Algorithmen und Datenstrukturen (ESE)
Entwurf, Analyse und Umsetzung von Algorithmen (IEMS)
WS 2014 / 2015
Vorlesung 3, Donnerstag 6. November 2014
(O-Notation, Theta, Omega)

Junior-Prof. Dr. Olaf Ronneberger
Image Analysis Lab
Institut für Informatik
Universität Freiburg

[basiert auf der Veranstaltung von Prof. Dr. Hannah Bast, Lehrstuhl für Algorithmen und Datenstrukturen, mit Folien aus „Informatik 2“ von Prof. Dr. Matthias Teschner]

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Klausurtermin steht fest: Freitag, 27.2.2015 16:00-18:00
  - Ihre Erfahrungen mit dem Ü2 (Induktion + Implementierung HeapSort)
- O-Notation
  - Motivation
  - Klassische Definition mit C und n_0
  - Einfachere Bestimmung über den $\lim_{n\to\infty}$
  - Praktischer Nutzen
  - f = O(g) heißt nicht immer f ist besser
  - Übungsblatt 3: ein paar Beweise / Rechenaufgaben dazu
    So was in der Art kommt meistens auch in der Klausur !

Erfahrungen mit dem Ü2

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 6. November 09:00
  - Aufgabe 1 und 2 (Beweise) fanden fast alle gut machbar
  - Aufgabe 3 (HeapSort) hat vielen Probleme bereitet und deutlich mehr als die vorgesehen 4 Std. gedauert
  - Unklar, was „heapify" und „repairHeap" machen sollen → bitte gleich im Forum nachfragen
  - Tips zum Debuggen vom Code?
    - Generell: Code in **kleine, einfach zu testende Module** strukturieren → Unit tests
    - Tests zuerst schreiben, **stückweise Funktionalität hinzufügen**, und den Code immer **compilierbar halten**
    - **Aber:** Unit tests sind nicht dazu da, die Interna der Implementation zu testen
    - **Zwischenergebnisse visualisieren** (z.B. heap in jedem Schritt als lineares Array ausgeben, als ASCII-art, heap-Verletzungen anzeigen, etc.)
    - im **Debugger** „break points" setzen und Variablen ansehen
    - für C++ Programmierer: **„valgrind"** um Speicherzugriffsfehler zu finden
    - Programmieren ist ein Handwerk, das man nur durch **viel Übung und Erfahrung** lernt.

O-Notation — Motivation

UNI FREIBURG

- Primär interessiert uns oft
  - das "Wachstum" einer Funktion, z.B. einer Laufzeit $T(n)$
  - Die Werte der Konstanten (z.B. 1ns) sind dabei oft sekundär
  - Und auch, wenn die Schranken erst ab $n \geq \ldots$ gelten
  - Zum Beispiel war beim Sortieren interessant, dass
    - die Laufzeit von MinSort "wächst wie" n^2
    - aber die Laufzeit von HeapSort "wächst wie" $n \cdot \log n$
  - Das wollen wir jetzt formaler machen, damit wir in Zukunft etwas schreiben bzw. sagen können wie:
    - Die Laufzeit des Algorithmus ist $O(n)$ "O von n"
    - Die Laufzeit des Algorithmus ist $\Omega(n)$ "Omega von n"
    - Die Laufzeit des Algorithmus ist $\Theta(n)$ "Theta von n"
  - O, Ω, Θ sind die **Landau-Symbole**

O-Notation — Definition

UNI FREIBURG

- Vorweg
  - Wir betrachten Funktionen $f : \mathbb{N} \to \mathbb{R} : n \mapsto f(n)$
    - $\mathbb{N}$: die natürlichen Zahlen ... typisch: Eingabegröße
    - $\mathbb{R}$: die reellen Zahlen ... typisch: Laufzeit
  - Zum Beispiel
    - f(n) = 3 · n
    - f(n) = 2 · n · log n
    - f(n) = n^2 / 10
    - f(n) = n^2 + 3 · n · log n - 4 · n

