Woche 3 – 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.)

Anmerkungen zu Serie 1: Induktionsbeweise

Für einen korrekten Induktionsbeweis ist es enorm wichtig, dass die Struktur eingehalten und auf alle Formalitäten geachtet wird. Hier werde ich nun anhand eines Beispiels noch einmal genau zeigen, wie ein guter Beweis aussehen sollte:

1. Zunächst muss gezeigt werden, welche Aussage (für natürliche Zahlen) überhaupt bewiesen werden soll. Dabei ist es wichtig, darauf zu achten, dass korrekt angegeben wird, für welche natürlichen Zahlen die Aussage gelten soll (bspw. $\forall n \in \mathbb{N}, n > 4$)
   We will prove by mathematical induction that for all $n \in \mathbb{N}$: $\sum_{i=1}^{n}(2i-1) = n^2$
2. Nun muss gezeigt werden, dass die mathematische Aussage für das kleinste Element der definierten Menge gilt. Das ist der sogenannte *Induktionsanfang* oder *Base Case*. Dies muss unbedingt beim Beweis stehen.
   Base Case ($n = 1$): $\sum_{i=1}^{1}(2i-1) = 1 = 1^2$
3. Jetzt definieren wir die *Induktionsvorraussetzung* oder *Induktionshypothese*. Dafür nehmen wir an, dass die Aussage für **ein beliebiges** $m \in \mathbb{N}, m \geq n_0$ gilt. Oft habt ihr hier leider angenommen, dass die Aussage für "alle $m \in \mathbb{N}, m \geq n_0$" gilt. Das ist allerdings sehr falsch, da wir nicht annehmen können, was wir beweisen wollen.
   Induction Hypothesis (I.H.): Let $m \in \mathbb{N}$ be arbitrary. Assume that $\sum_{i=1}^{m}(2i-1) = m^2$
4. Abschliessend kommt nun der wichtigste Teil des Induktionsbeweises: Der *Induktionsschritt*. Wir haben in Schritt 3 angenommen, dass die Aussage für ein beliebiges $m \in \mathbb{N}, m \geq n_o$ gilt. Nun wollen wir zeigen, dass daraus folgt, dass die Aussage auch für $(m+1)$ gilt. Dabei müssen wir allerdings sehr genau vorgehen und **jeden einzelnen** Schritt einzeln angeben. Für eine gute Struktur sollte man die Umformungen untereinander schreiben. Bitte hier keine Gleichungen angeben, sondern von der LHS (Left-Hand-Side) zur RHS (Right-Hand-Side) umformen. Dabei werdet ihr an einer Stelle eure Induktionsvorraussetzung verwenden müssen. Wenn ihr das tut, ist es extrem wichtig, dass ihr das **genau angebt**, sonst ist der Proof formal nicht korrekt.
   Induction Step (I.S.) $(m \to m+1)$:

$$\sum_{i=1}^{m+1}(2i-1)$$

$$= \sum_{i=1}^{m}(2i-1) + (2(m+1)-1)$$

$$= m^2 + 2m + 1 \qquad \textit{(Induction Hypothesis)}$$

$$= (m+1)^2$$

QED.

O-Notation – Wie löse ich Summen auf?

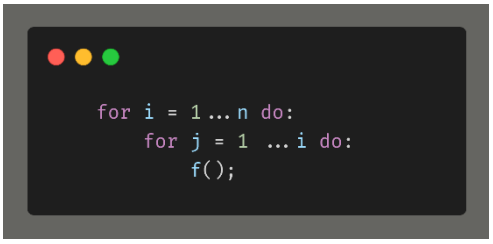
Zur Erinnerung die Tricks für Summen von Woche 2:

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. $\sum_{i=1}^{n} i*\log(i) \leq \sum_{i=1}^{n} n*\log(n) = n^2*\log(n)$

Es gibt häufig (bis jetzt an fast jeder Klausur) Aufgaben, bei denen ihr einen einfachen Algorithmus vorgegeben bekommt und dessen Laufzeit angeben sollt. Dabei müsst ihr üblicherweise O-Notation verwenden und eine möglichst enge Schranke für diese Laufzeit bestimmen. Das heisst: Läuft der Algorithmus tatsächlich in $O(n)$ und ihr schreibt, dass er in $O(2^n)$ läuft, habt ihr zwar teilweise recht – denn die Funktion zur Laufzeit eines Algorithmus mit linearer Laufzeit ist sicherlich auch nach oben durch eine exponentiell wachsende Funktion geschrieben und $O(2^n)$ damit eine obere Schranke – aber es ist ja nach einer möglichst engen Schranke gefragt. Also versucht, die Laufzeit des Algorithmus durch "genaues Hinsehen" (indem ihr alle vorkommenden Loops oder rekursiven Aufrufe korrekt zählt) oder – falls das nicht möglich ist – durch gutes Abschätzen nach oben und unten zu bestimmen.

