

Asymptotische Notation

Wir interessieren uns normalerweise nicht für exakte Laufzeiten, sondern für eine **asymptotische Klassifikation** der Laufzeit, die konstante Faktoren und additive Terme ignoriert.

Asymptotische Notation

Wir interessieren uns normalerweise nicht für exakte Laufzeiten, sondern für eine **asymptotische Klassifikation** der Laufzeit, die konstante Faktoren und additive Terme ignoriert.

- ▶ Wir interessieren uns für Laufzeiten bei großen Werten von n . Konstante additive Terme sind dann unwichtig.
- ▶ Eine superexakte Analyse (e.g. das *exakte* Zählen der Operationen auf einer RAM) ist schwierig, und würde die Resultate nicht verbessern, da das Rechenmodell die Realität nicht so exakt abbildet.
- ▶ Ein linearer speed-up (z.B. um einen konstanten Faktor) läßt sich z.B. erreichen wenn man den Algorithmus auf einem schnelleren Rechner laufen läßt.
- ▶ Laufzeiten sollte man durch einfache Funktionen ausdrücken können.

Asymptotische Notation

Wir interessieren uns normalerweise nicht für exakte Laufzeiten, sondern für eine **asymptotische Klassifikation** der Laufzeit, die konstante Faktoren und additive Terme ignoriert.

- ▶ Wir interessieren uns für Laufzeiten bei großen Werten von n . Konstante additive Terme sind dann unwichtig.
- ▶ Eine superexakte Analyse (e.g. das *exakte* Zählen der Operationen auf einer RAM) ist schwierig, und würde die Resultate nicht verbessern, da das Rechenmodell die Realität nicht so exakt abbildet.
- ▶ Ein linearer speed-up (z.B. um einen konstanten Faktor) läßt sich z.B. erreichen wenn man den Algorithmus auf einem schnelleren Rechner laufen läßt.
- ▶ Laufzeiten sollte man durch einfache Funktionen ausdrücken können.

Asymptotische Notation

Wir interessieren uns normalerweise nicht für exakte Laufzeiten, sondern für eine **asymptotische Klassifikation** der Laufzeit, die konstante Faktoren und additive Terme ignoriert.

- ▶ Wir interessieren uns für Laufzeiten bei großen Werten von n . Konstante additive Terme sind dann unwichtig.
- ▶ Eine superexakte Analyse (e.g. das *exakte* Zählen der Operationen auf einer RAM) ist schwierig, und würde die Resultate nicht verbessern, da das Rechenmodell die Realität nicht so exakt abbildet.
- ▶ Ein linearer speed-up (z.B. um einen konstanten Faktor) läßt sich z.B. erreichen wenn man den Algorithmus auf einem schnelleren Rechner laufen läßt.
- ▶ Laufzeiten sollte man durch einfache Funktionen ausdrücken können.

Asymptotische Notation

Wir interessieren uns normalerweise nicht für exakte Laufzeiten, sondern für eine **asymptotische Klassifikation** der Laufzeit, die konstante Faktoren und additive Terme ignoriert.

- ▶ Wir interessieren uns für Laufzeiten bei großen Werten von n . Konstante additive Terme sind dann unwichtig.
- ▶ Eine superexakte Analyse (e.g. das *exakte* Zählen der Operationen auf einer RAM) ist schwierig, und würde die Resultate nicht verbessern, da das Rechenmodell die Realität nicht so exakt abbildet.
- ▶ Ein linearer speed-up (z.B. um einen konstanten Faktor) läßt sich z.B. erreichen wenn man den Algorithmus auf einem schnelleren Rechner laufen läßt.
- ▶ Laufzeiten sollte man durch einfache Funktionen ausdrücken können.

Asymptotische Notation

Formale Definition

Sei f eine Funktion von \mathbb{N} nach \mathbb{R}^+ .

- ▶ $\mathcal{O}(f) = \{g \mid \exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \leq c \cdot f(n)]\}$
(Funktionen die asymptotisch **nicht schneller** als f wachsen)

Asymptotische Notation

Formale Definition

Sei f eine Funktion von \mathbb{N} nach \mathbb{R}^+ .

- ▶ $\mathcal{O}(f) = \{g \mid \exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \leq c \cdot f(n)]\}$
(Funktionen die asymptotisch **nicht schneller** als f wachsen)
- ▶ $\Omega(f) = \{g \mid \exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \geq c \cdot f(n)]\}$
(Funktionen die asymptotisch **nicht langsamer** als f wachsen)

Asymptotische Notation

Formale Definition

Sei f eine Funktion von \mathbb{N} nach \mathbb{R}^+ .