Definition Groß-O

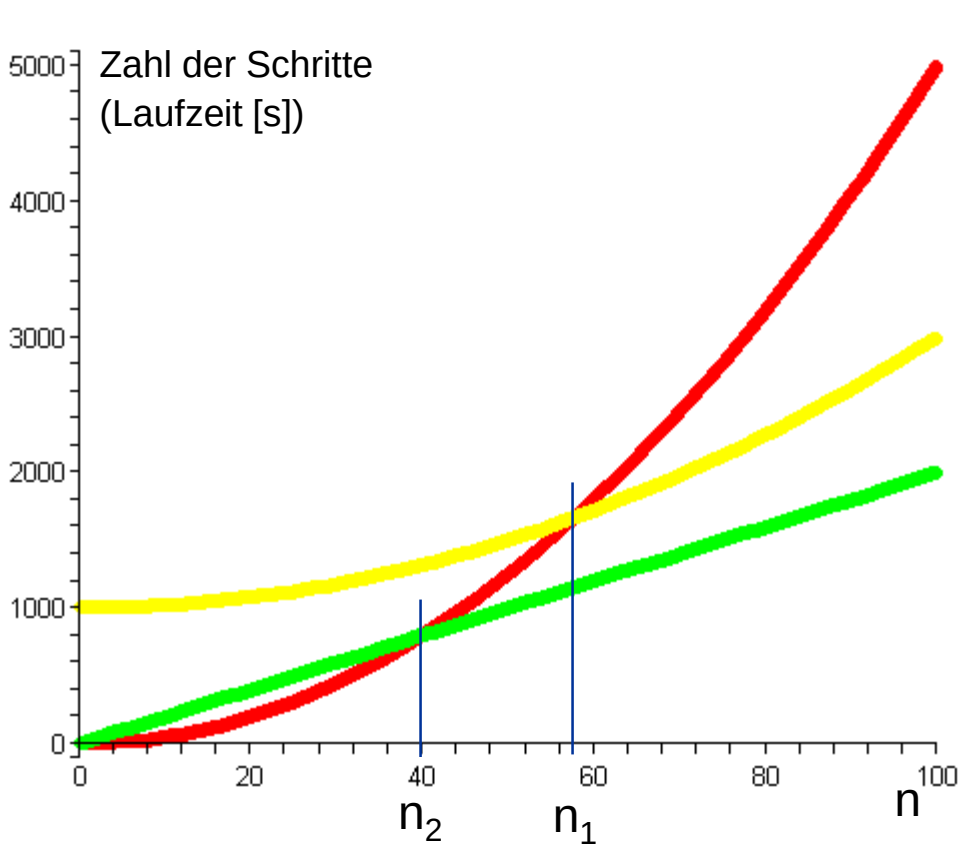
- Groß-O, Definition
  - Seien f und g zwei Funktionen $\mathbb{N} \to \mathbb{R}$
  - **Intuitiv:** Man sagt f ist Groß-O von g ...
    - ... wenn f "höchstens so stark wächst wie" g
    - es zählt die "Wachstumsrate", nicht die absoluten Werte!
  - **Informal:** Man schreibt $f = \mathcal{O}(g)$ ...
    - ... wenn ab irgendeinem Wert n_0 für alle $n \geq n_0$
    - $f(n) \leq C \cdot g(n)$ für irgendeine Konstante C
  - **Formal:** Korrekterweise müsste man schreiben: $f \in \mathcal{O}(g)$
    da O(g) definiert ist als eine Menge von Funktionen:

Steht für „ist", nicht „ist gleich"

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

„Die Menge aller Funktionen" „für die gilt:" „es existiert" „für alle" „so, dass"

Illustration Groß-O



$c \cdot g(n)$

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

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

f ist höchstens von der Größenordnung von g.

Beispiele Groß-O

- Groß-O, Beispiel
  - Sei $f(n) = 5 \cdot n + 7$ und $g(n) = n$
  - Dann gilt $f = O(g)$ bzw. man (als Informatiker) schreibt $5 \cdot n + 7 = O(n)$
  - **Achtung:** Das heißt nicht, dass $f(n) \leq g(n)$, sondern dass $f(n) \leq C \cdot g(n)$ ab einem bestimmten $n > n_0$
  - **Intuitiv:** $5 \cdot n + 7$ wächst höchstens "linear"
  - Beweis unter Verwendung der Definition von O:
  - Es muss gezeigt werden: $\exists n_0, C: \quad 5 \cdot n + 7 \leq C \cdot n \quad \forall n \geq n_0$
  - Beweis: $5 \cdot n + 7 \leq 5 \cdot n + n \qquad (\text{für } n \geq 7)$

