2. Aufwandsklassen und Aufwandsabschätzung

Reflektieren wir noch einmal was wir bisher gelernt haben: Wir haben gegebene Anforderungen versucht in Algorithmen umzusetzen. Ferner haben wir diese in Pseudocode und in C beschrieben. Nun kann es verschiedene Algorithmen geben, die die gegegebene Anforderung umsetzen. Interessant ist nun, wie wir solche Algorithmen vergleichen können. Wir benötigen also eine Vergleichsgröße. Solch eine Größe wäre zum Beispiel die Laufzeit eines Algorithmus'. Im folgenden beschäftigen wir uns also mit der Untersuchung der Laufzeitkomplexität eines Algorithmus'.

Wir müssen hierbei folgende Punkte beachten:

- Wir betrachten die Laufzeit eines Algorithmus' in Abhängigkeit der Eingabedaten. (Bekommt unser Algorithmus beispielsweise ein Array von Zahlen der Länge n als Eingabe und gibt dieses Array beispielsweise wieder sortiert zurück, dann müssen wir die Laufzeit dieses Algorithmus' in Abhängigkeit der Arraylänge betrachten.)
  Warum ist das wichtig? Wir wollen untersuchen, wie sich der Algorithmus bei wachsender Anzahl der Eingabewerte verhält. Mit Hilfe dieser Abhängigkeitsbetrachtung können wir die Laufzeit des Algorithmus' mathematisch durch eine Funktion beschreiben (*Laufzeitfunktion*).
- Wir abstrahieren von Rechnermodellen. Warum? Anweisungen wie if, Wertzuweisungen oder ähnliches laufen auf einem schnellen Super-PC natürlich schneller, als auf einem herkömmlichen Laptop. Da uns aber die Laufzeit bedingt durch den Algorithmus und nicht durch eine bestimmte Architektur interessiert, wollen wir davon abstrahieren und wollen nur wissen, wie oft diese 'Basisanweisungen' aufgerufen werden. Also wollen wir uns von der Hardware loslösen und die Thematik Laufzeit in eine mathematische Betrachtung überführen.
- Aus dem ersten Punkt wissen wir nun, dass wir die Laufzeit eines Algorithmus' als mathematische Funktion beschreiben können (eben in Abhängigkeit der Eingabegröße). Demnach können wir Algorithmen nun dahingehend vergleichen, in dem wir die entsprechenden Funktionen vergleichen. Theoretisch könnten wir die eigentliche Steigung der Funktionen betrachten, also quantitative Aussagen zum Anstieg machen und diese miteinander vergleichen. Uns interessiert aber lediglich das *Wachstumsverhalten* der Funktionen, also den *Charakter* der Funktionen (z.B. ob sie konstant, linear, quadratisch, kubisch, ... wachsen). Also ordnen wir die Funktionen in solche Wachstumsklassen ein.

Betrachten wir nun folgende Definition:

Definition der Aufwandsklassen mittels der Landau-Symbole ($\mathcal{O}, \Omega, \Theta$):

$$f \in \mathcal{O}(g) \Leftrightarrow \exists c > 0, n_0 \in \mathbb{N} : \ \forall n \geq n_0 : \ f(n) \leq c \cdot g(n)$$

$$f \in \Omega(g) \Leftrightarrow \exists c > 0, n_0 \in \mathbb{N} : \ \forall n \geq n_0 : \ f(n) \geq c \cdot g(n)$$

$$f \in \Theta(g) \Leftrightarrow \exists c_1, c_2 > 0, n_0 \in \mathbb{N} : \ \forall n \geq n_0 : \ c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

Betrachten wir zunächst die erste Zeile der Definition.

$$f \in \mathcal{O}(g) \Leftrightarrow \exists c > 0, n_0 \in \mathbb{N} : \forall n \geq n_0 : f(n) \leq c \cdot g(n)$$