- ▶ $\mathcal{O}(f) = \{g \mid \exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \leq c \cdot f(n)]\}$
(Funktionen die asymptotisch **nicht schneller** als f wachsen)
- ▶ $\Omega(f) = \{g \mid \exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \geq c \cdot f(n)]\}$
(Funktionen die asymptotisch **nicht langsamer** als f wachsen)
- ▶ $\Theta(f) = \Omega(f) \cap \mathcal{O}(f)$
(Funktionen die asymptotisch **gleiches** Wachstum wie f haben)

Asymptotische Notation

Formale Definition

Sei f eine Funktion von \mathbb{N} nach \mathbb{R}^+ .

- ▶ $\mathcal{O}(f) = \{g \mid \exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \leq c \cdot f(n)]\}$
(Funktionen die asymptotisch **nicht schneller** als f wachsen)
- ▶ $\Omega(f) = \{g \mid \exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \geq c \cdot f(n)]\}$
(Funktionen die asymptotisch **nicht langsamer** als f wachsen)
- ▶ $\Theta(f) = \Omega(f) \cap \mathcal{O}(f)$
(Funktionen die asymptotisch **gleiches** Wachstum wie f haben)
- ▶ $o(f) = \{g \mid \forall c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \leq c \cdot f(n)]\}$
(Funktionen die asymptotische **langsamer** als f wachsen)

Asymptotische Notation

Formale Definition

Sei f eine Funktion von \mathbb{N} nach \mathbb{R}^+ .

- ▶ $\mathcal{O}(f) = \{g \mid \exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \leq c \cdot f(n)]\}$
(Funktionen die asymptotisch **nicht schneller** als f wachsen)
- ▶ $\Omega(f) = \{g \mid \exists c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \geq c \cdot f(n)]\}$
(Funktionen die asymptotisch **nicht langsamer** als f wachsen)
- ▶ $\Theta(f) = \Omega(f) \cap \mathcal{O}(f)$
(Funktionen die asymptotisch **gleiches** Wachstum wie f haben)
- ▶ $o(f) = \{g \mid \forall c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \leq c \cdot f(n)]\}$
(Funktionen die asymptotische **langsamer** als f wachsen)
- ▶ $\omega(f) = \{g \mid \forall c > 0 \exists n_0 \in \mathbb{N}_0 \forall n \geq n_0 : [g(n) \geq c \cdot f(n)]\}$
(Funktionen die asymptotisch **schneller** als f wachsen)

Asymptotische Notation

Äquivalente Definition mit Grenzwerten (**gilt nur falls der jeweilige Grenzwert existiert**). f und g seien Funktionen von \mathbb{N}_0 nach \mathbb{R}_0^+ .

$$\blacktriangleright g \in \mathcal{O}(f): 0 \leq \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} < \infty$$

Asymptotische Notation

Äquivalente Definition mit Grenzwerten (**gilt nur falls der jeweilige Grenzwert existiert**). f und g seien Funktionen von \mathbb{N}_0 nach \mathbb{R}_0^+ .

$$\blacktriangleright g \in \mathcal{O}(f): 0 \leq \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} < \infty$$

$$\blacktriangleright g \in \Omega(f): 0 < \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} \leq \infty$$

Asymptotische Notation

Äquivalente Definition mit Grenzwerten (**gilt nur falls der jeweilige Grenzwert existiert**). f und g seien Funktionen von \mathbb{N}_0 nach \mathbb{R}_0^+ .

$$\blacktriangleright g \in \mathcal{O}(f): 0 \leq \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} < \infty$$

$$\blacktriangleright g \in \Omega(f): 0 < \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} \leq \infty$$

$$\blacktriangleright g \in \Theta(f): 0 < \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} < \infty$$

Asymptotische Notation

Äquivalente Definition mit Grenzwerten (**gilt nur falls der jeweilige Grenzwert existiert**). f und g seien Funktionen von \mathbb{N}_0 nach \mathbb{R}_0^+ .

$$\blacktriangleright g \in \mathcal{O}(f): 0 \leq \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} < \infty$$

$$\blacktriangleright g \in \Omega(f): 0 < \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} \leq \infty$$

$$\blacktriangleright g \in \Theta(f): 0 < \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} < \infty$$

$$\blacktriangleright g \in o(f): \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$$