    $\leq 6 \cdot n \qquad \square$

    $\longrightarrow \quad n_0 = 7, \; C = 6$
  - Alternativer Beweis:

    $5 \cdot n + 7 \leq 5 \cdot n + 7 \cdot n \qquad (\text{für } n \geq 1)$

    $\leq 12 \cdot n \qquad \square$

    $\longrightarrow \quad n_0 = 1, \; C = 12$

Beispiele Gross-O

- Es interessiert nur der Term höchster Ordnung, der am schnellsten wachsende Summand
- Terme niedriger Ordnung und Konstanten werden ignoriert.
- f(n) muss nur **von oben** durch c·g(n) beschränkt sein.

$$2 \cdot n^2 + 7 \cdot n - 20 \in \mathcal{O}(n^2) \qquad (\text{auch} \in \mathcal{O}(n^3), \text{ aber irrelevant})$$

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

$$7n \log n - 20 \in \mathcal{O}(n \log n)$$

$$5 \in \mathcal{O}(1)$$

$$2 \cdot n^2 + 7n \log n + n^3 \in \mathcal{O}(n^3)$$

Definition Groß-Omega

UNI FREIBURG

- Groß-Omega, Definition + Beispiel
  - **Intuitiv:** Man sagt f ist Groß-Omega von g ...

    ... wenn f "mindestens so stark wächst wie" g

    Also wie Groß-O, nur mit "mindestens" statt "höchstens"
  - **Formal:** Ω(g) ist auch eine Menge von Funktionen:

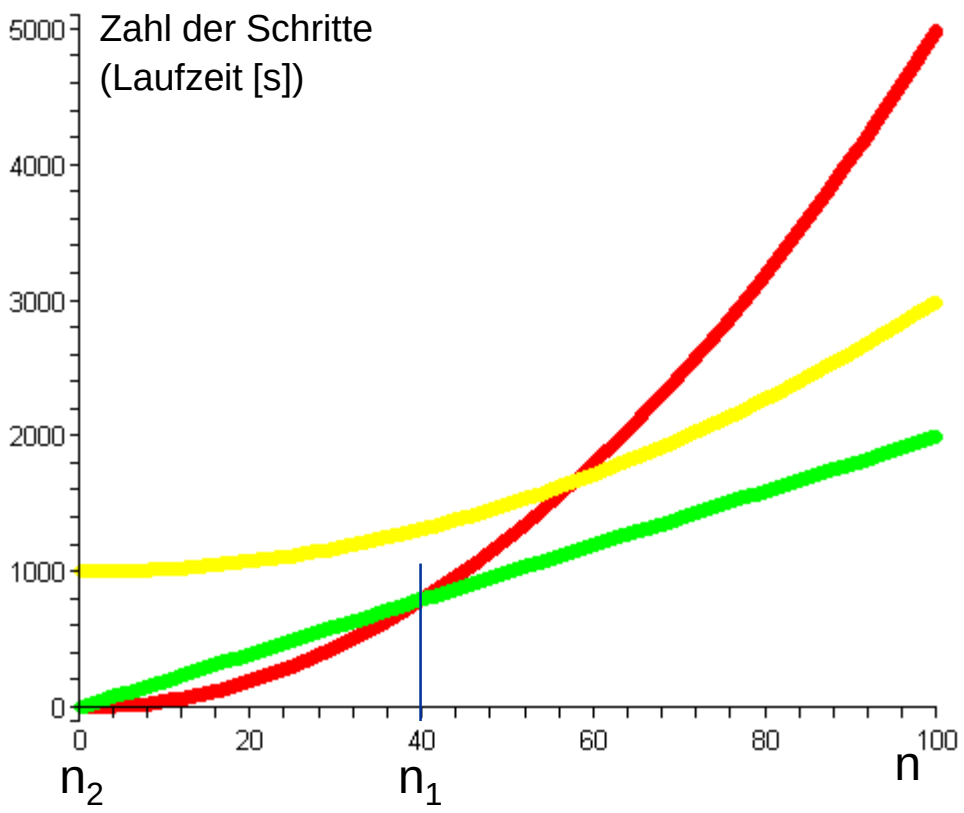
$$\Omega(g) = \{f : \mathbb{N} \to \mathbb{R} \mid \exists n_0 \in \mathbb{N}\ \exists C > 0\ \forall n > n_0 : f(n) \geq C \cdot g(n)\}$$

bei $\mathcal{O}(g)$ war das ein "$\leq$"