Was bedeutet das? Zunächst halten wir fest, dass $\mathcal{O}(g)$ eine Menge von Funktionen ist. Also bedeutet $f \in \mathcal{O}(g)$, dass die Funktion f in der Menge $\mathcal{O}(g)$ enthalten ist. Der Ausdruck rechts vom '$\Leftrightarrow$' liest sich 'Es existiert ein c größer 0 aus der Menge der Natürlichen Zahlen und ein n_0 aus der Menge der Natürlichen Zahlen, so dass für alle n, die größer sind als das n_0, gilt, dass $f(n)$ größer oder gleich $c \cdot g(n)$ ist.' Wir übersetzen '$\Leftrightarrow$' zu 'genau dann, wenn'. Folglich ist f genau dann in der Menge $\mathcal{O}(g)$ enthalten, wenn die rechte Seite gilt. Das bedeutet: Gilt die rechte Seite, dann ist f in der Menge $\mathcal{O}(g)$ enthalten. Ist f in $\mathcal{O}(g)$ enthalten, dann gilt die rechte Seite. (Biimplikation)
Wenn $f \in \mathcal{O}$ ist, ist g für f eine Obere Schranke. Das heißt in Anbetracht der Laufzeit, dass der Algorithmus, dessen Laufzeit mit f beschrieben ist, ab einer bestimmten Eingabegröße n_0 garantiert **nie** mehr Zeit benötigt, als seine Obere Schranke. Abbildung 1 illustriert diesen Sachverhalt.

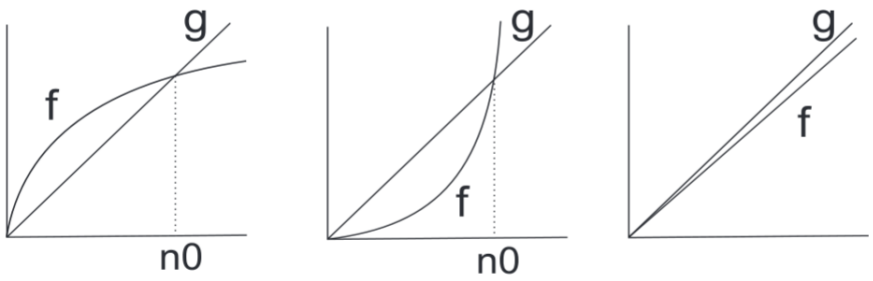

Abbildung 1: (v.l.n.r.) $f \in \mathcal{O}(g), f \in \Omega(g), f \in \Theta(g)$

Demnach bedeutet $f \in \Omega(g)$, dass g eine untere Schranke von f ist, die ab einem bestimmten n_0 nicht mehr unterschritten wird.
$f \in \Theta(g)$ bedeutet, dass f genauso wie g wächst, abgesehen von den konstanten Faktoren, die f und g beeinflussen.

In dem Sinne wollen wir noch zwischen 3 Szenarien unterscheiden:

- Worst-Case: Maximaler Aufwand eines Algorithmus' mit Laufzeitfunktion f (für den Worst-Case suchen wir also eine Obere Schranke g, so dass $f \in \mathcal{O}(g)$)
- Best-Case: Minimaler Aufwand eines Algorithmus' mit Laufzeitfunktion f (für den Best-Case suchen wir also eine Untere Schranke g, so dass $f \in \Omega(g)$)
- Average-Case: Durchschnittlicher Aufwand eines Algorithmus mit Laufzeitfunktion f bei einer durchschnittlichen Verteilung der Eingabe (für den Average-Case suchen wir also ein g, so dass $f \in \Theta(g)$)

Typische Wachstumsklassen:

$\mathcal{O}(1)$ (konstant)	$\mathcal{O}(log\ n)$ (logarithmisch)	$\mathcal{O}(n)$ (linear)	$\mathcal{O}(n \cdot log\ n)$
$\mathcal{O}(n^2)$ (quadratisch)	$\mathcal{O}(n^3)$ (kubisch)	$\mathcal{O}(c^n)$ (exponentiell)	