

# Algorithms and Datastructures

## O-Notation, L'Hopital

Albert-Ludwigs-Universität Freiburg



UNI  
FREIBURG

Prof. Dr. Rolf Backofen

Bioinformatics Group / Department of Computer Science  
Algorithms and Datastructures, November 2017

## $\mathcal{O}$ -Notation

Motivation / Definition

Examples

## $\Omega$ -Notation

## $\Theta$ -Notation

## Runtime

Summary

Limit / Convergence

L'Hôpital / l'Hospital

Practical use

### We are interested in:

- Example: sorting
  - Runtime of Minsort “is growing as”  $n^2$
  - Runtime of HeapSort “is growing as”  $n \log n$
- Growth of a function in runtime  $T(n)$ 
  - The role of constants (e.g.  $1ns$ ) is minor
  - it is enough if relation holds for some  $n \geq \dots$
- Describe the growth of the function **more formally**
  - By the means of Landau-Symbols [Wik]):
    - $\mathcal{O}(n)$  (Big O of  $n$ ),
    - $\Omega(n)$  (Omega of  $n$ ),
    - $\Theta(n)$  (Theta of  $n$ )

### Big $\mathcal{O}$ -Notation:

- Consider the function:  $f: \mathbb{N} \rightarrow \mathbb{R}, n \mapsto f(n)$ 
  - $\mathbb{N}$ : Natural numbers  $\rightarrow$  input size
  - $\mathbb{R}$ : Real numbers  $\rightarrow$  runtime
- **Example:**
  - $f(n) = 3n$
  - $f(n) = 2n \log n$
  - $f(n) = \frac{1}{10}n^2$
  - $f(n) = n^2 + 3n \log n - 4n$

### Big $\mathcal{O}$ -Notation:

- Given two functions  $f$  and  $g$ :

$$f, g: \mathbb{N} \rightarrow \mathbb{R}$$

- **Intuitive:**  $f$  is Big-O of  $g$  ( $f$  is  $\mathcal{O}(g)$ )
  - ... if  $f$  relative to  $g$  does not grow faster than  $g$
  - the growth rate matters, not the absolute values

### Big $\mathcal{O}$ -Notation:

#### ■ Informal: $f = \mathcal{O}(g)$

- “=” corresponds to *is not isequal*
- ... if for some value  $n_0$  for all  $n \geq n_0$
- $f(n) \leq C \cdot g(n)$  for a constant  $C$
- ( $f = \mathcal{O}(g)$ ): From a value  $n_0$  for all  $n \geq n_0 \rightarrow f(n) \leq C \cdot g(n)$ )

#### ■ Formal: $f \in \mathcal{O}(g)$

### Formal: $f \in \mathcal{O}(g)$

$$\mathcal{O}(g) = \{ f : \mathbb{N} \rightarrow \mathbb{R} \mid \exists n_0 \in \mathbb{N}, \exists C > 0, \forall n > n_0 : f(n) \leq C \cdot g(n) \}$$

“set of  
all functions”

“for which”

“it exists”

“for all”

“such that”

### Illustration of the Big O-Notation:



Figure: Runtime of two algorithms  $f_1, f_2$

### Example:

- $f(n) = 5n + 7, g(n) = n$   
 $\Rightarrow 5n + 7 \in \mathcal{O}(g)$   
 $\Rightarrow f \in \mathcal{O}(g)$
- **Intuitive:**  
 $f(n) = 5n + 7 \rightarrow$  linear growth

### Attention

$f(n) \leq g(n)$  is not guaranteed, better is  $f(n) \leq C \cdot g(n) \quad \forall n > n_0$ .



We have to proof:  $\exists n_0, \exists C, \forall n \geq n_0: 5n+7 \leq C \cdot n$ .

$$\begin{aligned} 5n+7 &\leq 5n+n \quad (\text{for } n \geq 7) \\ &= 6n \end{aligned}$$

$$\Rightarrow n_0 = 7, C = 6$$



Alternate proof:

$$\begin{aligned} 5n+7 &\leq 5n+7n \quad (\text{for } n \geq 1) \\ &= 12n \end{aligned}$$

$$\Rightarrow n_0 = 1, C = 12$$



### Big O-Notation:

- We are only interested in the term with the highest-order, the fastest growing summand, the others will be ignored
- $f(n)$  is limited from above by  $C \cdot g(n)$