  - Zum Beispiel 5 · n + 7 = Ω(n)
  - Beweis unter Verwendung der Definition von Ω :

$$\underbrace{5 \cdot n + 7}_{f(n)} \geq 1 \cdot \underbrace{n}_{g(n)} \qquad (\text{für } n \geq 1) \longrightarrow \begin{array}{l} n_0 = 1 \\ C = 1 \end{array}$$

Illustration Groß-Omega



$f_1(n)$ $\quad f_1 \in \Omega(g)$

$f_2(n)$ $\quad f_2 \in \Omega(g)$

$c \cdot g(n)$

f ist mindestens von der Größenordnung von g.

Beispiele Groß-Omega

UNI FREIBURG

- Es interessiert nur der Term höchster Ordnung, der am schnellsten wachsende Summand
- Terme niedriger Ordnung und Konstanten werden ignoriert.
- f(n) muss nur **von unten** durch c·g(n) beschränkt sein.

$$2 \cdot n^2 + 7 \cdot n - 20 \in \Omega(n^2) \qquad (\text{auch} \in \Omega(n), \text{ aber irrelevant})$$

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

$$7n \log n - 20 \in \Omega(n \log n)$$

$$5 \in \Omega(1)$$

$$2 \cdot n^2 + 7n \log n + n^3 \in \Omega(n^3)$$

Definition Groß-Theta

UNI FREIBURG

- Groß-Theta, Definition + Beispiel
  - **Intuitiv:** Man sagt f ist Theta von g ...

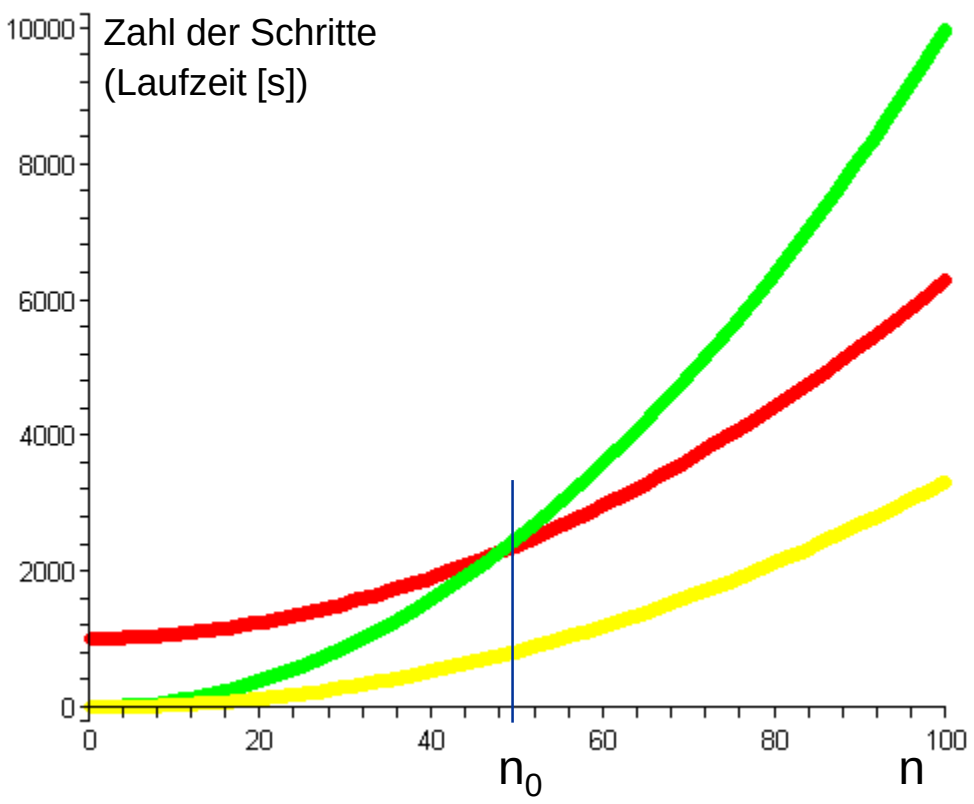
    ... wenn f "genauso so stark wächst wie" g
  - **Formal:** Die Menge $\Theta(g)$ ist definiert als

