

# $\mathcal{O}$ -Notation

Seit ihrer ersten Verwendung durch Bachmann im Jahr 1892 hat sich die  $\mathcal{O}$ -Notation (gesprochen: "groß Oh") durchgesetzt, um die Wachstumsgeschwindigkeit oder Größenordnung von Funktionen  $f : \mathbb{N} \rightarrow \mathbb{R}^+$  und damit von Rechenzeiten zu messen. Ziel ist es, die Beziehungen " $\leq$ ", " $\geq$ ", " $=$ ", " $<$ " und " $>$ " auch auf Funktionen anzuwenden, wobei die strikte Definition  $f \leq g$ , falls  $f(n) \leq g(n) \forall n \in \mathbb{N}$  ist, durch eine abgeschwächte Bedingung ersetzt wird.

## 1 Definitionen

" $\leq$ "  $g = \mathcal{O}(f)$  hat die Interpretation, daß  $g$  asymptotisch nicht schneller als  $f$  wächst, und ist definiert durch die Bedingung, daß  $\frac{g(n)}{f(n)}$  durch eine Konstante  $c$  nach oben beschränkt ist. D.h.:

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

" $\geq$ "  $g = \Omega(f)$  hat die Interpretation, daß  $g$  asymptotisch nicht langsamer als (oder mindestens so schnell wie)  $f$  wächst, und ist definiert durch  $f = \mathcal{O}(g)$ . D.h.:

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

" $=$ "  $g = \Theta(f)$  hat die Interpretation, daß  $f$  und  $g$  asymptotisch gleich schnell wachsen, und ist definiert durch  $g = \mathcal{O}(f)$  und  $g = \Omega(f)$ . D.h.  $\Theta(f) = \mathcal{O}(f) \cap \Omega(f)$  bzw.:

$$\Theta(f) = \{g \mid g : \mathbb{N} \rightarrow \mathbb{R}^+, \exists c > 0 \exists n_0 \in \mathbb{N} \forall n \geq n_0 : \frac{1}{c} \cdot f(n) \leq g(n) \leq c \cdot f(n)\}$$

" $<$ "  $g = o(f)$  hat die Interpretation, daß  $g$  asymptotisch langsamer als  $f$  wächst, und ist definiert durch die Bedingung, daß  $\frac{g(n)}{f(n)}$  eine Nullfolge ist. D.h.:

$$o(f) = \{g \mid g : \mathbb{N} \rightarrow \mathbb{R}^+, \forall c > 0 \exists n_0 \in \mathbb{N} \forall n \geq n_0 : g(n) < c \cdot f(n)\}$$

" $>$ "  $g = \omega(f)$  hat die Interpretation, daß  $g$  asymptotisch schneller als  $f$  wächst, und ist definiert durch  $f = o(g)$ . D.h.:

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

**Folgerung:** Es gilt  $f = \mathcal{O}(f)$ ,  $f = \Omega(f)$  und  $f = \Theta(f)$ .

**Bemerkung:** Man findet in der Literatur häufig die Schreibweisen  $g \in \mathcal{O}(f)$  anstelle von  $g = \mathcal{O}(f)$  und  $X \subseteq \mathcal{O}(f)$  anstelle von  $X = \mathcal{O}(f)$ . Verwendet man die Schreibweise  $g = \mathcal{O}(f)$ , so müssen solche Beziehungen stets von links nach rechts gelesen werden und sind nicht umkehrbar. Sprich: "*g ist von der Größenordnung  $\mathcal{O}(f)$* ".

## 2 Ein paar Rechengesetze zur $\mathcal{O}$ -Notation

- $c \cdot f = \mathcal{O}(f)$  für  $c \geq 0$  konstant
- $\mathcal{O}(c \cdot f) = \mathcal{O}(f)$  für  $c \geq 0$  konstant
- $c \cdot \mathcal{O}(f) = \mathcal{O}(f)$  für  $c \geq 0$  konstant
- $\mathcal{O}(\mathcal{O}(f)) = \mathcal{O}(f)$
- $\mathcal{O}(f_1) + \dots + \mathcal{O}(f_k) = \mathcal{O}(f_1 + \dots + f_k) = \mathcal{O}(\max\{f_1, \dots, f_k\})$  für  $k$  konstant
- $\mathcal{O}(f) \cdot \mathcal{O}(g) = \mathcal{O}(f \cdot g)$
- $g = \mathcal{O}(f) \Rightarrow \mathcal{O}(f + g) = \mathcal{O}(f)$