### Examples:

$$2n^2 + 7n - 20 \in \mathcal{O}(n^2)$$

$$2n^2 + 7n \log n - 20 \in$$

$$7n \log n - 20 \in$$

$$5 \in$$

$$2n^2 + 7n \log n + n^3 \in$$

### Harder Example:

- Polynomes are simple
- More problematic: combination of complex functions

$$2\sqrt{x} + 3\ln x \in \mathcal{O}(??)$$

### Omega-Notation:

#### ■ Intuitive:

- $f \in \Omega(g)$ ,  $f$  is growing at least as fast as  $g$
- So the same as Big-O but with *at-least* and not *at-most*

#### Formal: $f \in \Omega(g)$

$$\Omega(g) = \{f : \mathbb{N} \rightarrow \mathbb{R} \mid \exists n_0 \in \mathbb{N}, \exists C > 0, \forall n > n_0 : f(n) \geq C \cdot g(n)\}$$

“in  $O(n)$   
we had  $\leq$ ”



### Example:

Proof of  $f(n) = 5n + 7 \in \Omega(n)$ :

$$\underbrace{5n+7}_{f(n)} \geq \underbrace{1 \cdot n}_{g(n)} \quad (\text{for } n \geq 1)$$

$$\Rightarrow n_0 = 1, C = 1$$



### Illustration of the Omega-Notation:



Figure: Runtime of two algorithms  $f_1, f_2$

### Big Omega-Notation:

- We are only interested in the term with the highest-order, the fastest growing summand, the others will be ignored
- $f(n)$  is limited **from underneath** by  $c \cdot g(n)$

### Examples:

$$2n^2 + 7n - 20 \in \Omega(n^2)$$

$$2n^2 + 7n \log n - 20 \in$$

$$7n \log n - 20 \in$$

$$5 \in$$

$$2n^2 + 7n \log n + n^3 \in$$



### Theta-Notation:

- **Intuitive:**  $f$  is Theta of  $g$  ...
  - ... if  $f$  is growing as much as  $g$
  - $f \in \Theta(g)$ ,  $f$  is growing at the same speed as  $g$

### Formal: $f \in \Theta(g)$

$$\Theta(g) = \underbrace{\mathcal{O}(g) \cap \Omega(g)}_{\text{Intersection}}$$

### Example:

$$\begin{aligned} f(n) &= 5n + 7, f(n) \in \mathcal{O}(n), f(n) \in \Omega(n) \\ \Rightarrow f(n) &\in \Theta(n) \end{aligned}$$

*Proof for  $\mathcal{O}(g)$  and  $\Omega(g)$  look at slides 11 and 17*



- $f$  and  $g$  have the same “growth”

### Big O-Notation $\mathcal{O}(n)$ :

- $f$  is growing **at most** as fast as  $g$
- $C \cdot g(n)$  is the upper bound

### Big Omega-Notation $\Omega(n)$ :

- $f$  is growing **at least** as fast as  $g$
- $C \cdot g(n)$  is the lower bound

### Big Theta-Notation $\Theta(n)$ :

- $f$  is growing at **the same** speed as  $g$ 
  - $C_1 \cdot g(n)$  is the lower bound
  - $C_2 \cdot g(n)$  is the upper bound

Table: Common runtime types

Runtime	Growth
$f \in \Theta(1)$	constant time
$f \in \Theta(\log n) = \Theta(\log_k n)$	logarithmic time
$f \in \Theta(n)$	linear time
$f \in \Theta(n \log n)$	n-log-n time (nearly linear)
$f \in \Theta(n^2)$	squared time
$f \in \Theta(n^3)$	cubic time
$f \in \Theta(n^k)$	polynomial time
$f \in \Theta(k^n), f \in \Theta(2^n)$	exponential time

- So far discussed:
  - Membership in  $O(\dots)$  proofed by hand:  
Explicit calculation of  $n_0$  and  $C$
  - **However:** Both hint at **limits** in calculus

### Definition of “Limit”

- The **limit**  $L$  exists for an infinite sequence  $f_1, f_2, f_3, \dots$  if for all  $\varepsilon > 0$  one  $n_0 \in \mathbb{N}$  exists, such that for all  $n \geq n_0$  the following holds true:  $|f_n - L| \leq \varepsilon$
- A function  $f: \mathbb{N} \rightarrow \mathbb{R}$  can be written as a sequence  
 $\Rightarrow \lim_{n \rightarrow \infty} f_n = L$