    $\Theta(g) = \mathcal{O}(g) \cap \Omega(g)$

    $X \cap Y$: die Schnittmenge von X und Y
  - Zum Beispiel $5 \cdot n + 7 = \Theta(n)$
  - Beweis unter Verwendung der Definition von Θ :

$$
\begin{aligned}
& f = \mathcal{O}(g) && \text{(Beweis siehe Folie 8)} \\
& f = \Omega(g) && \text{(Beweis siehe letzte Folie 10)} \\
\Rightarrow \quad & f = \Theta(g)
\end{aligned}
$$

Illustration Groß-Theta



Obere Schranke

Laufzeit vom Algorithmus

Untere Schranke

f und g haben die gleiche Größenordnung

Landau-Symbole Zusammenfassung

- $f = O(g)$: Groß-O
  - f wächst höchstens so stark wie g
  - $C \cdot g(n)$ ist die obere Schranke
- $f = \Omega(g)$: Groß-Omega
  - f wächst mindestens so stark wie g
  - $C \cdot g(n)$ ist die untere Schranke
- $f = \Theta(g)$: Groß-Theta
  - f wächst genau so stark wie g
  - $C_2 \cdot g(n)$ ist die obere Schranke und $C_1 \cdot g(n)$ ist die untere Schranke

Typische Funktionenklassen zur Effizienzbeschreibung

$f \in \Theta(1)$	zeitkonstant, f ist beschränkt
$f \in \Theta(\log n) = \Theta(\log_2 n)$	logarithmisch wachsende Funktionen
$f \in \Theta(n)$	linear wachsende Funktionen
$f \in \Theta(n \log n)$	n-log-n wachsend, superlinear
$f \in \Theta(n^2)$	quadratisch wachsende Funktionen
$f \in \Theta(n^3)$	kubisch wachsende Funktionen
$f \in \Theta(n^k)$	polynomiell wachsende Funktionen
$f \in \Theta(2^n)$	exponentiell wachsende Funktionen

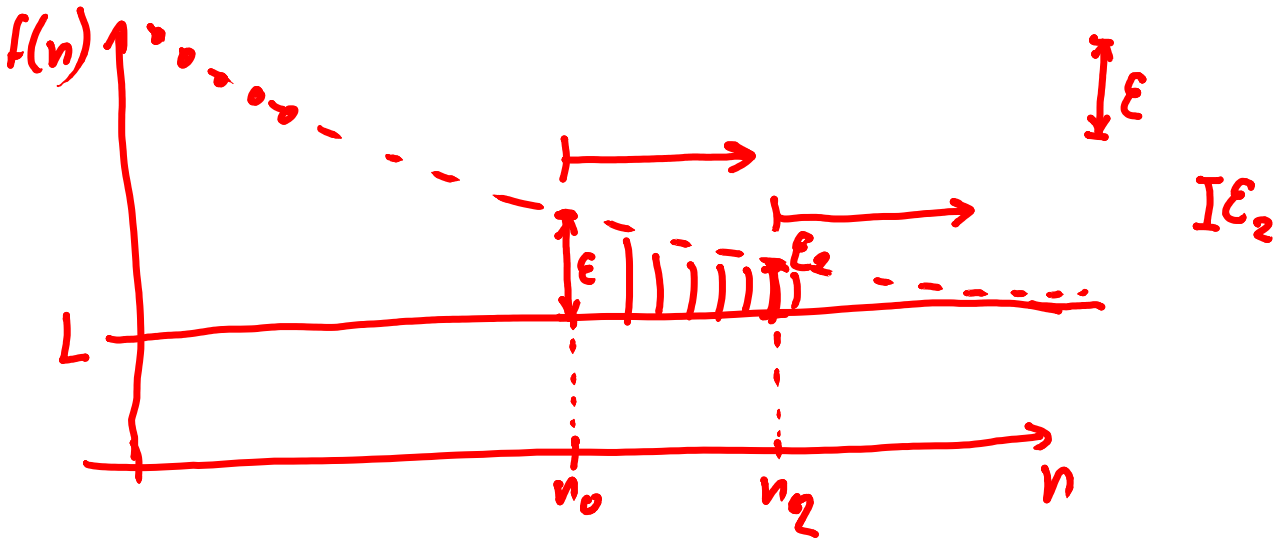
O-Notation — Grenzwerte 1/6

UNI FREIBURG

- In den bisherigen Beispielen ...
  - ... haben wir die Zugehörigkeit zu $O(...)$ etc. gewissermaßen "zu Fuß" bewiesen, indem wir explizit das n_0 und das C bestimmt haben
  - Die Definitionen erinnern aber sehr an den **Grenzwertbegriff** aus der **Analysis**