Üblicherweise ist euch auch genau vorgegeben, wie genau ihr die Laufzeit bestimmen sollt. Normalerweise wird in dem vorgegebenen Algorithmus eine Funktion, z.B. `f();` (mehrfach) aufgerufen und ihr sollt möglichst genau (in O-Notation) angeben, wie häufig diese aufgerufen wird.

Dabei könnt ihr so vorgehen:

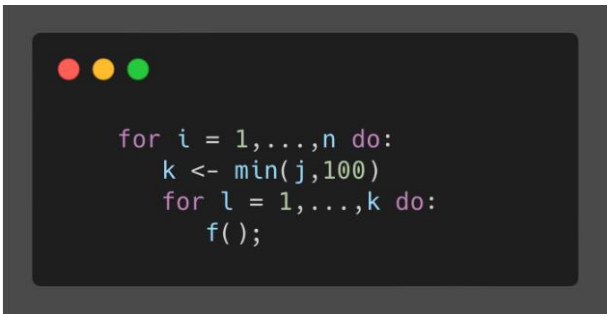


Gegeben sei folgender Algorithmus (in Pseudocode):

Wir sehen, dass es zwei for-Loops gibt, von denen der eine innerhalb des anderen liegt (nested for-Loop). Gucken wir uns zunächst nur den "inneren" Loop an: Dieser hat insgesamt i iterationen und in jeder Iteration wird `f()` genau einmal, also insgesamt im inneren Loop i -mal aufgerufen: $\sum_{j=1}^{i} 1 = i$. Dieser Loop liegt jedoch in einem "äusseren" Loop, der von $i = 1$ bis $i = n$ iteriert (also insgesamt n Iterationen). In jeder Iteration "startet" er den inneren Loop, den wir bereits analysiert haben. Mathematisch lässt sich das so ausdrücken: Die Funktion `f()` wird $T(n) = \sum_{i=1}^{n}(\sum_{j=1}^{i} 1)$ mal aufgerufen, wobei der grüne Teil den inneren Loop repräsentiert. Nun wissen wir (siehe Summen-Tricks oben): $T(n) = \sum_{i=1}^{n}(\sum_{j=1}^{i} 1) = \frac{n*(n+1)}{2} \leq n^2 \leq O(n^2)$, also haben wir eine ober Schranke. Nun zeigen wir noch, dass dies eine gute obere Schranke ist: $T(n) = \sum_{i=1}^{n}(\sum_{j=1}^{i} 1) = \sum_{i=1}^{n} i \geq \sum_{i=\lceil\frac{n}{2}\rceil}^{n} \frac{n}{2} \geq \frac{n^2}{4} \geq \Omega(n^2)$ (lower bound). Also gilt insgesamt: $T(n) = \Theta(n^2)$.

Weitere Aufgaben:

1) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für folgenden Algorithmus in Θ-Notation an:



```
for i = 1,...,n do:
    k <- min(j,100)
    for l = 1,...,k do:
        f();
```

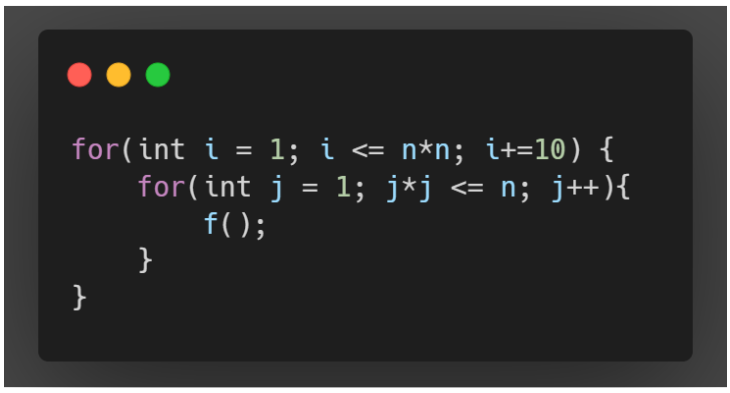
Lösung: $\Theta(n)$
Obere Schranke:

$$T(n) = \sum_{i=1}^{n} \sum_{l=1}^{\min(i,100)} 1 = \sum_{i=1}^{n} \min(i, 100) \leq \sum_{i=1}^{n} 100 = 100 * n \leq O(n)$$

Untere Schranke:

$$T(n) = \sum_{i=1}^{n} \min(i, 100) \geq \sum_{i=1}^{n} 1 = n \geq \Omega(n)$$

2) (FS13) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für folgenden Algorithmus in Θ-Notation an:

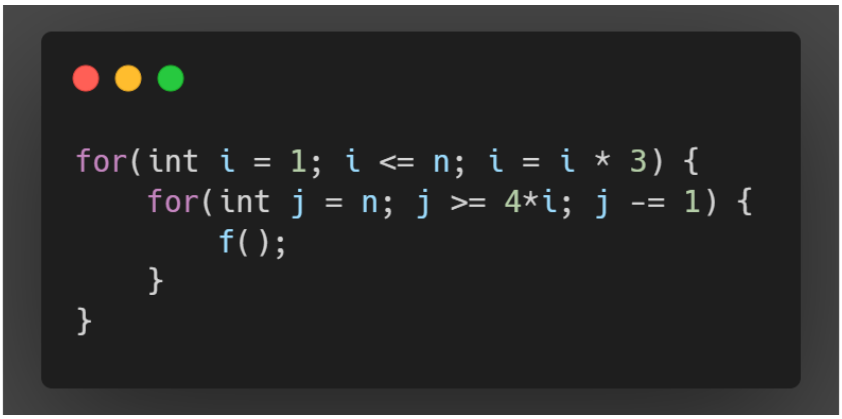


```
for(int i = 1; i <= n*n; i+=10) {
    for(int j = 1; j*j <= n; j++){
        f();
    }
}
```

Lösung: $\Theta(n^2\sqrt{n})$:

$$T(n) = \sum_{i=1}^{\frac{n*n}{10}} \sum_{j=1}^{\sqrt{n}} 1 = \sum_{i=1}^{\frac{n*n}{10}} \sqrt{n} = 0.1 \cdot n^2 \cdot \sqrt{n} = \Theta(n^2\sqrt{n})$$

3) (HS08) Geben Sie die asymptotische Laufzeit in Abhängigkeit von $n \in \mathbb{N}$ für folgenden Algorithmus in Θ -Notation an:



```
for(int i = 1; i <= n; i = i * 3) {
    for(int j = n; j >= 4*i; j -= 1) {
        f();
    }
}
```

Lösung: $\Theta(n \log n)$

$$T(n) = \sum_{i=1}^{\log_3 n} \sum_{j=1}^{n-4\cdot 3^i} 1 = \sum_{i=1}^{\log_3 n} \left(n - 4 \cdot 3^i\right) = \log_3 n \cdot n - 4 \cdot \sum_{i=1}^{\log_3 n} 3^i$$

$$= \log_3 n \cdot n - 4 \cdot \left(\frac{3^{\log_3 n + 1} - 1}{3 - 1}\right) = \log_3 n \cdot n - 2(3n - 1) = \Theta(n \log n)$$

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.

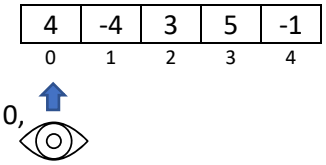
Hier nun ein Beispiel:

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

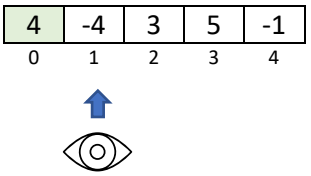
4	-4	3	5	-1
0	1	2	3	4

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

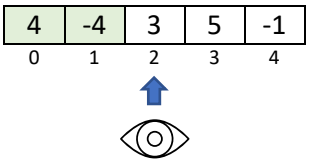
Jetzt beginnen wir, von links durch das Array zu iterieren, beobachten also zunächst Position **i = 0**: Wir setzen RANDMAX := RANDMAX + 4 (also gilt jetzt: RANDMAX == 4) und vergleichen: RANDMAX > MSS? Da MSS == 0, also RANDMAX > MSS setzen wir MSS := RANDMAX.



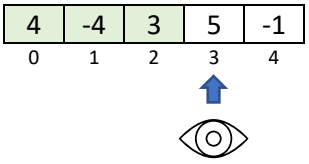
Nun sind wir bei Position **i = 1:** Für den grünen Teil kennen wir jetzt schon die MSS. Wir setzen RANDMAX := RANDMAX – 4 (also gilt jetzt: RANDMAX == 0). Da RANDMAX < MSS, verändern wir MSS nicht. Da RANDMAX ≥ 0, verändern wir RANDMAX nicht.



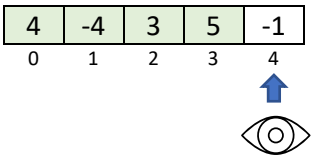
Nun sind wir bei Position **i = 2:** Für den grünen Teil kennen wir schon die MSS. Wir setzen RANDMAX := RANDMAX + 3 (also gilt jetzt: RANDMAX == 3). Da RANDMAX < MSS, verändern wir MSS nicht. Da RANDMAX > 0, verändern wir RANDMAX nicht.



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.