2 + 2n + 10 = O(n^2), n = O(n^2), n^2 = \Omega(n), n\log n = \Omega(n), 2n^2 + 3n = \Theta(n^2)$$

- When we say "the running time is $O(n^2)$" we mean that the worst-case running time is $O(n^2)$ — best case might be better.
- When we say "the running time is $\Omega(n^2)$", we mean that the *best case* running time is $\Omega(n^2)$ — the worst case might be worse.
- Insertion-sort:
  - Best case: $\Omega(n)$
  - Worst case: $O(n^2)$
  - We can also say that worst case is $\Theta(n^2)$ because there exists an input for which insertion sort takes $\Omega(n^2)$. Same for best case.
  - Therefore the running time is $\Omega(n)$ and $O(n^2)$.
  - But, we cannot say that the running time of insertion sort is $\Theta(n^2)$!!!
- Use of $O$-notation makes it much easier to analyze algorithms; we can easily prove the $O(n^2)$ insertion-sort time bound by saying that both loops run in $O(n)$ time.
- We often use $O(n)$ in equations and recurrences: e.g. $2n^2 + 3n + 1 = 2n^2 + O(n)$ (meaning that $2n^2 + 3n + 1 = 2n^2 + f(n)$ where $f(n)$ is some function in $O(n)$).
- We use $O(1)$ to denote constant time.
- One can also define $o$ and $\omega$ (little-oh and little-omega):
  - $f(n) = o(g(n))$ corresponds to $f(n) < g(n)$
  - $f(n) = \omega(g(n))$ corresponds to $f(n) > g(n)$
  - we will not use them; we'll aim for tight bounds $\Theta$.
- Not all functions are asymptotically comparable! There exist functions $f, g$ such that $f$ is not $O(g)$, $f$ is not $\Omega(g)$ (and $f$ is not $\Theta(g)$).

# 7 Review of Log and Exp

- Base 2 logarithm comes up all the time (from now on we will always mean $\log_2 n$ when we write $\log n$ or $\lg n$).
  - Number of times we can divide $n$ by 2 to get to 1 or less.
  - Number of bits in binary representation of $n$.
  - Inverse function of $2^n = 2 \cdot 2 \cdot 2 \cdots 2$ ($n$ times).
  - Way of doing multiplication by addition: $\log(ab) = \log(a) + \log(b)$
  - Note: $\log n << \sqrt{n} << n$
- Properties:
  - $\lg^k n = (\lg n)^k$
  - $\lg \lg n = \lg(\lg n)$
  - $a^{\log_b c} = c^{\log_b a}$
  - $a^{\log_a b} = b$
  - $\log_a n = \frac{\log_b n}{\log_b a}$
  - $\lg b^n = n \lg b$
  - $\lg xy = \lg x + \lg y$
  - $\log_a b = \frac{1}{\log_b a}$