O-Notation — Grenzwerte 2/6

- Grenzwert Definition
  - Eine unendliche Folge $f_1, f_2, f_3, \ldots$ hat einen Grenzwert L, wenn für alle $\varepsilon > 0$ ein $n_0 \in N$ existiert so dass für alle $n \geq n_0$ gilt dass $|f_n - L| \leq \varepsilon$
  - In Symbolen schreibt man dann $\lim_{n\to\infty} f_n = L$
  - Eine Funktion $f : N \to R$ kann man genauso gut als Folge $f(1), f(2), f(3), \ldots$ auffassen und schreibt $\lim_{n\to\infty} f(n) = L$



O-Notation — Grenzwerte 3/6

UNI FREIBURG

- Beispiel für einen Beweis von einem Grenzwert
  (sollten Sie eigentlich in Mathe 1 schon mal gesehen haben)

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

Ingenieurslösung: ∞ einsetzen: $\frac{1}{\infty} = 0$

Zu zeigen: Für jedes beliebige vorgegebene $\epsilon > 0$
gibt es ein n_0, so dass für $\forall n \geq n_0$ gilt:

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

z.B. für $\epsilon = 0.01$ ist $\frac{1}{n} \leq \epsilon$ ab $n \geq 100$

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

damit gilt für $n \geq n_0$:

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

O-Notation — Grenzwerte 4/6

UNI FREIBURG

- Satz
  - Seien $f, g : \mathbb{N} \to \mathbb{R}$ und der Grenzwert $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ existiert (evtl. ist er ∞)
  - Dann gelten

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

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

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

O-Notation — Grenzwerte 5/6

- Beweis von (1) ... die anderen Beweise gehen analog

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

Beweis von "$\Rightarrow$":

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

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

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

O-Notation — Grenzwerte 6/6

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

Beweis von "$\Leftarrow$":

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty \quad \Rightarrow \quad \lim_{n\to\infty} \frac{f(n)}{g(n)} = C \quad \text{für irgendein } C < \infty$$

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

$$\Rightarrow \quad \exists n_0\, \forall n \geq n_0 : \quad f(n) \leq \underline{(C+1)} \cdot g(n)$$

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

Die Konst. von der O-Notation

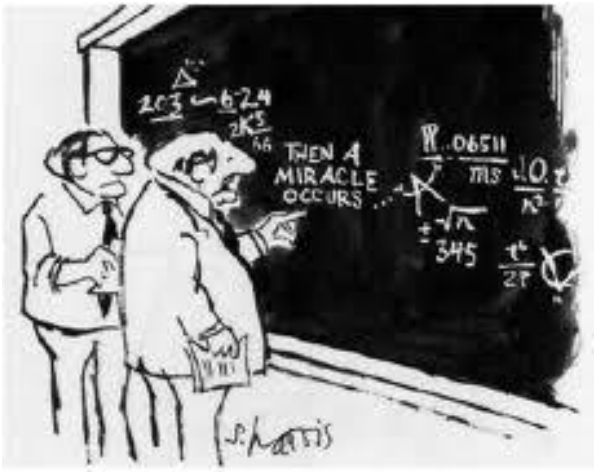
Rechnen mit Grenzwerten 1/2

UNI FREIBURG

- Variante 1: "zu Fuß"
  - Dafür hatten wir gerade das Beispiel $\lim_{n\to\infty} 2 + \frac{1}{n} = 2$
- Variante 2: Regel von L'Hôpital
  - Seien $f, g : \mathbb{N} \to \mathbb{R}$ wie gehabt
  - Es existieren die ersten Ableitungen f' und g', sowie der Grenzwert $\lim_{n\to\infty} \frac{f'(n)}{g'(n)}$, dann gilt

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

- Variante 3: göttliche Inspiration
  - Erst mit Promotion erlaubt ...



