Woche 2 – Übersicht & Tricks

(Disclaimer: Die hier vorzufindenden Notizen haben keinerlei Anspruch auf Korrektheit oder Vollständigkeit und sind nicht Teil des offiziellen Vorlesungsmaterials. Alle Angaben sind ohne Gewähr.)

O-Notation:

Def.: Sei $f: \mathbb{N} \to \mathbb{R}^+$ eine Kostenfunktion. Wir definieren nun eine Menge von Funktionen, die auch nicht schneller wachsen als f, konkret:

$$O(f) \coloneqq \{g: \mathbb{N} \to \mathbb{R}^+ \mid es\ gibt\ c > 0,\ n_0 \in \mathbb{N}\ so,\ dass\ g(n) < c * f(n)\ für\ alle\ n > n_0\}$$

- Aus der Definition folgt:
  - Die Funktion f ist asymptotisch eine obere Schranke für alle $g \in O(f)$
  - Beim Untersuchen des asymptotischen Verhaltens von Funktionen im Kontext der O-Notation können wir konstante Faktoren ignorieren
  - Abgeschlossenheit unter Addition:

$$g_1 \in O(f), g_2 \in O(f) \Rightarrow g_1 + g_2 \in O(f)$$

Beweis für Abgeschlossenheit unter Addition[1]:

Seien $g_1: \mathbb{N} \to \mathbb{R}$ *und* $g_2: \mathbb{N} \to \mathbb{R}$ *Funktionen mit* $g_1 \in O(f), g_2 \in O(f)$*. Dann gilt per Def. der O-Notation:* $\exists c_1 > 0, n_1 \in \mathbb{N} : g(n) < c_1 * f(n)\ \forall n > n_1$ *and*
$\exists c_2 > 0, n_2 \in \mathbb{N} : g_2(n) < c_2 * f(n)\ \forall n > n_2$
Es gilt: $g_1 + g_2 \in O(f) \iff \exists c_0 > 0, n_0 \in \mathbb{N} : g_1(n) + g_2(n) < c_0 * f(n)\ \ \forall n > n_0$
Nun zeigen wir, dass solche c_0 *und* n_0 *existieren: Wir wählen* $c_0 \coloneqq c_1 + c_2$ *und* $n_0 \coloneqq max\{n_1, n_2\}$*, und sehen:* $g_1(n) + g_2(n) \leq c_1 * f(n) + c_2 * f(n) = (c_1 + c_2) * f(n) = c_0 * f(n)\ \ \forall n > n_0$ ■

Analyse	O-Notation	Beschreibung
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$	$f \in O(g)$ und $g \notin O(f)$	„f wächst **langsamer*** als g“
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = C \in \mathbb{R}^+$	$f \in O(g)$ und $g \in O(f)$	„f und g wachsen **gleich schnell***“
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$	$f \notin O(g)$ und $g \in O(f)$	„f wächst **schneller*** als g“

Wichtige Komplexitätsklassen der Laufzeitanalyse:

Konstant: $O(1)$	Linear: $O(n)$	Polynomiell: $O(n^c), c \in \mathbb{R}^+$
Logarithmisch: $O(\log(n))$	Superlinear: $O(n * \log(n))$	Exponentiell: $O(c^n), c \in \mathbb{R}^+$

Tricks zum Umformen und Abschätzen von Summen:

1. $\sum_{i=1}^{n} i = \frac{n*(n+1)}{2}$
2. $\sum_{j=1}^{n}\sum_{k=j}^{n} 1 = \sum_{j=1}^{n}(n-j+1) = \frac{n(n+1)}{2}$ (denn $\sum_{j=1}^{n}(n-j+1)$ ist letzten Endes nur die Summe aus 1., bloss "andersherum gezählt")
3. $\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} 1 = n^3$
4. $\sum_{i=1}^{n} i^3 = \frac{n^2*(n+1)^2}{4}$
5. $\sum_{i=1}^{n}\sum_{k=1}^{i} 1 = \sum_{i=1}^{n} i = \frac{n*(n+1)}{2}$
6. $\sum_{i=1}^{n} i*\log(i) \geq \sum_{i=\frac{n}{2}}^{n} i*\log(i) \geq \sum_{i=\frac{n}{2}}^{n} \frac{n}{2}*\log\left(\frac{n}{2}\right) \geq \frac{n^2}{4}*\log\left(\frac{n}{2}\right)$
7. $\sum_{i=1}^{n} i*\log(i) \leq \sum_{i=1}^{n} n*\log(n) = n^2*\log(n)$

Maximum Subarray-Sum Algorithmus ("Kadane's Algorithm"):

