Algorithmns and Datastructures

O-Notation, L'Hopital

Prof. Dr. Rolf Backofen

Bioinformatics Group / Department of Computer Science

Algorithmns and Datastructures, November 2017

Structure

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

Big $\mathcal{O}$-Notation:

- Given two functions f and g:
  $f, g \colon \mathbb{N} \to \mathbb{R}$
- **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

$\mathcal{O}$-Notation

Definition

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

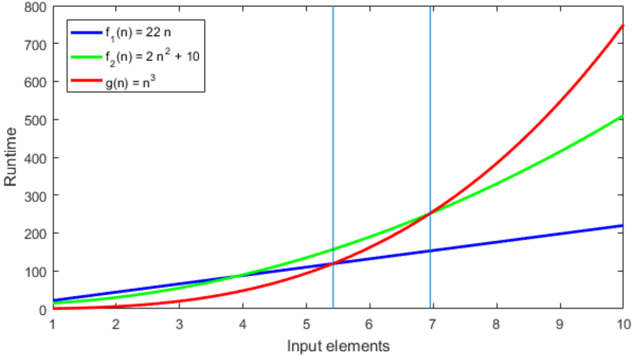
$\mathcal{O}(g) = \{\ \mathrm{f}\ : \mathbb{N} \rightarrow \mathbb{R}\ \text{—}\ \exists n_0 \in \mathbb{N},\ \exists C > 0,\ \forall n > n_0\colon f(n) \leq C \cdot g(n)\}$

"set of all functions" "for which" "it exists" "for all" "such that"

$\mathcal{O}$-Notation

Examples

Illustration of the Big O-Notation:



$g(n)$

$f_2 \in \mathcal{O}(g)$

$f_1 \in \mathcal{O}(g)$

Figure: Runtime of two algorithms f_1, f_2

$\mathcal{O}$-Notation

Examples

Example:

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

Attention

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

$\mathcal{O}$-Notation
Proof

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

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

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

$\mathcal{O}$-Notation

Proof

Alternate proof:

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

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

$\mathcal{O}$-Notation

Examples

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:

$$\begin{aligned} 2\,n^2 + 7\,n - 20 &\in \mathcal{O}(n^2) \\ 2\,n^2 + 7\,n\log n - 20 &\in \\ 7\,n\log n - 20 &\in \\ 5 &\in \\ 2\,n^2 + 7\,n\log n + n^3 &\in \end{aligned}$$

Harder Example:

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

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

Ω-Notation

Definition

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: $f \in \Omega(g)$

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

"in $O(n)$
we had $\leq$"

Ω-Notation

Proof

Example:

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

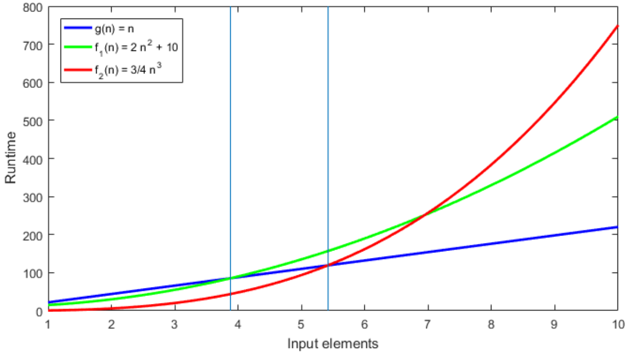
$$\underbrace{5\,n + 7}_{f(n)} \;\; \geq \;\; \underbrace{1 \cdot n}_{g(n)} \quad (\text{for } n \geq 1)$$

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

Ω-Notation

Examples

Illustration of the Omega-Notation:



$f_2 \in \Omega(g)$

$f_1 \in \Omega(g)$

$g(n)$

Figure: Runtime of two algorithms f_1, f_2

Ω-Notation

Examples

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:

$$
\begin{aligned}
2\,n^2 + 7\,n - 20 &\in \Omega(n^2) \\
2\,n^2 + 7\,n\log n - 20 &\in \\
7\,n\log n - 20 &\in \\
5 &\in \\
2\,n^2 + 7\,n\log n + n^3 &\in
\end{aligned}
$$

