# Algorithms & Computing

Philip Thomas K.

June 29, 2021

# Contents

**1 Algorithm Analysis** **2**
1.1 Growth Rate . . . . . 2
1.2 Asymptotic Notation . . . . . 3
1.2.1 Big-Oh . . . . . 3
1.2.2 Big-Omega . . . . . 3
1.2.3 Big-Theta . . . . . 4
1.2.4 Properties of Big-Oh Notation . . . . . 4
1.2.5 Tight Big-Oh Notation . . . . . 6

# Lecture No. 1

# Algorithm Analysis

## 1.1 Growth Rate

In algorithm analysis, we only analyse correct algorithms. To measure the algorithm running time, we use machine independent approach by assuming that every basic operation takes constant time:

- Addition
- Subtraction
- Multiplication
- Memory Access

Some non-basic operation examples:

- Sorting
- Searching

The efficiency of an algorithm is the growth rate: how the running time changes as the input size increases.

**Example 1.1.1.** Let there be $n$ inputs. If an algorithm needs $2n$ basic operations and another needs $3n$ basic operations, we consider them to be in the same efficiency category. Suppose that $n$ has increased from 100 to 1000, the growth rate of the $2n$-operation algorithm is:

$$\frac{2 \times 1000 - 2 \times 100}{2 \times 100} = 9.$$

And the growth rate of the $3n$-operation algorithm is:

$$\frac{3 \times 1000 - 3 \times 100}{3 \times 100} = 9.$$

Growth rate analysis is akin to comparison of second-order differentials.

## 1.2 Asymptotic Notation

### 1.2.1 Big-Oh

This notation is the asymptotic upper bound for the growth rate of the running time of an algorithm.

**Definition 1.** Big-Oh notation. Let $f(n)$ be the running time of an algorithm, where $n$ is the input size. Let $g(n)$ be some time complexity function we are trying to relate to our algorithm. Then:

$$f(n) = O(g(n)) \iff \exists c, n_0 \in \mathbb{R}^+ \text{ s.t } \forall n \geqslant n_0,\ f(n) \leqslant cg(n).$$

**Example 1.2.1.** Let $f(n) = 2n^2$. Then:

$$f(n) = O(n^4).$$

Because $f(n) \leqslant n^4$ when $n \geqslant 2$.

$$f(n) = O(n^3).$$

Because $f(n) \leqslant n^3$ when $n \geqslant 2$.

$$f(n) = O(n^2).$$

Because $f(n) \leqslant 2n^2$ when $n \geqslant 1$. This is the best answer.
**Example 1.2.2.** More examples:

$$\begin{aligned} 1 + 4n &= O(n) \\ 7n^2 + 10n + 3 &= O(n^2) \\ \sin n &= O(1) \\ 10 &= O(1) \end{aligned}$$

### 1.2.2 Big-Omega

This notation is the asymptotic lower bound for the growth rate of the running time of an algorithm.

**Definition 2.** Big-Omega notation. Let $f(n)$ be the running time of an algorithm, where $n$ is the input size. Let $g(n)$ be some time complexity function we are trying to relate to our algorithm. Then:

$$f(n) = \Omega(g(n)) \iff \exists c, n_0 \in \mathbb{R}^+ \text{ s.t } \forall n \geqslant n_0,\ f(n) \geqslant cg(n).$$

**Example 1.2.3.** Let $f(n) = 2n^2$ then:

$$\begin{aligned} f(n) &= \Omega(n) \\ f(n) &= \Omega(n^2) \end{aligned}$$

### 1.2.3 Big-Theta

This notation is the asymptotic tight bound for the growth rate of the running time of an algorithm.

**Definition 3.** Big-Theta notation. Let $f(n)$ be the running time of an algorithm, where $n$ is the input size. Let $g(n)$ be some time complexity function we are trying to relate to our algorithm. Then:

$$f(n) = \Theta(g(n)) \iff [f(n) = O(g(n))] \wedge [f(n) = \Omega(g(n))].$$

**Example 1.2.4.** Let $f(n) = n^2$ and $g(n) = 2n^2$, then:

$$\begin{aligned} f(n) &= O(g(n)) \\ f(n) &= \Omega(g(n)) \\ \therefore f(n) &= \Theta(g(n)) \end{aligned}$$

