Lösungen der Präsenzaufgaben von Übungsblatt 1

Algorithmen und Datenstrukturen (WS 2013, Ulrike von Luxburg)

Ausnahmsweise, da im Allgemeinen *keine* Lösungen zu den Präsenzaufgaben bereitgestellt werden.

Lösungen zu Aufgabe 1

(a) $f \in \Theta(g)$, da beide linear, also $f, g \in \Theta(n)$

(b) $f \in o(g)$, da f quadratisch, g kubisch, bzw. formaler, weil $\lim_{n\to\infty} \frac{n^2}{n^3} = \lim_{n\to\infty} \frac{1}{n} \to 0$

(c) $f \in \Theta(g)$, da $\log(n^2) = 2\log(n)$

(d) $f \in o(g)$, da $\lim_{n\to\infty} \frac{n}{n \log n} = \lim_{n\to\infty} \frac{1}{\log n} \to 0$

(e) $f \in \omega(g)$, da $\lim_{n\to\infty} \frac{n^{1.01}}{n(\log n)^5} = \lim_{n\to\infty} \frac{n^{0.01}}{(\log n)^5} \to \infty$. Allgemein gilt für alle Konstanten $c, d > 0$, dass $\lim_{n\to\infty} \frac{n^c}{\log(n)^d} \to \infty$. Merke: Jede positive Potenz von n, selbst eine so kleine wie $n^{0.01} = \sqrt[100]{n}$ wächst asymptotisch schneller als jede Potenz eines Logarithmus, z.B., als $\log(n)^5$.

(f) $f \in \Theta(g)$, da $2^{n+1} = 2 \cdot 2^n \in \Theta(2^n)$

(g) $f \in \omega(g)$, da $\lim_{n\to\infty} \frac{n!}{2^n} = \lim_{n\to\infty} \frac{n}{2} \cdot \underbrace{\frac{(n-1)(n-2)\cdots 3 \cdot 2 \cdot 1}{2 \cdot 2 \cdots 2 \cdot 2 \cdot 2}}_{>1} > \lim_{n\to\infty} \frac{n}{2} \to \infty$

(h) $f \in o(g)$, da $g(n) = 2^{(\log_2(n))^2} = 2^{\log_2(n) \cdot \log_2(n)} = (2^{\log_2(n)})^{\log_2(n)} = n^{\log_2(n)}$ und weiterhin $\lim_{n\to\infty} \frac{(\log_2 n)^{\log_2 n}}{n^{\log_2 n}} = \lim_{n\to\infty} (\frac{\log_2 n}{n})^{\log_2 n} \to 0$, da offensichtlich $\frac{\log_2 n}{n}$ gegen 0 geht und somit erst recht dessen $\log_2 n$-te Potenz.

Lösungen zu Aufgabe 2

(a) $1 = (1 \cdot 1) = ((1 \cdot 1) \cdot (1 \cdot 1)) = ((1 \cdot 1) \cdot (1 \cdot 1)) \cdot ((1 \cdot 1) \cdot (1 \cdot 1)) = \ldots$, also immer 1

(b) Wir sehen, dass $C(n) = \underbrace{1}_{\text{Zeile 1}} + \underbrace{2 \cdot C(n-1)}_{\text{Zeile 5}}$, wollen aber eine geschlossene Formel dafür.

Um auf eine Vermutung zu gelangen, bestimmen wir $C(1) = 1$, $C(2) = 1 + 2 \cdot C(1) = 3$, $C(3) = 1 + 2 \cdot C(2) = 7$, usw., erhalten also die Folge: $1, 3, 7, 15, 31, 63, 127, \ldots$. Dies legt die Vermutung nahe, dass $C(n) = 2^n - 1$ gilt, was aber noch zu beweisen ist (per Induktion).

Induktionsanfang (IA) für $n = 1$: $C(1) = 1 = 2^1 - 1$, also von der behaupteten Form $C(n) = 2^n - 1$ für $n = 1$.