The limit is converging:

$$\forall \varepsilon > 0 \exists n_0 \in \mathbb{N} \forall n \geq n_0: |f_n - L| \leq \varepsilon$$

- Example for the proof of a limit
- Function  $f(n) = 2 + \frac{1}{n}$  with limit  $\lim_{n \rightarrow \infty} f(n) = 2$
- “Engineering” solution: use  $n = \infty$

$$\frac{1}{\infty} = 0 \Rightarrow \lim_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} 2 + \frac{1}{n} = 2$$

- Now a more formal proof for  $\lim_{n \rightarrow \infty} 2 + \frac{1}{n} = 2$
- We need to show: for all given  $\varepsilon$  there is an  $n_0$  such that for all  $n \geq n_0$

$$\left| 2 + \frac{1}{n} - 2 \right| = \left| \frac{1}{n} \right| \leq \varepsilon$$

- E.g.: for  $\varepsilon = 0.01$  we get  $\frac{1}{n} \leq \varepsilon$  for  $n \geq 100$
- In general

$$n_0 = \left\lceil \frac{1}{\varepsilon} \right\rceil$$

- Then we get:

$$\left| \frac{1}{n} \right| = \frac{1}{n} \leq \frac{1}{n_0} = \frac{1}{\left\lceil \frac{1}{\varepsilon} \right\rceil} \leq \frac{1}{\frac{1}{\varepsilon}} = \varepsilon \quad \square$$



Let  $f, g: \mathbb{N} \rightarrow \mathbb{R}$  with an existing limit

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = L$$

Hence the following holds:

$$f \in \mathcal{O}(g) \quad \Leftrightarrow \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty \quad (1)$$

$$f \in \Omega(g) \quad \Leftrightarrow \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0 \quad (2)$$

$$f \in \Theta(g) \quad \Leftrightarrow \quad 0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty \quad (3)$$

$$f \in \mathcal{O}(g) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

Forward proof ( $\Rightarrow$ ):

$$f \in \mathcal{O}(g) \stackrel{\text{def. of } \mathcal{O}(n)}{\Rightarrow} \exists n_0, C \forall n \geq n_0 : f(n) \leq C \cdot g(n)$$

$$\Rightarrow \exists n_0, C \forall n \geq n_0 : \frac{f(n)}{g(n)} \leq C$$

$$\Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \leq C \quad \square$$

### Backward proof ( $\Leftarrow$ ):

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

$$\Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = C \quad \text{For some } C \in \mathbb{R} \text{ (Limit)}$$

$$\text{def. limes} \Rightarrow \exists n_0, \forall n \geq n_0 : \frac{f(n)}{g(n)} \leq C + \varepsilon \quad (\text{e.g. } \varepsilon = 1)$$

$$\Rightarrow \exists n_0, \forall n \geq n_0 : f(n) \leq \underbrace{(C+1)}_{O\text{-notation constant}} \cdot g(n)$$

$$\Rightarrow f \in \mathcal{O}(g) \quad \square$$

## ■ Intuitive:

$$\lim_{n \rightarrow \infty} 2 + \frac{1}{n} = 2 + \frac{1}{\infty} = 2$$

## ■ With L'Hôpital:

- Let  $f, g : \mathbb{N} \rightarrow \mathbb{R}$
- If  $\lim_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} g(n) = \infty/0$

$$\Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$$

## ■ Holy inspiration

you need a doctoral degree for that

**The limit can not be determined in the way of an Engineer:**

$$\lim_{n \rightarrow \infty} \frac{\ln(n)}{n} = \frac{\lim_{n \rightarrow \infty} \ln(n)}{\lim_{n \rightarrow \infty} n} \quad \xrightarrow{\text{plugging in}} \quad \frac{\infty}{\infty}$$

**Determine the limit with using L'Hôpital:**

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$$

## Using L'Hôpital:

Numerator: **f(n)**:  $n \mapsto \ln(n)$

Denominator: **g(n)**:  $n \mapsto n$

$$\Rightarrow f'(n) = \frac{1}{n} \quad (\text{derivation from Numerator})$$

$$\Rightarrow g'(n) = 1 \quad (\text{derivation from Denominator})$$

$$\lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)} = \lim_{n \rightarrow \infty} \frac{1}{n} = 0 \quad \Rightarrow \quad \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{\ln(n)}{n} = 0$$