Asymptotische Notation

Äquivalente Definition mit Grenzwerten (**gilt nur falls der jeweilige Grenzwert existiert**). f und g seien Funktionen von \mathbb{N}_0 nach \mathbb{R}_0^+ .

$$\blacktriangleright g \in \mathcal{O}(f): 0 \leq \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} < \infty$$

$$\blacktriangleright g \in \Omega(f): 0 < \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} \leq \infty$$

$$\blacktriangleright g \in \Theta(f): 0 < \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} < \infty$$

$$\blacktriangleright g \in o(f): \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$$

$$\blacktriangleright g \in \omega(f): \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \infty$$

Missbrauch dieser Notation

1. Man schreibt $f = \mathcal{O}(g)$, anstatt $f \in \mathcal{O}(g)$. Dies ist **keine** Gleichheit (wie kann eine Funktion das gleiche wie eine Funktionsmenge sein?).
2. Man schreibt $f(n) = \mathcal{O}(g(n))$, anstatt $f \in \mathcal{O}(g)$, with $f : \mathbb{N} \rightarrow \mathbb{R}^+, n \mapsto f(n)$, and $g : \mathbb{N} \rightarrow \mathbb{R}^+, n \mapsto g(n)$.
3. Man schreibt $\mathcal{O}(f(n)) = \mathcal{O}(g(n))$, anstatt $\mathcal{O}(f(n)) \subseteq \mathcal{O}(g(n))$.

Missbrauch dieser Notation

1. Man schreibt $f = \mathcal{O}(g)$, anstatt $f \in \mathcal{O}(g)$. Dies ist **keine** Gleichheit (wie kann eine Funktion das gleiche wie eine Funktionsmenge sein?).
2. Man schreibt $f(n) = \mathcal{O}(g(n))$, anstatt $f \in \mathcal{O}(g)$, with $f: \mathbb{N} \rightarrow \mathbb{R}^+, n \mapsto f(n)$, and $g: \mathbb{N} \rightarrow \mathbb{R}^+, n \mapsto g(n)$.
3. Man schreibt $\mathcal{O}(f(n)) = \mathcal{O}(g(n))$, anstatt $\mathcal{O}(f(n)) \subseteq \mathcal{O}(g(n))$.

Missbrauch dieser Notation

1. Man schreibt $f = \mathcal{O}(g)$, anstatt $f \in \mathcal{O}(g)$. Dies ist **keine** Gleichheit (wie kann eine Funktion das gleiche wie eine Funktionsmenge sein?).
2. Man schreibt $f(n) = \mathcal{O}(g(n))$, anstatt $f \in \mathcal{O}(g)$, with $f : \mathbb{N} \rightarrow \mathbb{R}^+, n \mapsto f(n)$, and $g : \mathbb{N} \rightarrow \mathbb{R}^+, n \mapsto g(n)$.
3. Man schreibt $\mathcal{O}(f(n)) = \mathcal{O}(g(n))$, anstatt $\mathcal{O}(f(n)) \subseteq \mathcal{O}(g(n))$.

Asymptotische Notation

Man kann einen Ausdruck mit asymptotischer Notation als **Menge** ansehen:

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

repräsentiert

$$\{f : \mathbb{N} \rightarrow \mathbb{R}^+ \mid f(n) = n^2 \cdot g(n) + h(n)$$

$$\text{mit } g(n) \in \mathcal{O}(n) \text{ und } h(n) \in \mathcal{O}(\log n)\}$$

Asymptotische Notation

Lemma 1

Seien f, g Funktionen mit $\exists n_0 > 0 \forall n \geq n_0 : f(n) > 0$ (das gleiche für g). Dann

- ▶ $c \cdot f(n) = \mathcal{O}(f(n))$ für eine beliebige Konstante c
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$
- ▶ $\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n))$
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(\max\{f(n), g(n)\})$

Alle obigen Relationen gelten auch für Ω und Θ .

Asymptotische Notation

Lemma 1

Seien f, g Funktionen mit $\exists n_0 > 0 \forall n \geq n_0 : f(n) > 0$ (das gleiche für g). Dann

- ▶ $c \cdot f(n) = \mathcal{O}(f(n))$ für eine beliebige Konstante c
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$
- ▶ $\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n))$
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(\max\{f(n), g(n)\})$

Alle obigen Relationen gelten auch für Ω und Θ .