Und noch einige nützliche Hinweise:

- Potenzen von  $n$  sind bzgl. des Exponenten geordnet, d.h.  $n^a = \mathcal{O}(n^b)$  für  $a \leq b$
- Die Ordnung von  $\log n$  ist unabhängig von der gewählten Basis, d.h.  $\log_a n = \mathcal{O}(\log_b n)$ ,  $\forall a, b > 1$
- Der Logarithmus wächst langsamer als jede Potenz von  $n$ , d.h.  $\log n = \mathcal{O}(n^a)$ ,  $\forall a > 0$ ; aber  $n^a \neq \mathcal{O}(\log n)$
- $n^a = \mathcal{O}(b^n)$ ,  $\forall a, b > 1$ ; aber  $b^n \neq \mathcal{O}(n^a)$  für  $b > 1$

### Beispiele:

a)  $f_1(n) = 2n^2 + 5n + 12$

Es gilt  $12 \leq 5n \forall n \geq 3$ , also  $12 = \mathcal{O}(n)$ ,

$5n + 12 \leq 2n^2 \forall n \geq 4$ , also  $5n + 12 = \mathcal{O}(n^2)$ ,

$2n^2 + 5n + 12 \leq 3n^2 \forall n \geq 7$ , also  $f_1(n) = \mathcal{O}(n^2)$ .

Ferner gilt  $2n^2 + 5n + 12 \geq n^2 \forall n \geq 0$ , also  $f_1 = \Omega(n^2)$  und folglich  $f_1(n) = \Theta(n^2)$ .

b)  $f_2(n) = 5n^5 - 27n^3 + 2n - 5112$

Es gilt  $f_2(n) \leq 5n^5 \forall n \geq 0$ , also  $f_2(n) = \mathcal{O}(n^5)$ .

Ferner gilt  $f_2(n) \geq 5n^5 - 3n^5 = 2n^5 \forall n \geq 5$ , also  $f_2(n) = \Omega(n^5)$  und folglich  $f_2(n) = \Theta(n^5)$ .

c)  $f_3(n) = 2^n + n^2$

Wegen  $2^n + n^2 \leq 2 \cdot 2^n \forall n \geq 4$  gilt  $f_3(n) = \mathcal{O}(2^n)$ . Außerdem ist  $f_3(n) \geq 2^n \forall n \geq 0$  und damit  $f_3(n) = \Theta(2^n)$ .

d)  $f_4(n) = \lceil \frac{n}{\log n} \rceil$

Aus  $\lim_{n \rightarrow \infty} \frac{\frac{n}{\log n}}{n} = \lim_{n \rightarrow \infty} \frac{1}{\log n} = 0$  folgt, daß  $f_4(n) = o(n)$  und damit  $f_4(n) = \mathcal{O}(n)$ .

e)  $f_5(n) = n^2 \log^2 n + 10 \frac{n^3}{\log n} + 5n$

$\frac{n^3}{\log n}$  ist der am schnellsten wachsende Summand, also gilt  $f_5(n) = \mathcal{O}(\frac{n^3}{\log n})$ .

### 3 Wachstumsverhalten von Funktionen

Eine Funktion  $f : \mathbb{N} \rightarrow \mathbb{R}^+$  heißt

- *konstant*, wenn  $f = \mathcal{O}(1)$  ist,
- *logarithmisch* wachsend, wenn  $f = \mathcal{O}(\log n)$  ist,
- *polylogarithmisch* wachsend, wenn  $f = \mathcal{O}(\log^k n)$  für ein  $k \in \mathbb{N}$  ist, also wenn  $f$  asymptotisch nicht schneller als ein Polynom in  $\log n$  wächst,
- *linear/quadratisch/kubisch* wachsend, wenn  $f = \mathcal{O}(n)/f = \mathcal{O}(n^2)/f = \mathcal{O}(n^3)$  ist,
- *quasilinear* wachsend, wenn  $f = \mathcal{O}(n \log^k n)$  für ein  $k \in \mathbb{N}$  ist,
- *polynomiell* wachsend, wenn  $f = \mathcal{O}(n^k)$  für ein  $k \in \mathbb{N}$  ist,
- *faktoriell* wachsend, wenn  $f = \mathcal{O}(n!)$  ist,
- *exponentiell* wachsend, wenn  $f = \Omega(2^{n^\varepsilon})$  für ein  $\varepsilon > 0$  ist,
- *echt exponentiell* wachsend, wenn  $f = \Omega(2^{\varepsilon n})$  für ein  $\varepsilon > 0$  ist.