"I think you should be more explicit here in step two."

from *What's so Funny about Science?* by Sidney Harris (1977)

Beispiel L'Hôpital

- Grenzwert lässt sich ingenieursmäßig nicht bestimmen:

$$\lim_{n\to\infty} \frac{\ln n}{n} = ? \qquad \text{einsetzen ergibt: } = \frac{\infty}{\infty}$$

- Mit L'Hôpital: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$
  - Zähler: $f(n) := \ln n$, Nenner: $g(n) := n$
  - Ableitung vom Zähler existiert: $f'(n) = \frac{1}{n}$
  - Ableitung von Nenner existiert: $g'(n) = 1$
  - Grenzwert existiert auch: $\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} \frac{1/n}{1} = \lim_{n\to\infty} \frac{1}{n} = 0$
  - Also ist: $\lim_{n\to\infty} \frac{\ln n}{n} = \lim_{n\to\infty} \frac{1}{n} = 0$

Rechnen mit Grenzwerten 2/2

UNI FREIBURG

- Was darf man ohne Beweis annehmen?
  - Gute Frage!
  - Da gibt es keine klare Regel
  - Im Zweifelsfall immer mehr beweisen als weniger !
  - **Beispiel 1:** $\lim_{n\to\infty} 1/n = 0$

    Brauchen Sie nicht mehr weiter zu beweisen
  - **Beispiel 2:** $\lim_{n\to\infty} 1/n^2 = 0$

    Einfach auf sowas wie Beispiel 1 zurückführen
  - **Beispiel 3:** $\lim_{n\to\infty} (\log n)/n = 0$

    Hier ist ein Argument angebracht, z.B. mit L'Hôpital

Praktischer Nutzen

UNI FREIBURG

- Mit der O-Notation können wir viel einfacher die Laufzeit von Algorithmen abschätzen.
- Dazu zuerst ein paar Rechenregeln und dann praktische Anwendungen

Zusammenfassung Eigenschaften

- Transitivität

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

- Reflexivität $f \in \Theta(f) \quad f \in \Omega(f) \quad f \in O(f)$

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

- Austausch-Symmetrie $f \in O(g) \leftrightarrow g \in \Omega(f)$

O-Kalkül

- Einfache Regeln

$$f = O(f)$$
$$kO(f) = O(f)$$
$$O(f + k) = O(f)$$

- Addition

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

- Multiplikation

$$O(f) \cdot O(g) = O(f \cdot g)$$

Laufzeitkomplexität für imperative Sprachkonstrukte

- Problemgröße n in allen Beispielen
- elementare Operationen

`i1 := 0;`	**O(1)**

- Sequenz elementarer Operationen

`i1 := 0;`	O(1)	$327 \cdot O(1) = \mathbf{O(1)}$
`i2 := 0;`	O(1)	
...	...	
`i327 := 0;`	O(1)	

wenn 327 konstant bzw. unabhängig von n ist

Laufzeitkomplexität für imperative Sprachkonstrukte

- Schleife

`for i := 1 to n do`	O(n)	O(1) · O(n) = **O(n)**
`begin`		
`  a[i] := 0;`	O(1)	
`end;`		

`for i := 1 to n do`	O(n)	O(n)	O(1) · O(n) = **O(n)**
`begin`			
`  a1[i] := 0;`	O(1)	37 · O(1) = O(1)	
`  …`	…		
`  a37[i] := 0;`	O(1)		
`end;`			

Laufzeitkomplexität für imperative Sprachkonstrukte

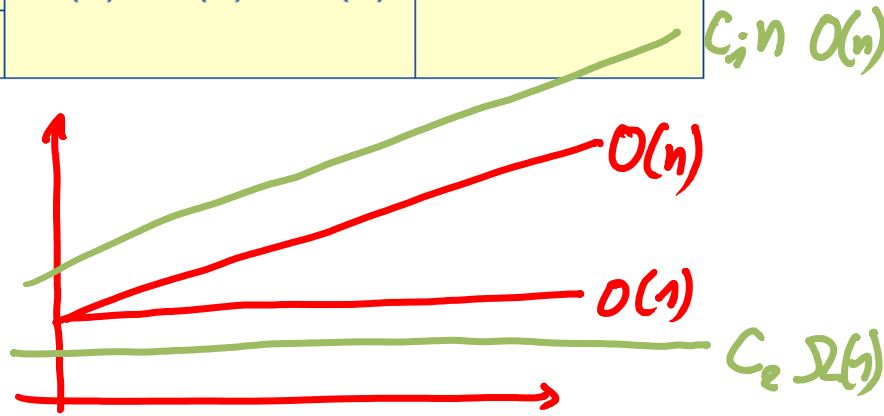
- Schleife

`for i := 1 to n do`	$O(n)$	$O(n) \cdot O(n-1) =$ $O(n) \cdot O(n) = O(n^2)$	$O(1) \cdot O(n^2) = \mathbf{O(n^2)}$
`  for j := 1 to n-1 do`	$O(n-1)$		
`  begin`		$137 \cdot O(1) = O(1)$	
`    a1[i][j] := j;`	$O(1)$		
`    …`	…		
`    a137[i][j] := i;`	$O(1)$		
`  end;`			