Asymptotische Notation

Lemma 1

Seien f, g Funktionen mit $\exists n_0 > 0 \forall n \geq n_0 : f(n) > 0$ (das gleiche für g). Dann

- ▶ $c \cdot f(n) = \mathcal{O}(f(n))$ für eine beliebige Konstante c
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$
- ▶ $\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n))$
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(\max\{f(n), g(n)\})$

Alle obigen Relationen gelten auch für Ω und Θ .

Asymptotische Notation

Lemma 1

Seien f, g Funktionen mit $\exists n_0 > 0 \forall n \geq n_0 : f(n) > 0$ (das gleiche für g). Dann

- ▶ $c \cdot f(n) = \mathcal{O}(f(n))$ für eine beliebige Konstante c
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$
- ▶ $\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n))$
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(\max\{f(n), g(n)\})$

Alle obigen Relationen gelten auch für Ω und Θ .

Asymptotische Notation

Lemma 1

Seien f, g Funktionen mit $\exists n_0 > 0 \forall n \geq n_0 : f(n) > 0$ (das gleiche für g). Dann

- ▶ $c \cdot f(n) = \mathcal{O}(f(n))$ für eine beliebige Konstante c
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$
- ▶ $\mathcal{O}(f(n)) \cdot \mathcal{O}(g(n)) = \mathcal{O}(f(n) \cdot g(n))$
- ▶ $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(\max\{f(n), g(n)\})$

Alle obigen Relationen gelten auch für Ω und Θ .

Rechenregel für \mathcal{O} -Notation

Zu zeigen:

$$T_1(n) + T_2(n) = \mathcal{O}(\max(f(n), g(n)))$$

für $T_1(n) \in \mathcal{O}(f(n))$ und $T_2(n) \in \mathcal{O}(g(n))$.

- ▶ da $T_1(n) = \mathcal{O}(f(n))$, gibt es $c_1 > 0$ und $n_1 \in \mathbb{N}$ mit $T_1(n) \leq c_1 f(n)$ für $n \geq n_1$
- ▶ da $T_2(n) = \mathcal{O}(g(n))$, gibt es $c_2 > 0$ und $n_2 \in \mathbb{N}$ mit $T_2(n) \leq c_2 g(n)$ für $n \geq n_2$
- ▶ Setze $n_0 := \max(n_1, n_2)$, dann ist für $n \geq n_0$

$$\begin{aligned} T_1(n) + T_2(n) &\leq c_1 f(n) + c_2 g(n) \\ &\leq (c_1 + c_2) \max(f(n), g(n)) \quad \checkmark \end{aligned}$$

Rechenregel für \mathcal{O} -Notation

Zu zeigen:

$$T_1(n) + T_2(n) = \mathcal{O}(\max(f(n), g(n)))$$

für $T_1(n) \in \mathcal{O}(f(n))$ und $T_2(n) \in \mathcal{O}(g(n))$.

- ▶ da $T_1(n) = \mathcal{O}(f(n))$, gibt es $c_1 > 0$ und $n_1 \in \mathbb{N}$ mit $T_1(n) \leq c_1 f(n)$ für $n \geq n_1$
- ▶ da $T_2(n) = \mathcal{O}(g(n))$, gibt es $c_2 > 0$ und $n_2 \in \mathbb{N}$ mit $T_2(n) \leq c_2 g(n)$ für $n \geq n_2$
- ▶ Setze $n_0 := \max(n_1, n_2)$, dann ist für $n \geq n_0$

$$\begin{aligned} T_1(n) + T_2(n) &\leq c_1 f(n) + c_2 g(n) \\ &\leq (c_1 + c_2) \max(f(n), g(n)) \quad \checkmark \end{aligned}$$

Laufzeiten

Funktion	Eingabelänge n							
	10	10^2	10^3	10^4	10^5	10^6	10^7	10^8
$\log n$	33ns	66ns	0.1 μ s	0.1 μ s	0.2 μ s	0.2 μ s	0.2 μ s	0.3 μ s
\sqrt{n}	32ns	0.1 μ s	0.3 μ s	1 μ s	3.1 μ s	10 μ s	31 μ s	0.1ms
n	100ns	1 μ s	10 μ s	0.1ms	1ms	10ms	0.1s	1s
$n \log n$	0.3 μ s	6.6 μ s	0.1ms	1.3ms	16ms	0.2s	2.3s	27s
$n^{3/2}$	0.3 μ s	10 μ s	0.3ms	10ms	0.3s	10s	5.2min	2.7h
n^2	1 μ s	0.1ms	10ms	1s	1.7min	2.8h	11d	3.2y
n^3	10 μ s	10ms	10s	2.8h	115d	317y	$3.2 \cdot 10^5$ y	
1.1^n	26ns	0.1ms	$7.8 \cdot 10^{25}$ y					
2^n	10 μ s	$4 \cdot 10^{14}$ y						
$n!$	36ms	$3 \cdot 10^{142}$ y						