Θ-Notation
Definition

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: $f \in \Theta(g)$

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

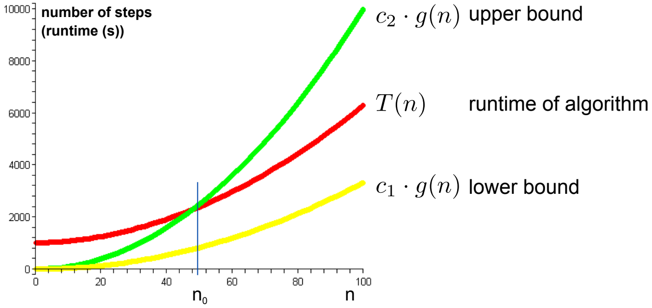
Example:

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

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

Θ-Notation

Graphs



- 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

Runtime

Common Runtimes

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 $O(\ldots)$ 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 $f_1, f_2, f_3, \ldots$ if for all $\epsilon > 0$ one $n_0 \in \mathbb{N}$ exists, such that for all $n \geq n_0$ the following holds true: $|f_n - L| \leq \epsilon$
- A function $f : \mathbb{N} \to \mathbb{R}$ can be written as a sequence $\Rightarrow \lim\limits_{n \to \infty} f_n = L$

The limit is converging:

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

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

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

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

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

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

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

- Then we get:

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

$\mathcal{O}$-Notation

Limit / Convergence

Let $f, g \colon \mathbb{N} \to \mathbb{R}$ with an existing limit

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

Hence the following holds:

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

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

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

$\mathcal{O}$-Notation

Limit / Convergence

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

Forward proof ($\Rightarrow$):

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

$$\begin{aligned} &\Rightarrow \exists n_0,\, C\; \forall n \geq n_0 : \frac{f(n)}{g(n)} \leq\; C \\ &\Rightarrow \qquad\qquad\qquad \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq\; C \quad \square \end{aligned}$$

Backward proof ($\Leftarrow$):

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

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

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

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

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

Limits with L'Hôpital

- **Intuitive:**

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

- **With L'Hôpital:**
  - Let $f, g : \mathbb{N} \to \mathbb{R}$
  - If $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty/0$

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

- **Holy inspiration**

you need a doctoral degree for that

Limits with L'Hôpital

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

$$\lim_{n\to\infty} \frac{\ln(n)}{n} = \frac{\lim_{n\to\infty} \ln(n)}{\lim\limits_{n\to\infty} n} \xrightarrow{\text{plugging in}} \frac{\infty}{\infty}$$

Determine the limit with using L'Hôpital:

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

Limits with L'Hôpital

Using L'Hôpital:

Numerator: $\mathbf{f(n)}: n \mapsto \ln(n)$
Denominator: $\mathbf{g(n)}: n \mapsto n$

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

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

Limits with L'Hôpital

What can we take for granted without proofing?

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

Examples:

$$\lim_{n\to\infty} \frac{1}{n} = 0 \qquad \text{is trivial}$$

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

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

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

$\mathcal{O}$-Notation

Characteristics

- Transitivity:

$$\begin{aligned}
f \in \Theta(g) \ \wedge \ g \in \Theta(h) \quad &\rightarrow \quad f \in \Theta(h) \\
f \in \mathcal{O}(g) \ \wedge \ g \in \mathcal{O}(h) \quad &\rightarrow \quad f \in \mathcal{O}(h) \\
f \in \Omega(g) \ \wedge \ g \in \Omega(h) \quad &\rightarrow \quad f \in \Omega(h)
\end{aligned}$$

- Symmetry:

$$\begin{aligned}
f \in \Theta(g) \quad &\leftrightarrow \quad g \in \Theta(f) \\
f \in \mathcal{O}(g) \quad &\leftrightarrow \quad g \in \Omega(f)
\end{aligned}$$

- Reflexivity:

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

$\mathcal{O}$-Notation

Calculation Rules

- 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:

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

$\mathcal{O}$-Notation

Runtime Complexity

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

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

- Sequences of basic operations

$$\begin{array}{ll} \texttt{i1 = 0} & \mathcal{O}(1) \\ \texttt{i2 = 0} & \mathcal{O}(1) \\ \dots & \dots \\ \texttt{i327 = 0} & \mathcal{O}(1) \end{array} \left.\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right\} \quad 327 \cdot \mathcal{O}(1) = \mathcal{O}(1)$$

- Loops

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

$$\left.\begin{array}{c}\mathcal{O}(n)\\ \hline \mathcal{O}(1)\end{array}\right\} \quad \mathcal{O}(1)\cdot\mathcal{O}(n) = \mathcal{O}(n)$$

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

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

$\mathcal{O}$-Notation

Runtime Complexity

- Loops

`for i in range(0, n):`	$\mathcal{O}(n)$	$\mathcal{O}(n) \cdot \mathcal{O}(n) = \mathcal{O}(n^2$	$\mathcal{O}(1) \cdot \mathcal{O}(n^2) = \mathcal{O}(n^2)$
`    for j in range(0, n-1):`	$\mathcal{O}(n-1)$		
`        a1[i][j] = 0`	$\mathcal{O}(1)$	$137 \cdot \mathcal{O}(1) = \mathcal{O}(1)$	
`        ...`	$\dots$		
`        a137[i][j] = 0`	$\mathcal{O}(1)$		

$\mathcal{O}$-Notation

Runtime Complexity

- Conditions

`if x < 100:`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\max\{1, n\}) = \mathcal{O}(n)$
`    y = x`	$\mathcal{O}(1)$		
`else :`			
`    for i in range(0, n):`	$\mathcal{O}(n)$	$\mathcal{O}(n) \cdot \mathcal{O}(1) = \mathcal{O}(n)$	
`        if a[i] > y:`	$\mathcal{O}(1)$		
`            y = a[i]`	$\mathcal{O}(1)$		

$\mathcal{O}$-Notation

Arithmetic mean

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

```python
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)):`	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n) \cdot \mathcal{O}(i)$ $= \mathcal{O}(n^2)$
`    s = 0`	$\mathcal{O}(1)$		
`    for j in range(0, i+1):`	$\mathcal{O}(i+1)$	$\mathcal{O}(i)$	
`        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?

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

Way of speaking:

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

$\mathcal{O}$-Notation

Discussion

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

$\mathcal{O}$-Notation

Discussion

Examples:

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

$\mathcal{O}$-Notation

Discussion

When is A faster then B?

We search the minimal n_0:

$$\begin{aligned}
f(n_0) &= g(n_0) \\
80\, n_0 &= 2\, n_0 \log_2 n_0 \\
40 &= \log_2 n_0 \\
n_0 &= 2^{40} \\
&= (2^{10})^4 = (1024)^4 \\
&\approx (10^3)^4 = 10^{12} \\
&\approx 1\,\text{trillion}
\end{aligned}$$

A ist faster than B if n_0 has more than 1 trillion elements

Runtime Examples

Continued

- 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

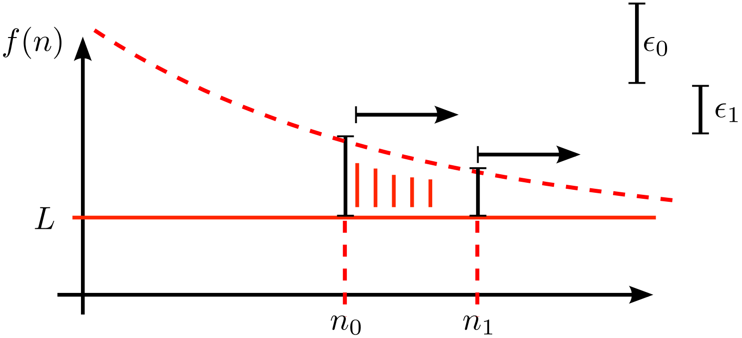
$$3^n \in \mathcal{O}(2^n) \Leftrightarrow \lim_{n\to\infty} \frac{3^n}{2^n} < \infty$$

- However:

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

Additional Figure

- Figure for slide 28



Further Literature

- **General**

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

Further Literature

- **Big O notation**

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