Wir wollen uns nun zwei Dinge im Zusammenhang mit der Zeitkomplexität verdeutlichen:

#### 3.1 Zeitaufwand für kleine Problemfälle

**Beispiel 1** Seien  $P$  ein Problem und  $A_i$ ,  $i = 1, \dots, 4$  Algorithmen zur Lösung von  $P$  mit dem Zeitaufwand:

$$A_1 : 1000n, \quad A_2 : 100n \cdot \log_2 n, \quad A_3 : 10n^2, \quad A_4 : 2^n$$

Der schnellste Algorithmus für Problemgrößen  $n$  ist:

$$\begin{aligned} 2 \leq n \leq 9 & : A_4 \\ 10 \leq n \leq 58 & : A_3 \\ 59 \leq n \leq 1024 & : A_2 \\ 1025 \leq n & : A_1 \end{aligned}$$

$n$	4	16	64	256	1024	4096
$1000n$	4000	16000	64000	256000	1024000	4096000
$100n \cdot \log_2 n$	800	6400	38400	204800	1024000	$\approx 4915200$
$10n^2$	160	2560	40960	655360	10485760	167772160
$2^n$	16	65536	$\approx 10^{19}$	$\approx 10^{70}$	$\approx 10^{308}$	$\approx 10^{1233}$

Ein großes Wachstum kann für kleine Problemfälle sehr wohl einen schnellen Algorithmus liefern!

## 3.2 Auswirkung verschiedener Wachstumsgesetze bei großen Problemfällen

Für die folgenden Übersichten nehmen wir an, daß der Zeitbedarf in Einheiten von 1 Millisekunde angegeben wird.

### Beispiel 2

- Welche Größe eines Problemfalls kann in einer bestimmten Zeit gelöst werden?

Algorithmus	Zeitkomplexität	maximale Fallgröße zu berechnen in		
		1 sec	1 min	1 h
$A_1$	$n$	1000	60000	3600000
$A_2$	$n \cdot \log_2 n$	140	4895	204095
$A_3$	$n^2$	31	244	1897
$A_4$	$2^n$	9	15	21

Rechenbeispiel:  $A_4$ , 1 sec:

Gesucht ist  $\max\{n \mid 2^n \leq 1000\}$ . Es ergibt sich  $n = \lfloor \frac{\ln 1000}{\ln 2} \rfloor$ .

- Wieviel kann man einen gegebenen Problemfall vergrößern, wenn man einen 10mal schnelleren Prozessor einsetzt und dieselbe Rechenzeit gebraucht?

Algorithmus	Zeitkomplexität	maximale Fallgröße vor   nach der Beschleunigung	
		$A_1$	$n$
$A_2$	$n \cdot \log_2 n$	$g_2$	$\approx 10g_2$ für große $g_2$
$A_3$	$n^2$	$g_3$	$3.16g_3$
$A_4$	$2^n$	$g_4$	$g_4 + 3.3$

Rechenbeispiel:  $A_3$ :

Gesucht ist  $\max\{n \mid g_3^2 \geq \frac{1}{10}n^2\}$ . Es folgt  $n \leq \sqrt{10g_3^2} = \sqrt{10}g_3 \approx 3.16g_3$ .

Rechenbeispiel:  $A_4$ :

Gesucht ist  $\max\{n \mid 2^{g_4} \geq \frac{1}{10} \cdot 2^n\}$ . Es folgt  $n \leq \log_2(10 \cdot 2^{g_4}) = \log_2 10 + \log_2 2^{g_4} = \log_2 10 + g_4 \approx 3.3 + g_4$ .

Es ist oft nicht sinnvoll, mehr Rechenzeit zu investieren oder schnellere Rechner einzusetzen, wenn ein Problem in vertretbarem Zeitaufwand nicht mehr lösbar ist. Die Suche nach schnelleren Algorithmen kann hier mehr Erfolg bringen.