## What can we take for granted without proofing?

- Only things that are trivial
- It is always better to proof it

## Examples:

$$\lim_{n \rightarrow \infty} \frac{1}{n} = 0 \quad \text{is trivial}$$

$$\lim_{n \rightarrow \infty} \frac{1}{n^2} = 0 \quad \text{is trivial}$$

$$\lim_{n \rightarrow \infty} \frac{\log(n)}{n} = 0 \quad \text{use L'Hopital}$$

### Practical use:

- It is much easier to determine the runtime of an algorithm by using the  $\mathcal{O}$ -Notation
  - 1 Computing rules
  - 2 Practical use



### ■ Transitivity:

$$f \in \Theta(g) \wedge g \in \Theta(h) \rightarrow f \in \Theta(h)$$

$$f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h) \rightarrow f \in \mathcal{O}(h)$$

$$f \in \Omega(g) \wedge g \in \Omega(h) \rightarrow f \in \Omega(h)$$

### ■ Symmetry:

$$f \in \Theta(g) \leftrightarrow g \in \Theta(f)$$

$$f \in \mathcal{O}(g) \leftrightarrow g \in \Omega(f)$$

### ■ Reflexivity:

$$f \in \Theta(f) \quad f \in \Omega(f) \quad f \in \mathcal{O}(f)$$

### ■ Trivial:

$$\begin{aligned}f &\in \mathcal{O}(f) \\k \cdot \mathcal{O}(f) &= \mathcal{O}(f) \\ \mathcal{O}(f+k) &= \mathcal{O}(f)\end{aligned}$$

### ■ Addition:

$$\mathcal{O}(f) + \mathcal{O}(g) = \mathcal{O}(\max\{f, g\})$$

### ■ Multiplication:

$$\mathcal{O}(f) \cdot \mathcal{O}(g) = \mathcal{O}(f \cdot g)$$

- The input size for all examples is  $n$
- Basic operations

$i1 = 0$	$\mathcal{O}(1)$
----------	------------------

- Sequences of basic operations

$i1 = 0$	$\mathcal{O}(1)$	}	$327 \cdot \mathcal{O}(1) = \mathcal{O}(1)$
$i2 = 0$	$\mathcal{O}(1)$		
...	...		
$i327 = 0$	$\mathcal{O}(1)$		

### ■ Loops

```
for i in range(0, n):  
    a[i] = 0
```

$$\left. \frac{\mathcal{O}(n)}{\mathcal{O}(1)} \right\} \mathcal{O}(1) \cdot \mathcal{O}(n) = \mathcal{O}(n)$$

```
for i in range(0, n):  
    a1[i] = 0  
    ...  
    a137[i] = 0
```

$$\left. \begin{array}{c} \frac{\mathcal{O}(n)}{\mathcal{O}(1)} \\ \dots \\ \mathcal{O}(1) \end{array} \right\} \left. \begin{array}{c} 137 \cdot \mathcal{O}(1) \\ = \mathcal{O}(1) \end{array} \right\} \mathcal{O}(1) \cdot \mathcal{O}(n) = \mathcal{O}(n)$$

### ■ Loops

```
for i in range(0, n):  
    for j in range(0, n-1):  
        a1[i][j] = 0  
        ...  
        a137[i][j] = 0
```

$$\left. \begin{array}{l} \frac{\mathcal{O}(n)}{\mathcal{O}(n-1)} \\ \frac{\mathcal{O}(1)}{\mathcal{O}(1)} \\ \dots \\ \frac{\mathcal{O}(1)}{\mathcal{O}(1)} \end{array} \right\} \left. \begin{array}{l} \mathcal{O}(n) \cdot \mathcal{O}(n) \\ = \mathcal{O}(n^2) \\ 137 \cdot \mathcal{O}(1) \\ = \mathcal{O}(1) \end{array} \right\} \left. \begin{array}{l} \mathcal{O}(1) \cdot \mathcal{O}(n^2) \\ = \mathcal{O}(n^2) \end{array} \right\}$$

### ■ Conditions

if  $x < 100$ :

$y = x$

else:

for  $i$  in range(0,  $n$ ):

if  $a[i] > y$ :

$y = a[i]$

$$\left. \begin{array}{l} \frac{\mathcal{O}(1)}{\mathcal{O}(1)} \\ \frac{\mathcal{O}(n)}{\mathcal{O}(1)} \end{array} \right\} \left. \begin{array}{l} \mathcal{O}(1) \\ \mathcal{O}(n) \cdot \mathcal{O}(1) \\ = \mathcal{O}(n) \end{array} \right\} \begin{array}{l} \mathcal{O}(\max\{1, n\}) \\ = \mathcal{O}(n) \end{array}$$