Laufzeitkomplexität für imperative Sprachkonstrukte

- bedingte Anweisung

`if x<100 then`	O(1)	O(1)	O(max{1,n}) = **O(n)**
`  y:=x;`	O(1)		
`else`		O(n) · O(1) = O(n)	
`  for i:=1 to n`	O(n)		
`  if a[i]>y then`	O(1)		
`  y:=a[i];`	O(1)		



Beispiel
Arithmetisches Mittel

- Eingabe: Feld x mit n Zahlen
- Ausgabe: Feld a mit n Zahlen, wobei a[i] dem arithmetischen Mittel von x[0] bis x[i] entspricht

```
for i:=0 to n-1 do
   begin
   s:=0;
   for j:=0 to i do
      begin
      s:=s+x[j];
      end;
   a[i]:=s/(i+1);
   end;
return a;
```

Laufzeitkomplexität

`for i:=0 to n-1 do`	O(n)	O(n)	O(n) · O(i) = **O(n^2)**
`begin`			
`  s:=0;`	O(1)		
`  for j:=0 to i do`	O(i+1)	O(i)	
`  s:=s+x[j];`	O(1)		
`  a[i]:=s/(i+1);`	O(1)	O(1)	
`end;`			

i ist von n abhängig, kann nicht als konstant angesehen werden.

- Wie oft werden Anweisungen in der inneren Schleife in Abhängigkeit von der Problemgröße n ausgeführt?

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

Gaußsche Summenformel

O-Notation — Diskussion 1/2

UNI FREIBURG

- Sprechweise
  - Die O-Notation schaut sich das Verhalten der Funktionen an, wenn $n \to \infty$ geht (es interessieren nur die $n \geq n_0$)
  - Wenn man Laufzeiten, Kosten, etc. als $O(...)$ oder $\Omega(...)$ oder $\Theta(...)$ ausdrückt, spricht man daher von

    asymptotischer Analyse
- Vorsicht
  - Asymptotische Analyse sagt nichts über das Laufzeitverhalten bei "kleinen" Eingabegrößen ($n < n_0$) aus
  - Für $n < 2$ oder $n < 10$ ist das egal, da wird schon nichts Schlimmes passieren
  - Aber das n_0 ist nicht immer so klein ... nächste Folie

O-Notation — Diskussion 2/2

UNI FREIBURG

- Beispiel
  - Algorithmus A hat Laufzeit $f(n) = 80 \cdot n$
  - Algorithmus B hat Laufzeit $g(n) = 2 \cdot n \cdot \log_2 n$
  - Dann ist $f = O(g)$ aber **nicht** $f = \Theta(g)$
  - Das heißt, A ist asymptotisch schneller als B
    - d.h. ab irgendeinem n_0 ist für alle $n \geq n_0$ $f(n) \leq g(n)$
  - Berechnung von n_0: bei n_0 schneiden sich die beiden Funktionen:

$$f(n_0) = g(n_0)$$
$$80 \cdot n_0 = 2 \cdot n_0 \cdot \log_2 n_0$$
$$40 = \log_2 n_0$$
$$n_0 = 2^{40} \quad = (2^{10})^4 = (1024)^4 \approx (10^3)^4 = 10^{12} = \text{ 1 Billionen}$$

  - D.h., A ist erst ab 1 Billionen Elemente schneller als B

Literatur / Links

- O-Notation / Ω-Notation / Θ-Notation
  - In Mehlhorn/Sanders:

    2.1 Asymptotic Notation
  - In Cormen/Leiserson/Rivest

    2.1 Asymptotic Notation
  - In Wikipedia

    http://en.wikipedia.org/wiki/Big_O_notation

    http://de.wikipedia.org/wiki/Landau-Symbole

Übungsblatt 3

UNI FREIBURG

- Nochmal Beweise.
- Aufgabe 3 ist direkt aus der letzten Klausur übernommen