In der Vorlesung wurden verschiedene Lösungsansätze für das Maximum Subarray-Sum Problem vorgestellt. Hier werde ich nun den schnellsten, induktiven Algorithmus noch einmal mit einem Beispiel präsentieren:

a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
0	1	2	3	4	5	6	7

Wir iterieren von links nach rechts über das Array A, d.h., wir bewegen uns in jeder Iteration um eine Position nach rechts. Dabei merken wir uns zwei Werte, MSS und RANDMAX. In MSS speichern wir die bislang berechnete Maximum Subarray-Sum (am Anfang initialisieren wir diesen Wert also mit 0) und in RANDMAX (anfangs auch 0) speichern wir die Subarray-Sum bis zu der Position im Array, an der wir uns gerade befinden. In jeder Iteration aktualisieren wir nun RANDMAX := RANDMAX + a_i (Also addieren wir in der i-ten Iteration das Element an Stelle i zu unserer Randsumme) und prüfen, ob diese neue Randsumme jetzt grösser als die bislang berechnete Maximum Subarray-Sum ist, sprich ob RANDMAX > MSS. Falls dem so ist, aktualisieren wir: MSS := RANDMAX, andernfalls lassen wir MSS unverändert. Abschliessend prüfen wir noch in jeder Iteration, ob unser neues RANDMAX < 0 (und setzen RANDMAX := 0, falls dem so ist), denn sollte die Subarray-Sum bis zur Position i (also bis an den Rand) negativ sein, müssen wir ab der nächsten (i+1-ten) Position wieder bei 0 beginnen. Denn bspw. ist -2 + 4 + … < 0 + 4 + … und die MSS kann grundsätzlich an jeder Position starten.

Hier nun ein Beispiel:

Wir wollen die Maximum Subarray-Sum für dieses Array berechnen:

4	-4	3	5	-1
0	1	2	3	4

Zunächst setzen wir RANDMAX := 0 und MSS := 0.

4	-4	3	5	-1
0	1	2	3	4

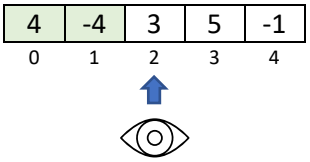
Jetzt beginnen wir, von links durch das Array zu iterieren, beobachten also zunächst Position **i = 0**: Wir setzen RANDMAX := RANDMAX + 4 (also gilt jetzt: RANDMAX == 4) und vergleichen: RANDMAX > MSS? Da MSS == 0, also RANDMAX > MSS setzen wir MSS := RANDMAX.

4	-4	3	5	-1
0	1	2	3	4

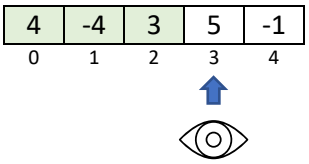
Nun sind wir bei Position **i = 1:** Für den grünen Teil kennen wir jetzt schon die MSS. Wir setzen RANDMAX := RANDMAX – 4 (also gilt jetzt: RANDMAX == 0). Da

RANDMAX < MSS, verändern wir MSS nicht. Da RANDMAX > 0, verändern wir RANDMAX nicht.

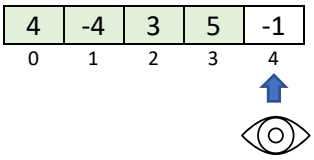
Nun sind wir bei Position **i = 2:** Für den grünen Teil kennen wir schon die MSS. Wir setzen RANDMAX := RANDMAX + 3 (also gilt jetzt: RANDMAX == 3). Da RANDMAX < MSS, verändern wir MSS nicht. Da RANDMAX > 0, verändern wir RANDMAX nicht.



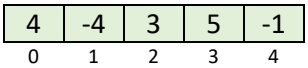
Nun sind wir bei Position **i = 3:** Für den grünen Teil kennen wir schon die MSS. Wir setzen RANDMAX := RANDMAX + 5 (also gilt jetzt: RANDMAX == 8). Da RANDMAX > MSS, setzen wir: MSS := RANDMAX. Da RANDMAX > 0, verändern wir RANDMAX nicht.



Nun sind wir bei Position **i = 4:** Für den grünen Teil kennen wir schon die MSS. Wir setzen RANDMAX := RANDMAX - 1 (also gilt jetzt: RANDMAX == 7). Da RANDMAX < MSS, verändern wir MSS nicht. Da RANDMAX > 0, verändern wir RANDMAX nicht.



Wir sind am Ende des Arrays angekommen und haben MSS == 8 als Maximum Subarray-Sum berechnet.



Hands-On: O-Notation:

(Das hier ist eine Aufgabe au seiner Altklausur, HS19 Algorithmen & Datenstrukturen):

Gilt: $\sum_{i=1}^{n} \log_2(i) \in O(n * \log(n))$?

Lösungsweg 1: Da die O-Notation asymptotisch obere Schranken für Funktionen definiert, versuchen wir, die Summe nach oben abzuschätzen:

$$\sum_{i=1}^{n} \log_2(i) \leq \sum_{i=1}^{n} \log_2(n) = n * \log_2(n) \in O(n * \log(n))$$

Lösungsweg 2: Es gilt: $\log(a * b) = \log(a) + \log(b)$. Dies verwenden wir nun:

$$\sum_{i=1}^{n} \log_2(i) = \log_2(\prod_{i=1}^{n} i) = \log_2(n!) \leq \log_2(n^n) = n * \log_2(n) \in O(n * \log(n))$$

Erklärung zu Lösungsweg 2: $\log(n!) \leq \log(n^n)$, den der Logarithmus ist eine streng monoton wachsende Funktion, also gilt: $\log(x) < \log(y) \Leftrightarrow x < y$ und $n! < n^n$, da offensichtlich:

$n! = 1 * 2 * \ldots * n$ (insgesamt n Faktoren) während $n^n = n * n * \ldots * n$ (insgesamt n Faktoren)