1 Operation = 10ns; 100MHz

Alter des Universums: ca. $13.8 \cdot 10^9$ y

Typische Laufzeitklassen

\mathcal{O} -Notation erlaubt **Klassifizierung** der Effizienz von Algorithmen

$\Theta(1)$: konstante Laufzeit

- ▶ unabhängig von Problemgröße
- ▶ *Beispiel*: Löschen von erstem Element in verketteter Liste

$\Theta(\log n)$: logarithmische Laufzeit

- ▶ Laufzeit wächst langsamer als Problemgröße
- ▶ typisch für Divide & Conquer Algorithmen
- ▶ *Beispiel*: Suchen in sortierter Liste

$\Theta(n)$: lineare Laufzeit

- ▶ Laufzeit wächst vergleichbar zur Problemgröße
- ▶ jedes Eingabe-Element erfordert $\mathcal{O}(1)$ Arbeit
- ▶ *Beispiele*: Suchen in unsortierter Liste, Löschen von Element im Array

Typische Laufzeitklassen

$\Theta(n \log n)$: “loglinear” Laufzeit

- ▶ Laufzeit wächst schneller als Problemgröße
- ▶ typisch für Divide & Conquer
- ▶ *Beispiele*: Quicksort, FFT

$\Theta(n^2)$: quadratische Laufzeit

- ▶ typisch für Algorithmen, die Element paarweise kombinieren
- ▶ *Beispiele*: Insertion Sort, Matrix-Vektor Multiplikation

$\Theta(n^3)$: kubische Laufzeit

- ▶ *Beispiel*: Matrix-Matrix Multiplikation

$\Theta(2^n)$: exponentielle Laufzeit

- ▶ auch als “unlösbar” bezeichnet (intractable)
- ▶ *Beispiel*: Traveling Salesman (kürzeste Route, so dass alle Städte exakt einmal besucht)

Hinweise

- ▶ Man sollte asymptotische Notation nicht in Induktionsbeweisen verwenden.
- ▶ Für beliebige Konstanten a, b gilt $\log_a n = \Theta(\log_b n)$. Deshalb ignorieren wir die Basis des Algorithmus in asymptotischer Notation.
- ▶ Für diese Vorlesung: $\log n = \log_2 n$, d.h., wir nehmen 2 als Standardbasis für den Logarithmus.

Hinweise

- ▶ Man sollte asymptotische Notation nicht in Induktionsbeweisen verwenden.
- ▶ Für beliebige Konstanten a, b gilt $\log_a n = \Theta(\log_b n)$. Deshalb ignorieren wir die Basis des Algorithmus in asymptotischer Notation.
- ▶ Für diese Vorlesung: $\log n = \log_2 n$, d.h., wir nehmen 2 als Standardbasis für den Logarithmus.

Hinweise

- ▶ Man sollte asymptotische Notation nicht in Induktionsbeweisen verwenden.
- ▶ Für beliebige Konstanten a, b gilt $\log_a n = \Theta(\log_b n)$. Deshalb ignorieren wir die Basis des Algorithmus in asymptotischer Notation.
- ▶ Für diese Vorlesung: $\log n = \log_2 n$, d.h., wir nehmen 2 als Standardbasis für den Logarithmus.

Asymptotische Notation

Eine asymptotische Klassifizierung von Laufzeiten ist eine gute Möglichkeit um Effizienz von Algorithmen zu vergleichen:

- ▶ Falls die Laufzeitanalyse genau genug ist und die Laufzeit auch in der Praxis auftaucht (d.h. keine reine worst-case Schranke), dann ist ein Algorithmus mit besserer asymptotischer Laufzeit einem schwächeren für genügend großes n überlegen.
- ▶ Aber:

Asymptotische Notation

Eine asymptotische Klassifizierung von Laufzeiten ist eine gute Möglichkeit um Effizienz von Algorithmen zu vergleichen:

- ▶ Falls die Laufzeitanalyse genau genug ist und die Laufzeit auch in der Praxis auftaucht (d.h. keine reine worst-case Schranke), dann ist ein Algorithmus mit besserer asymptotischer Laufzeit einem schwächeren für genügend großes n überlegen.
- ▶ **Aber:**
 - ▶ Algorithmus A: Laufzeit $f(n) = 1000 \log n = \mathcal{O}(\log n)$.
 - ▶ Algorithmus B: Laufzeit $g(n) = \log^2 n$.

