Kapitel 3

Analyse von Algorithmen

Wir betrachten hier weniger die Korrektheit als den Ressourcenverbrauch (in erster Linie Laufzeit und Speicherplatz) von Algorithmen und zwar in Abhängigkeit von der Größe der Eingabe. Dabei sollen konstante Faktoren ignoriert werden, weil z.B. die Laufzeit eines elementaren Schrittes und die Größe eines Speicherwortes variieren können.

3.1 Definition (Asymptotisches Wachstum)
Zu einer Funktion $f : \mathbb{N}_0 \to \mathbb{R}$ *wird definiert:*

(i) Die Menge

$$\mathcal{O}(f(n)) = \left\{ g : \mathbb{N}_0 \to \mathbb{R} \; : \; \begin{array}{l} \text{es gibt Konstanten } c, n_0 > 0 \text{ mit} \\ |g(n)| \leq c \cdot |f(n)| \text{ für alle } n > n_0 \end{array} \right\}$$

der Funktionen, die höchstens so schnell wachsen wie f.

(ii) Die Menge

$$\Omega(f(n)) = \left\{ g : \mathbb{N}_0 \to \mathbb{R} \; : \; \begin{array}{l} \text{es gibt Konstanten } c, n_0 > 0 \text{ mit} \\ c \cdot |g(n)| \geq |f(n)| \text{ für alle } n > n_0 \end{array} \right\}$$

der Funktionen, die mindestens so schnell wachsen wie f.

(iii) Die Menge

$$\Theta(f(n)) = \left\{ g : \mathbb{N}_0 \to \mathbb{R} \; : \; \begin{array}{l} \text{es gibt Konstanten } c_1, c_2, n_0 > 0 \text{ mit} \\ c_1 \leq \frac{|g(n)|}{|f(n)|} \leq c_2 \text{ für alle } n > n_0 \end{array} \right\}$$

der Funktionen, die genauso schnell wachsen wie f.

(iv) Die Menge

$$o(f(n)) = \left\{ g : \mathbb{N}_0 \to \mathbb{R} \; : \; \begin{array}{l} \textit{zu jedem } c > 0 \textit{ ex. ein } n_0 > 0 \textit{ mit} \\ c \cdot |g(n)| \leq |f(n)| \textit{ für alle } n > n_0 \end{array} \right\}$$

der Funktionen, die gegenüber f verschwinden.

(v) Die Menge

$$\omega(f(n)) = \left\{ g : \mathbb{N}_0 \to \mathbb{R} \; : \; \begin{array}{l} \textit{zu jedem } c > 0 \textit{ ex. ein } n_0 > 0 \textit{ mit} \\ |g(n)| \geq c \cdot |f(n)| \textit{ für alle } n > n_0 \end{array} \right\}$$

der Funktionen, denen gegenüber f verschwindet.

3.2 Bemerkung
Für Funktionen mit mehreren Veränderlichen sind die Wachstumsklassen entsprechend definiert.

Die erste Aussage ist vor allem für Algorithmen interessant, in denen Teilmengen fester Größe betrachtet werden.

3.3 Satz
Für festes $k \in \mathbb{N}_0$ gilt

$$\binom{n}{k} \in \Theta(n^k) \; .$$

■ **Beweis:** Für alle $n > k =: n_0$ gilt

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!} = \frac{n}{k} \cdot \frac{n-1}{k-1} \cdot \ldots \cdot \frac{n-(k-1)}{k-(k-1)} \; .$$

Wegen $\frac{n}{k} \leq \frac{n-i}{k-i} \leq n$, $i = 0, \ldots, k-1$, folgt daraus

$$\left(\frac{n}{k}\right)^k = \left(\frac{1}{k}\right)^k \cdot n^k \leq \binom{n}{k} \leq n^k$$

und damit $\binom{n}{k} \in \Omega(n^k) \cap \mathcal{O}(n^k) = \Theta(n^k)$. □

In den folgenden Näherungsformeln wird statt der Funktion selbst der Fehler der Abschätzung asymptotisch angegeben, und zwar einmal additiv und einmal multiplikativ. Die Schreibweise bedeutet, dass es in der jeweiligen Wachstumsklasse eine Folge gibt, für die Gleichheit herrscht.

3.4 Satz
Für alle $n \in \mathbb{N}_0$ gilt

(i) $H_n := \sum_{k=1}^{n} \frac{1}{k} = \ln n + \mathcal{O}(1)$

benutzt in Beispiel 1.11 (Teileranzahlen)

(ii) $n! = \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \cdot \left(1 + \Theta\left(\frac{1}{n}\right)\right)$ *(Stirlingformel)*

3.1 Standardanalysen

Worst-, average- und best-case Analyse

Die weitaus meisten Bedarfsanalysen sind für den *worst-case*: Wie groß ist der Ressourcenbedarf im ungünstigsten Fall? Eine Analyse wird jedoch aussagekräftiger, wenn man auch den besten und mittleren Fall hinzunimmt.

3.5 Beispiel (Laufzeitanalyse der binären Suche)
Betrachte noch einmal binäre Suche nach einem Element x in einem Array $M[1, \ldots, n-1]$ mit $n = 2^k$ (vgl. Beispiel 2.7).

Günstigster Fall: *Steht das Element an mittlerer Position, endet die Suche nach einem Schritt. Die* best-case *Laufzeit ist also in* $\Theta(1)$.

Ungünstigster Fall: *Wie in Beispiel 2.7 festgestellt, endet die Suche nach höchstens k Schritten, und dieser Fall kann sowohl dadurch eintreten, dass $x \notin M$, als auch durch eine ungünstige Position (z.B. $x = M[1]$). Die* worst-case *Laufzeit ist damit in* $\Theta(k) = \Theta(\log n)$.

Mittlerer Fall: *Wie in Beispiel 2.23 gezeigt ist die erwartete Anzahl der Schritte um höchstens eins kleiner als die maximale. Die* average-case *Laufzeit ist damit ebenfalls in* $\Theta(k-1) = \Theta(\log n)$.

Im allgemeinen ist die Laufzeit also irgendwo in $\Omega(1) \cap \mathcal{O}(n \log n)$, *meistens jedoch nahe der oberen Schranke.*