Induktionsvoraussetzung (IV): Sei nun also die Aussage $C(n) = 2^n - 1$ wahr für alle $1, \ldots, n$.

Müssen daraus nun die Induktionsbehauptung (IB) $C(n+1) \stackrel{!}{=} 2^{n+1} - 1$ folgern.

Erhalten für $C(n+1)$ aus der rekursiven Darstellung, dass

$$C(n+1) = 1 + 2 \cdot C(n) \stackrel{IV}{=} 1 + 2 \cdot (2^n - 1) = 2^{n+1} - 1,$$

was zu zeigen war. Merke: Im Induktionsschritt muss man das Problem für den *nächsten* Wert (hier $n+1$) irgendwie auf Probleme für *bekannte* Werte (hier $\leq n$) zurückführen, für welche man dann die IV einsetzen darf. Einige letzte Umformungen führen dann hoffentlich zur IB.

Wir können somit induktiv schließen, dass $C(n) = 2^n - 1$ wahr ist für alle $n \geq 1$.

Lösungen zu Aufgabe 3
Wir gehen in dieser Aufgabe davon aus, dass die Additionen und Zuweisungen jeweils Zeit $\Theta(1)$ benötigen. Wir brauchen daher nur die Anzahl der Durchläufe der (inneren) Schleife abzuschätzen. In der originalen Aufgabe hatte sich der Fehlerteufel eingeschlichen, durch welchen in der dritten Teilaufgabe die innere while-Schleife eine Endlosschleife ist. Wir gehen hier von der korrigierten Version (siehe Homepage) aus, nach welcher die Zuweisung $j \leftarrow j/2$ ebenfalls *innerhalb* der while-Schleife steht.

(a) Spielt man die Schleifen gedanklich durch, so sieht man, dass die Anzahl Aufrufe der 3. Zeile insgesamt $(n+1) + n + (n-1) + (n-2) + \ldots + 3 + 2 + 1$ beträgt, bzw. kompakter ausgedrückt $\sum_{k=1}^{n+1} k = \frac{(n+1)(n+2)}{2} \in \Theta(n^2)$. Merke: Wenn man bei derartig verschachtelten for-Schleifen die innere Schleife nicht jedesmal bis "zum Ende" durchlaufen lässt, sondern bereits am Schleifenzähler der äußeren Schleife abbricht, so halbiert man zwar die Laufzeit, hat aber asymptotisch immernoch quadratische Laufzeit.

(b) i wird in jedem Schleifendurchlauf hochgezählt bis $i^2 < n$ nicht mehr gilt. Somit nimmt i die Werte $1, 2, 3, \ldots, \lceil\sqrt{n}\rceil$ an, wobei für $i = \lceil\sqrt{n}\rceil$ die Schleife verlassen wird. Damit ist die Anzahl der Durchläufe $\Theta(\sqrt{n})$.

(c) Zu Beginn der inneren Schleife hat j jedesmal den Wert n und wird dann in jedem Schritt halbiert. Für $n = 64$ erhalten wir etwa $j = 64, 32, 16, 8, 4, 2, 1$, wobei die Schleife für $j = 1$ verlassen wird. Die Anzahl der Durchläufe der inneren Schleife ist damit $\log_2(n)$, wobei wir die Rundung auf eine Ganzzahl getrost ignorieren. Außerdem wird die äußere Schleife n Mal durchlaufen, so dass insgesamt Laufzeit $\Theta(n \log n)$ entsteht.

Lösungen zu Aufgabe 4

(a) $\Theta(n^2)$, wobei $length(A) = n$. Die Zeilen 4-6 werden zunächst $n-1$ Mal durchlaufen, dann $n-2$ Mal, usw., bis letztlich $3, 2, 1$ Mal. Ähnlich wie in Aufgabe 3.a) erhalten wir somit die Gesamtanzahl Durchläufe zu $\sum_{k=1}^{n-1} k = \frac{(n-1)n}{2} \in \Theta(n^2)$.

