Landau-Symbole

Die fünf sogenannten *Landau-Symbole* sollten Ihnen bereits bekannt sein:

$O(f)$ ist eine Klasse von Funktionen von $\mathbb{N}$ nach $\mathbb{N}$, zu der die Funktion g gehört, wenn Konstanten N_0 und $c > 0$ existieren mit:

$$g(n) \leq c \cdot f(n) \text{ für alle } n \geq N_0$$

Wird in der Ungleichung das $\leq$-Zeichen durch $\geq$ ersetzt, d.h.

$$g(n) \geq c \cdot f(n) \text{ für alle } n \geq N_0$$

dann ist g in der Klasse $\Omega(f)$.

Die Funktionenklasse $\Theta(f)$ ist $O(f) \cap \Omega(f)$.

$o(f)$ enthält die Funktion g, falls für alle $c > 0$ ein N_0 existiert, sodass für alle $n \geq N_0$ die Ungleichung $g(n) \leq c \cdot f(n)$ gilt.

Für $\omega(f)$ muss analog $f(n) \leq c \cdot g(n)$ erfüllt sein.

Rekursion und Rekursionsgleichungen

Rekursion spielt in der Algorithmentheorie eine große Rolle!

Gegeben: Ein Problem, das von einem Parameter abhängt.
Löse dieses Problem für Parameterwert n unter
Zuhilfenahme der Lösung für Parameterwert m
(üblicherweise $m < n$).

(Manchmal auch $m_1, \ldots, m_k$, wobei alle $m_i \neq n$)

Beispiele:

`sort`$(a_1, \ldots, a_n)$: `IF` $n > 1$ `THEN` `sort`$(a_1, \ldots, a_{n-1})$;
füge a_n ein.

`nsort`$(a_1, \ldots, a_n)$: `FIND` j `WITH` $a_j \geq a_i$ `für alle` i;
`SWAP`(a_j, a_n); `nsort`$(a_1, \ldots, a_{n-1})$.

Realistischere Beispiele:

Fibonacci-Zahlen, Euklidscher Algorithmus,...

Divide & Conquer

Divide and Conquer = Divide et impera = Teile und herrsche

Dieses Konzept kann man als Spezialfall des Rekursions-Schemas betrachten. Es läuft üblicherweise so ab:

(Die zu bearbeitenden Daten seien in einem Array $a_1, \ldots, a_n$ gegeben)

---

1) Teile $a_1, \ldots, a_n$ auf in $b_1, \ldots, b_m$ und $c_1, \ldots, c_{n-m}$
2) Führe rekursiv die geforderte Berechnung sukzessive auf beiden Teil-Arrays aus. Die Ergebnisse seien LSG_b und LSG_c.
3) Füge die Lösungen LSG_b und LSG_c zur Lösung LSG_a des ursprünglichen Problems zusammen.

(Beachte: Die Schritte 1) und 3) sollten möglichst in Linearzeit durchgeführt werden.)

Divide & Conquer (Forts.)

Für einen „guten“ Divide & Conquer Algorithmus ist es im Normalfall sehr wichtig, dass die beiden Teile, auf denen die rekursiven Aufrufe ausgeführt werden, etwa gleich groß sind, d.h. wir streben an, dass $m \approx \frac{n}{2}$ gilt.

Darüberhinaus ist es wichtig, dass die Schritte 1) und 3), die die rekursiven Aufrufe quasi „einrahmen“, nicht zu viel Zeit verbrauchen. Hier fordert man zweckmäßigerweise, dass diese Schritte jeweils mit linearer Laufzeit auskommen, d.h. die Gesamtlaufzeit eines Aufrufs soll (ohne die Rekursion aus Schritt 2) in $O(n)$ bzw. $\Theta(n)$ liegen.

Wir werden das am Beispiel von Sortierverfahren durchspielen.

Beispiel: Mergesort

Als erstes Beispiel betrachten wir ein Sortierverfahren, das man *Sortieren durch Mischen*, englisch *mergesort* nennt.

Eingabe ist ein Array $a_1, \ldots, a_n$ OBdA sei n gerade

AUFGABE: Sortiere $a_1, \ldots, a_n$ nach vorgegebenem Schlüssel!

```
mergesort(a_1, ..., a_n):
    IF n ≥ 2 THEN
        b_1, ..., b_{n/2} := mergesort(a_1, ..., a_{n/2});
        c_1, ..., c_{n/2} := mergesort(a_{n/2+1}, ..., a_n);
        a_1, ..., a_n := mische b_1, ..., b_{n/2} mit c_1, ..., c_{n/2};
    END
```

Bemerkungen

1) Der Aufteilungsschritt ist trivial – man muss nur den Index n halbieren und dann die Hälften kopieren, d.h. $O(n)$ reicht aus!
2) Die Aufteilung ist optimal, da $m = n/2$ gilt.
3) Das Mischen im letzten Schritt ist relativ einfach in Linearzeit realisierbar! (Details bitte selbst überlegen)

Was bedeutet das für die Gesamtlaufzeit?

Die sogenannte *Aufruftiefe* (auch *Rekursionstiefe*) ist $\log n$.

Je „Aufrufebene“ ist der Aufwand in $O(n)$.

Das ergibt eine Gesamtrechenzeit in $O(n \log n)$.

Nachteil von mergesort: Zusätzlicher Speicherbedarf!

mergesort ist kein „in situ“-Verfahren (in situ = in place)