The efficiency analysis framework concentrates on the order of growth of an algorithm's basic count as the principal indicator of the algorithm's efficiency. $t(n)$ and $g(n)$ can be any non-negative functions defined on the set of natural numbers. $t(n)$ will be an algorithm's running time (usually indicated by its basic operation count $C(n)$) and $g(n)$ will be some simple function to compare the count with.

O-notation

A function $t(n)$ is said to be $O\big(g(n)\big)$, denoted $t(n) \in O\big(g(n)\big)$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n, i.e. if there exist some positive constant c and some non-negative integer n_0 such that

$$t(n) \leq cg(n) \qquad \text{for all } n \geq n_0$$

example $100n + 5 \in O(n^2)$

$$100n + 5 \leq 100n + n \text{ (for all } n \geq 5) \; = 101n \leq 101n^2$$

As the values of the constants c and n_0 require by the definition.
s

Ω-notation

A function $t(n)$ is said to be in $\Omega\big(g(n)\big)$, denoted $t(n) \in \Omega\big(g(n)\big)$, if $t(n)$ is bounded below some positive constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and some negative integer n_0 such that

$$t(n) \geq cg(n) \qquad \text{for all } n \geq n_0$$

example $n^3 \in \Omega(n^2)$

$$n^3 \geq n^2 \qquad \text{for all } n \geq 0$$

we can select $c = 1$ and $n_0 = 0$

Θ-notation

A function $t(n)$ is said to be in $\Theta\big(g(n)\big)$, denoted $t(n) \in \Theta\big(g(n)\big)$, if $t(n)$ is bounded both above and below by some positive constant multiples of $g(n)$ for all large n, i.e. if there exists some positive constants c_1 and c_2 and some non-negative integer n_0 such that

$$c_2 g(n) \leq t(n) \leq c_1 g(n) \qquad \text{for all } n \geq n_0$$

example $\frac{1}{2}n(n-1) \in \Theta(n^2)$

First we prove theright inequality (the upper bound)

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \qquad \text{for all } n \geq 0$$

Second, we prove the left inequality (the lower bound):

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n \qquad \text{for all } n \geq 2 = \frac{1}{4}n^2$$

We can select $c_2 = \frac{1}{4}$, $c_1 = \frac{1}{2}$ and $n_0 = 2$

Useful Property involving

Theorem

If $t_1(n) \in O\big(g_1(n)\big)$ and $t_2(n) \in O\big(g_2(n)\big)$, then $t_1(n) + t_2(n) \in O\big(\max\{g_1(n), g_2(n)\}\ \big)$

Proof

The proof extends to orders of growth the simple fact about four arbitrary real numbers a_1, b_1, a_2, b_2: if $a_1 \le b_1$ and $a_2 \le b_2$, then $a_1 + a_2 \le 2\max\{b_1, b_2\}$

Since $t_1(n) \in O\big(g_1(n)\big)$, there exists some positive constant c_1 and some non-negative integer n_1 such that

$$t_1(n) \le c_1 g_1(n) \qquad \text{for all } n \ge n_1$$

Similarily, since $t_2(n) \in O\big(g_2(n)\big)$

$$t_2(n) \le c_2 g_2(n) \qquad \text{for all } n \ge n_2$$

Let us denote $c_3 = \max\{c_1, c_2\}$, and consider $n \ge \max\{n_1, n_2\}$ so that we can use both inequalities. Adding them yiedls the following:

$$\begin{aligned} t_1(n) + t_2(n) &\le c_1 g_1(n) + c_2 g_2(n) \\ &\le c_3 g_1(n) + c_3 g_2(n) = c_3 \Big[g_1(n) + g_2(n)\Big] \\ &\le c_3 2\max\{g_1(n), g_2(n)\} \end{aligned}$$

Hence, $t_1(n) + t_2(n) \in O\big(\max\{g_1(n), g_2(n)\}\big)$, with the constants c and n_0 required by the O definition being $2c_3 = 2\max\{c_1, c_2\}$ and $\max\{n_1, n_2\}$, respectively.

What does this property imply for an algorithm that comprises two consecutively executed parts? It implies that the algorithm's overall efficiency is determined by the part with higher order of growth, i.e. its least efficient part.

Using Limits for Comparing Orders of Growth

A convenient method for computing orders of growth is based on computing the limit of the ratio of two functions. Three principal cases may arise

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0, & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n) \\ c, & \text{implies that } t(n) \text{ has the same order of growth as } g(n) \\ \infty, & \text{implies that } t(n) \text{ has a larger order of growth than } g(n) \end{cases}$$

Example 1

Compare the orders of growth of $\frac{1}{2}n(n-1)$ and n^2.

$$\begin{aligned} \lim_{n\to\infty} \frac{\frac{1}{2}n(n-1)}{n^2} &= \frac{1}{2} \lim_{n\to\infty} \frac{n^2 - n}{n^2} \\ &= \frac{1}{2} \lim_{n\to\infty} (1 - \frac{1}{n}) \\ &= \frac{1}{2} \end{aligned}$$

since the limit is equal to a positive constant, the functions have the same order of growth or symbolically, $\frac{1}{2}n(n-1) \in \Theta(n^2)$

Example 2
Compate the orders of growth of $\log_2 n$ and $\sqrt{n}$

$$\begin{aligned}
\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}} &= \lim_{n\to\infty} \frac{(\log_2 n)'}{(\sqrt{n})'} \\
&= \lim_{n\to\infty} \frac{(\log_2 e)\frac{1}{n}}{\frac{1}{2\sqrt{n}}} \\
&= 2\log_2 e \lim_{n\to\infty} \frac{1}{\sqrt{n}} \\
&= 0
\end{aligned}$$

Since the limit is equal to zero, $\log_2 n$has a smaller order of growth than $\sqrt{n}$

Example 3
Compare the orders of growth of $n!$ and 2^n.

$$\begin{aligned}
\lim_{n\to\infty} \frac{n!}{2^n} &= \lim_{n\to\infty} \frac{\sqrt{2\pi n}(\frac{n}{e})^n}{2^n} \\
&= \lim_{n\to\infty} \sqrt{2\pi n}\frac{n^n}{2^n e^n} \\
&= \lim_{n\to\infty} \sqrt{2\pi n}\left(\frac{n}{2e}\right)^n \\
&= \infty
\end{aligned}$$