Algorithmns and Datastructures

O-Notation, L'Hopital

Albert-Ludwigs-Universität Freiburg

Prof. Dr. Rolf Backofen

Bioinformatics Group / Department of Computer Science

Algorithmns and Datastructures, March 2016

Structure

Feedback from the exercises

1. Mathematical exercises mostly easy
2. Programming exercise took more time, but still mostly manageable
3. Python programming gradually gets more comfortable
4. The first two problems were quite easy. I just wondered why it was stated to prove them with strong induction.

Feedback from the exercises

Tipps:

1. Try and test small pieces, do not postpone the testing
2. Try to first write (some) tests and then the code (Test Driven Development)
   $\rightarrow$ Proven to be more straight, faster, less time for debugging
3. Python programming gradually gets more comfortable
   $\rightarrow$ Please use the forum such that everyone can participate and learn from each other

Feedback from the lecture

No feedback yet, please let us know what we can improve.

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)

UNI FREIBURG

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:**
  - $f(n) = 3n$
  - $f(n) = 2n \log n$
  - $f(n) = \frac{1}{10}n^2$
  - $f(n) = n^2 + 3n \log n - 4n$

Big $\mathcal{O}$-Notation:

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

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

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

"set of all functions"

"for which"

"it exists"

"for all"

"such that"

Illustration of the Big O-Notation:

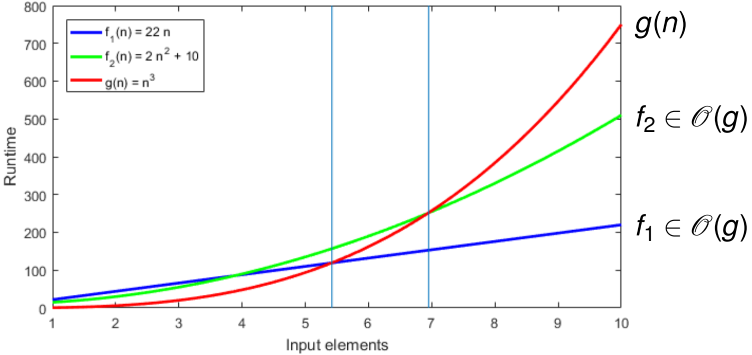


Figure: Runtime of two algorithms f_1, f_2

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 $f(n) \leq C \cdot g(n) \ \forall n > n_0$.

$\mathcal{O}$-Notation
Proof

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

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

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

UNI FREIBURG

Alternate proof:

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

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

Big O-Notation:

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

Examples:

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

Harder Example:

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

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

UNI FREIBURG

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 $\leq$"

Example:

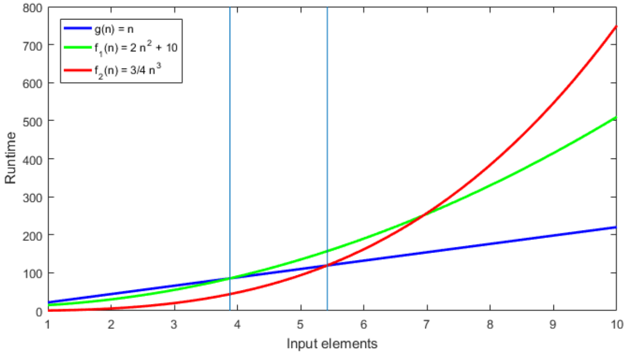
Proof of $f(n) = 5n + 7 \in \Omega(n)$:

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

$\Rightarrow n_0 = 1, C = 1$ □

UNI FREIBURG

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

UNI FREIBURG

Big Omega-Notation:

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

Examples:

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

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

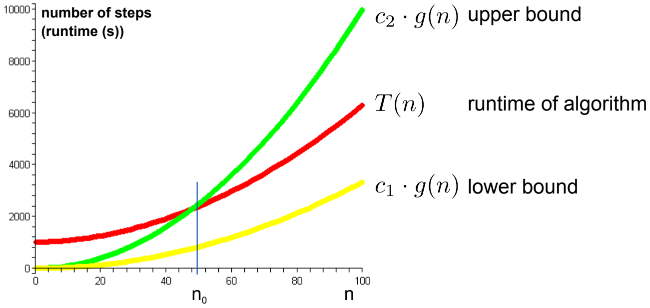
Formal: $f \in \Theta(g)$

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

Example:

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

Proof for $\mathcal{O}(g)$ and $\Omega(g)$ look at slides 15 and 21



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

UNI
FREIBURG

Definition of "Limit"

- The limit L exists for an infinite sequence $f_1, f_2, f_3, \ldots$ 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 $f : \mathbb{N} \to \mathbb{R}$ can be written as a sequence $\Rightarrow \lim\limits_{n \to \infty} f_n = L$

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 limes $\lim_{n\to\infty} f(n) = 2$
- "Engineering" solution: use $n = \infty$

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

$\mathcal{O}$-Notation

Limit / Convergence

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

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

- 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:

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

UNI FREIBURG

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:

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

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

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

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

Forward proof ($\Rightarrow$):

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

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

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

UNI FREIBURG

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 $\lim\limits_{n\to\infty} f(n) = \lim\limits_{n\to\infty} g(n) = \infty/0$

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

- **Holy inspiration**

you need a doctoral degree for that

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

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

Determine the limit with using L'Hôpital:

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

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)

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

What can we take for granted without proofing?

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

Examples:

$$\begin{aligned}
\lim_{n\to\infty} \frac{1}{n} &= 0 && \text{is trivial} \\
\lim_{n\to\infty} \frac{1}{n^2} &= 0 && \text{is trivial} \\
\lim_{n\to\infty} \frac{\log(n)}{n} &= 0 && \text{use L'Hopital}
\end{aligned}$$

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:

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

- Symmetry:

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

- Reflexivity:

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

UNI FREIBURG

- 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

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

- 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
```

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

- Loops

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

- 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]`	$\mathcal{O}(1)$		

- 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)):`	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n) \cdot \mathcal{O}(i)$ $= \mathcal{O}(n^2)$
`    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 $\Theta(\ldots)$

UNI FREIBURG

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 $f(n) = 80\,n$
  - 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 $n \geq n_0:\ f(n) \leq g(n)$

UNI FREIBURG

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 35

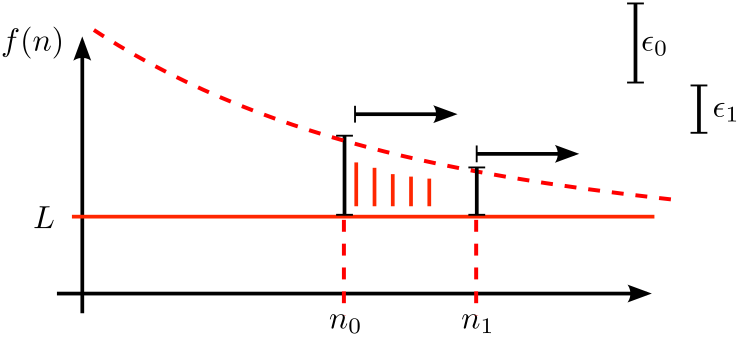
$$3^n \in \mathcal{O}(2^n) \Leftrightarrow \lim_{n\to\infty} \frac{3^n}{2^n} < \infty$$

- 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 32



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