- **Input:** List  $x$  with  $n$  numbers
- **Output:**  $a[i]$  is the arithmetic mean of  $x[0]$  to  $x[i]$

```
def arithMean(x):  
    a = [0] * len(x)  
    for i in range(0, len(x)):  
        s = 0  
        for j in range(0, i+1):  
            s = s + x[j]  
  
        a[i] = s / (i+1)  
  
    return a
```

# $\mathcal{O}$ -Notation Runtime complexity

for i in range(0, len(x)):	$\frac{\mathcal{O}(n)}{\mathcal{O}(1)}$	}	$\mathcal{O}(n)$	}	$\mathcal{O}(n) \cdot \mathcal{O}(i)$ $= \mathcal{O}(n^2)$
s = 0	$\mathcal{O}(1)$				
for j in range(0, i+1):	$\frac{\mathcal{O}(i+1)}{\mathcal{O}(1)}$	}	$\mathcal{O}(i)$		
s = s + x[j]	$\mathcal{O}(1)$				
a[i] = s / (i+1)	$\mathcal{O}(1)$				

- How often will the instructions in the loop be executed, when the problem has size  $n$ ?

$$1 + 2 + \dots + n = \frac{n \cdot (n+1)}{2} \in \mathcal{O}(n^2)$$



### Way of speaking:

- With the  $\mathcal{O}$ -Notation we look at the behavior of a function when  $n \rightarrow \infty$
- We only analyze the runtime when  $n \geq n_0$
- We talk about **asymptotic analysis**, when we discuss cost, runtime, etc. as  $\mathcal{O}(\dots)$ ,  $\Omega(\dots)$  or  $\Theta(\dots)$

### Attention:

- If you are using **asymptotic analysis**, you can not make any predictions about the runtime of smaller input sizes ( $n < n_0$ )
- For small input sizes (mostly  $n < 10$ ), the runtime is predictably small
- $n_0$  does not necessarily have to be small

### Examples:

- Let A and B be algorithms
  - A has the runtime  $f(n) = 80n$
  - B has the runtime  $f(n) = 2n \log_2 n$
- So  $f = \mathcal{O}(g)$  but **not**  $\Theta(g)$ 
  - $\Rightarrow$  A is asymptotic faster than B
  - $\Rightarrow$  There is a  $n_0$  for that  $n \geq n_0: f(n) \leq g(n)$

### When is A faster than B?

We search the minimal  $n_0$ :

$$f(n_0) = g(n_0)$$

$$80 n_0 = 2 n_0 \log_2 n_0$$

$$40 = \log_2 n_0$$

$$n_0 = 2^{40}$$

$$= (2^{10})^4 = (1024)^4$$

$$\approx (10^3)^4 = 10^{12}$$

$$\approx 1 \text{ trillion}$$

A is faster than B if  $n_0$  has more than 1 trillion elements

- Logarithm of different bases differ only by a constant

$$\log_a n = \frac{\log_b n}{\log_b a} = \frac{1}{\log_b a} \cdot \log_b n$$

- Hence:  $\log_a n \in \Theta(\log_b n)$
- For exponent this does not hold

$$3^n \notin \Theta(2^n)$$

- Proof: Use equation (1) from Slide 31

$$3^n \in \mathcal{O}(2^n) \Leftrightarrow \lim_{n \rightarrow \infty} \frac{3^n}{2^n} < \infty$$

- However:

$$\lim_{n \rightarrow \infty} \frac{3^n}{2^n} = \lim_{n \rightarrow \infty} \left(\frac{3}{2}\right)^n = \infty$$

■ Figure for slide 28



## ■ General

- [MS08] [Kurt Mehlhorn and Peter Sanders.](#)  
Algorithms and data structures, 2008.  
[https://people.mpi-inf.mpg.de/~mehlhorn/  
ftp/Mehlhorn-Sanders-Toolbox.pdf](https://people.mpi-inf.mpg.de/~mehlhorn/ftp/Mehlhorn-Sanders-Toolbox.pdf).

## ■ Big O notation

[Wik] [Big O notation](https://en.wikipedia.org/wiki/Big_O_notation)

[https://en.wikipedia.org/wiki/Big\\_O\\_notation](https://en.wikipedia.org/wiki/Big_O_notation)