**Note.** If $P(n)$ is a polynomial of degree $k$, then:

$$P(n) = \Theta\left(n^k\right).$$

### 1.2.4 Properties of Big-Oh Notation

Big-Oh is used because it projects the worst-case time complexity. Constant factors may be ignored:

$$\forall k \in \mathbb{R}^+,\ kf(n) = O(f).$$

However the coefficient $a$ in the exponent of $2^{an}$ should not be ignored:

$$2^{3n} = O\left(2^{3n}\right) \qquad 2^{3n} \neq O\left(2^n\right).$$

Higher powers of $n$ grow faster than lower powers:

$$O\left(n^5\right) \text{ grows faster than } O\left(n^4\right).$$

Exponential functions grow faster than powers:

$$n^k = O\left(b^n\right),\ \forall b > 1,\ k \geqslant 0.$$

Logarithms grow slower than powers:

$$\log_b n = O\left(n^k\right),\ \forall b > 1,\ k > 0.$$

Asymptotically, the base of the logarithm does not matter because:

$$\log_a b = \frac{\log_c b}{\log_c a}.$$

**Example 1.2.5.** Base of logarithm does not matter.

$$\log_2 n = \frac{1}{\log_{10} 2} \times \log_{10} n.$$

The first part of which is just a constant:

$$\frac{1}{\log_{10} 2} = 3.322.$$

therefore:

$$\log_2 n = O(\log_{10} n).$$

Asymptotically, any polynomial function of $n$ does not matter after taking its logarithm:

$$\log(n^{475} + n^2 + n + 96) = O(\log n).$$

Because:

$$n^{475} + n^2 + n + n + 96 = O(n^{475}).$$

and:

$$\log(n^{475}) = 475 \log n.$$

The growth rate of a polynomial is given by the growth rate of its leading term. So if $f$ is a polynomial of degree $d$,then $f = O(n^d)$:

$$an^3 + bn^2 = O(n^3).$$

The growth rate of the sum of a polynomial and exponential is the growth rate of the exponential:

$$\begin{aligned} an^3 + b2^n &= O(2^n) \\ an^3 + b2^n + c3^n &= O(3^n) \end{aligned}$$

The growth rate of the sum of a polynomial and logarithm is the growth rate of the polynomial:

$$an + b \log n = O(n).$$

The growth rate of a sum of terms is usually the growth rate of its fastest growing term, however, when we are dealing with the sum of terms where the number of terms grows as $n$ increases, it does not hold true.

The sum of the first $n$ $r$-th powers grows as the $(r+1)$-th power:

$$\begin{aligned} 1 + 2 + 3 + \cdots + n &= \frac{n(n+1)}{2} \\ &= O(n^2) \\ 1 + 2^2 + 3^2 + \cdots + n^2 &= \frac{n(n+1)(2n+1)}{6} \\ &= O\left(n^3\right) \\ 1 + 2^r + 3^r + \cdots + n^r &= O\left(n^{r+1}\right) \end{aligned}$$

If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.

*Proof.* Growth rate is transitive for big-oh. We know that because $f(n) = O(g(n))$:

$$f(n) \leqslant cg(n), \text{ for some } c > 0, n \geqslant m.$$

and because $g(n) = O(h(n))$:

$$g(n) \leqslant dh(n), \text{ for some } d > 0, n \geqslant p.$$

and therefore:

$$f(n) \leqslant (cd)h(n), \text{ for some } cd > 0, n \geqslant \max(p, m).$$

□

If $f(n)$, $g(n)$ are $O(h(n))$, then:

$$f(n) + g(n) = O(h(n)).$$

*Proof.* We know that because $f(n) = O(h(n))$:

$$f(n) \leqslant ch(n), \text{ for some } c > 0, n \geqslant m.$$

and because $g(n) = O(h(n))$:

$$g(n) \leqslant dh(n), \text{ for some } d > 0, n \geqslant p.$$

and therefore:

$$\begin{aligned} f(n) + g(n) &\leqslant ch(n) + dh(n), \ n \geqslant \max(m, p) \\ &\leqslant (c + d)h(n), \ (c + d) > 0, \ n \geqslant \max(m, p) \end{aligned}$$

□

If $P_1(n)$ is $O(f(n))$, $P_2(n)$ is $O(g(n))$, then:

$$P_1(n) + P_2(n) = O(\max(f(n), g(n))).$$

and:

$$P_1(n)P_2(n) = O(f(n)g(n)).$$

If the limit below exists and is finite, then $f(n) = O(g(n))$:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)}.$$

### 1.2.5 Tight Big-Oh Notation

The tight Big-Oh notation gives the most accurate asymptotic bound of the growth rate. We also expect that the tight Big-Oh notation is in the simplest form.

**Example 1.2.6.** Although

$$n = O(n^2).$$

but it is not as tight as:

$$O(n).$$