Es gilt $f = o(g)$. Aber solange $\log n \leq 1000$ ist Algorithmus B effizienter.

Asymptotische Notation

Eine asymptotische Klassifizierung von Laufzeiten ist eine gute Möglichkeit um Effizienz von Algorithmen zu vergleichen:

- ▶ Falls die Laufzeitanalyse genau genug ist und die Laufzeit auch in der Praxis auftaucht (d.h. keine reine worst-case Schranke), dann ist ein Algorithmus mit besserer asymptotischer Laufzeit einem schwächeren für genügend großes n überlegen.
- ▶ **Aber:**
 - ▶ Algorithmus A: Laufzeit $f(n) = 1000 \log n = \mathcal{O}(\log n)$.
 - ▶ Algorithmus B: Laufzeit $g(n) = \log^2 n$.

Es gilt $f = o(g)$. Aber solange $\log n \leq 1000$ ist Algorithmus B effizienter.

Asymptotische Notation

Eine asymptotische Klassifizierung von Laufzeiten ist eine gute Möglichkeit um Effizienz von Algorithmen zu vergleichen:

- ▶ Falls die Laufzeitanalyse genau genug ist und die Laufzeit auch in der Praxis auftaucht (d.h. keine reine worst-case Schranke), dann ist ein Algorithmus mit besserer asymptotischer Laufzeit einem schwächeren für genügend großes n überlegen.
- ▶ **Aber:**
 - ▶ Algorithmus A: Laufzeit $f(n) = 1000 \log n = \mathcal{O}(\log n)$.
 - ▶ Algorithmus B: Laufzeit $g(n) = \log^2 n$.

Es gilt $f = o(g)$. Aber solange $\log n \leq 1000$ ist Algorithmus B effizienter.

Asymptotische Notation

Eine asymptotische Klassifizierung von Laufzeiten ist eine gute Möglichkeit um Effizienz von Algorithmen zu vergleichen:

- ▶ Falls die Laufzeitanalyse genau genug ist und die Laufzeit auch in der Praxis auftaucht (d.h. keine reine worst-case Schranke), dann ist ein Algorithmus mit besserer asymptotischer Laufzeit einem schwächeren für genügend großes n überlegen.
- ▶ **Aber:**
 - ▶ Algorithmus A: Laufzeit $f(n) = 1000 \log n = \mathcal{O}(\log n)$.
 - ▶ Algorithmus B: Laufzeit $g(n) = \log^2 n$.

Es gilt $f = o(g)$. Aber solange $\log n \leq 1000$ ist Algorithmus B effizienter.

Komplexität der elementaren Bausteine

Elementarer Verarbeitungsschritt:

- ▶ $\mathcal{O}(1)$

Sequenz:

- ▶ Addition in \mathcal{O} -Notation

Bedingter Verarbeitungsschritt:

- ▶ Maximum von Komplexität von if und else Block, sowie
- ▶ $\mathcal{O}(1)$ für Auswertung der Bedingung

Wiederholung:

- ▶ Anzahl Wiederholungen multipliziert mit Komplexität Schleifenkörper, sowie
- ▶ Anzahl Wiederholungen + 1 multipliziert mit $\mathcal{O}(1)$ für Auswertung der Schleifenbedingung

Komplexität Datenstrukturenoperationen

Feld via Array:

- ▶ elementAt $\mathcal{O}(1)$, insert $\mathcal{O}(n)$, erase $\mathcal{O}(n)$

Feld via LinkedList:

- ▶ elementAt $\mathcal{O}(n)$, insert $\mathcal{O}(n)$, erase $\mathcal{O}(n)$

Stack via Array:

- ▶ push, pop, top alle $\mathcal{O}(1)$

Stack via LinkedList:

- ▶ push, pop, top alle $\mathcal{O}(1)$

Queue via LinkedList:

- ▶ enqueue, dequeue beide $\mathcal{O}(1)$