(b) Der Algorithmus sortiert die Zahlen des Arrays absteigend nach Größe. Er entspricht dem sogenannten Bubble-Sort, wobei das "$>$" in Zeile 4 gegen ein "$<$" ausgetauscht werden kann, um eine aufsteigende Reihenfolge zu erhalten.

(c) Um formal zu verifizieren, dass der Algorithmus wirklich das gewünschte liefert, kann man wie folgt vorgehen (nicht in der Vorlesung behandelt): Wir betrachten die *äußere* Schleife, und möchten Zusicherungen finden, die nach jedem Schleifendurchlauf wahr sind und letztlich das gewünschte Ergebnis (die absteigende Sortierung aller Elemente) beweisen. Solche Zusicherungen nennt man eine "Schleifeninvariante". Schaut man sich den Algorithmus genau an, so sieht man, dass im Schleifendurchlauf für jedes k zunächst das Minimum der Zahlen aus $A[1] \ldots A[k]$ nach hinten durchgereicht wird (die aktuelle "*bubble*") und letztlich an Stelle $A[k]$ stehen bleibt, bevor dann mit $k \mapsto k-1$ wieder am Anfang $A[1]$ genau so fortgefahren wird (diesmal wird eine Position weiter links gestoppt). Wenn man diese Einsicht etwas formaler ausdrückt, erhält man folgende zwei Zusicherungen:

(1) Am Ende vom Durchlauf mit Wert k sind die Elemente $A[k] \ldots A[n]$ absteigend sortiert

(2) Am Ende vom Durchlauf mit Wert k sind alle Elemente $A[1], \ldots, A[k-1] \geq A[k]$

Diese Einsichten kann man nun wie bei einer Induktion nutzen: Gestartet mit dem ersten Schleifendurchlauf $k = n$, sieht man schnell ein, dass (1) und (2) erfüllt sind, da 1. trivialerweise $A[n] \ldots A[n]$ absteigend sortiert sind, sowie 2. keines der Elemente $A[1] \ldots A[n-1]$ kleiner als $A[n]$ ist, da wir ja eben immer das Minimum nach rechts durchgereicht haben.

Diese Induktion "zählt" wie die äußere Schleife rückwärts und stoppt bei $k = 2$. Nun also der Schritt $k+1 \mapsto k$: Für jedes $2 \leq k < n$ erhalten wir, dass nach diesem Schleifendurchlauf (2) erfüllt ist, da erneut das Minimum von $A[1] \ldots A[k]$ zur Stelle $A[k]$ durchgereicht wurde, also somit kleiner gleich allen $A[1], \ldots, A[k-1]$ ist. Es bleibt zu zeigen, dass (1) gilt: Wir wissen von "vorher" (Zusicherung (2) für Wert $k+1$), dass $A[k]$ größer gleich allen $A[k+1], \ldots A[n]$ ist. Außerdem (Zusicherung (1) für $k+1$), dass die Elemente $A[k+1] \ldots A[n]$ absteigend

sortiert sind. Beides zusammen liefert, dass daher auch $A[k], A[k+1] \dots A[n]$ absteigend sortiert sind. Dies ist aber genau die zu zeigende Zusicherung (1) für k.

Am Ende dieser Induktion liefern die Zusicherungen für $k = 2$, dass 1. die Elemente $A[2] \dots A[n]$ absteigend sortiert sind, und 2. dass $A[1] \geq A[2]$. Insgesamt erhalten wir also aus dieser Schleifeninvariante, dass am Ende $A[1] \dots A[n]$ absteigend sortiert sind.

[btw: Diese Präsenzaufgabe "schaut ein bißchen über den Tellerrand". Nur tatsächliche Vorlesungsinhalte sind klausurrelevant.]