Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
P. Bamberger
P. Schneider

Algorithmen und Datenstrukturen
Sommersemester 2020
Musterlösung Übungsblatt 2

Abgabe: Mittwoch, 27.05.2020, 16:00 Uhr.

Aufgabe 1: $\mathcal{O}$-Notation *(7 Punkte)*

Beweisen oder widerlegen Sie die folgenden Aussagen anhand der *Mengendefinition* der $\mathcal{O}$-Notation (Vorlesungsfolien Woche 2, Folie 11 und 12).

(a) $3n^3 + 8n^2 + n \in \mathcal{O}(n^3)$ *(1 Punkt)*

(b) $2^n \in o(n!)$ *(2 Punkte)*

(c) $2\log n \in \Omega((\log n)^2)$ *(2 Punkte)*

(d) $\max\{f(n), g(n)\} \in \Theta(f(n) + g(n))$ für nicht negative Funktionen f und g. *(2 Punkte)*

Musterlösung

(a) Wahr. Sei $n_0 = 1$ und $c = 12$. Für $n \geq n_0$ gilt $n^3 \geq n^2 \geq n$ und daher $3n^3 + 8n^2 + n \leq 12n^3 = cn^3$.

(b) Wahr. Sei $c > 0$. Wähle $n_0 = \max\{\frac{1}{c}, 8\}$. Dann gilt für $n \geq n_0$

$$c \cdot n! \overset{n \geq 1/c}{\geq} \frac{1}{n} \cdot n! = (n-1)! \geq (n-1) \cdot (n-2) \cdot \ldots \cdot \lfloor \frac{n}{2} \rfloor \overset{n \geq 8}{\geq} 4^{\frac{n}{2}} = 2^n$$

(c) Falsch. Sei $c > 0$. Es gilt

$$\begin{array}{lrcl} & 2\log n & \geq & c(\log n)^2 \\ \Leftrightarrow & 2 & \geq & c\log n \\ \Leftrightarrow & \frac{2}{c} & \geq & \log n \\ \Leftrightarrow & 4^{\frac{1}{c}} & \geq & n \end{array}$$

Für gegebenes $n_0 \geq 1$ wähle $n = \max\{n_0, 4^{\frac{1}{c}}\} + 1$. Für dieses n gilt $n > n_0$ und $2\log n < c(\log n)^2$.

(d) Wahr. Wähle $n_0 = 1$, $c_1 = \frac{1}{2}$ und $c_2 = 1$. Dann gilt für $n \geq n_0$

$$c_1 \cdot (f(n) + g(n)) \leq \max\{f(n), g(n)\} \overset{f,g \geq 0}{\leq} c_2(f(n) + g(n))$$

Aufgabe 2: Sortieren nach Asymptotischem Wachstum *(6 Punkte)*

Sortieren Sie folgende Funktionen nach asymptotischem Wachstum. Schreiben Sie $g <_{\mathcal{O}} f$ falls $g \in \mathcal{O}(f)$ *und* $f \notin \mathcal{O}(g)$. Schreiben Sie $g =_{\mathcal{O}} f$ falls $f \in \mathcal{O}(g)$ *und* $g \in \mathcal{O}(f)$ (kein Beweis nötig).

$\sqrt{n}$	2^n	$n!$	$\log(n^3)$
3^n	n^{100}	$\log(\sqrt{n})$	$(\log n)^2$
$\log n$	$10^{100}n$	$(n+1)!$	$n \log n$
$2^{(n^2)}$	n^n	$\sqrt{\log n}$	$(2^n)^2$

Musterlösung

$$\sqrt{\log n} <_{\mathcal{O}} \log(\sqrt{n}) =_{\mathcal{O}} \log n =_{\mathcal{O}} \log(n^3) <_{\mathcal{O}} (\log n)^2 <_{\mathcal{O}} \sqrt{n} <_{\mathcal{O}} 10^{100} n <_{\mathcal{O}} n \log n$$
$$<_{\mathcal{O}} n^{100} <_{\mathcal{O}} 2^n <_{\mathcal{O}} 3^n <_{\mathcal{O}} (2^n)^2 <_{\mathcal{O}} n! <_{\mathcal{O}} (n+1)! <_{\mathcal{O}} n^n <_{\mathcal{O}} 2^{(n^2)}$$

Aufgabe 3: Stabiles Sortieren *(7 Punkte)*

Ein Sortieralgorithmus heißt stabil, falls Elemente mit gleichem Schlüssel in der gleichen Reihenfolge bleiben. Man nehme z.B. an, man möchte folgende Strings sortieren wobei der Sortierschlüssel in diesem Beispiel der *Anfangsbuchstabe in alphabetischer Reihenfolge* ist:

$$[\text{“}tuv\text{”}, \text{“}adr\text{”}, \text{“}bbc\text{”}, \text{“}tag\text{”}, \text{“}taa\text{”}, \text{“}abc\text{”}, \text{“}sru\text{”}, \text{“}bcb\text{”}]$$

Ein *stabiler* Sortieralgorithmus muss dann folgenden Output generieren:

$$[\text{“}adr\text{”}, \text{“}abc\text{”}, \text{“}bbc\text{”}, \text{“}bcb\text{”}, \text{“}sru\text{”}, \text{“}tuv\text{”}, \text{“}tag\text{”}, \text{“}taa\text{”}]$$

Ein Sortieralgorithmus wäre nicht stabil (bzgl. diesem Sortierschlüssel), wenn er z.B. Folgendes ausgibt:

$$[\text{“}abc\text{”}, \text{“}adr\text{”}, \text{“}bbc\text{”}, \text{“}bcb\text{”}, \text{“}sru\text{”}, \text{“}taa\text{”}, \text{“}tag\text{”}, \text{“}tuv\text{”}]$$

(a) Welche der in der Vorlesung gezeigten Sortieralgorithmen (ausser `CountingSort`) sind in der gegebenen Form *nicht* stabil? Beweisen Sie jeweils anhand eines geeigneten Beispiels. *(4 Punkte)*

(b) Beschreiben Sie eine Methode, welche jeden Sortieralgorithmus stabil macht ohne die *asymptotische* Laufzeit zu ändern. Begründen Sie. *(3 Punkte)*

Musterlösung

(a)
- Selection Sort ist nicht stabil. Betrachte als Eingabe das Array $[x, y, z]$ mit $x.key = y.key = 1$ und $z.key = 0$. Im ersten Schritt werden x und z vertauscht, da z den kleinsten Schlüssel im Array hat, d.h. man erhält $[z, y, x]$. Dieses Array bleibt danach unverändert (da $y.key = x.key$), entspricht also der Ausgabe von Selection Sort. Die Elemente x und y wurden also vertauscht.
- Quicksort ist nicht stabil. Betrachte als Eingabe das Array $[x, y, z, w]$ mit $x.key = 1$, $y.key = z.key = 2$ und $w.key = 0$ und nehme an x wird als Pivot gewählt. Im ersten divide-Schritt werden y und w vertauscht (d.h. man erhält $[x, w, z, y]$) und teilt das Array in $[x, w]$ und $[z, y]$. Rekursive Sortierung auf den Teilarrays liefert $[w, x]$ und $[z, y]$, d.h. die Rückgabe ist $[w, x, z, y]$. Die Elemente y und z wurden also vertauscht.
- Mergesort: Implementiert man Mergesort entsprechend dem Pseudocode auf S. 26 von Vorlesung 01, ist der Algorithmus nicht stabil. Entscheidend ist hier in Zeile 7 des Codes die Bedingung $A[i] < A[j]$, was dazu führen kann, dass Elemente mit gleichem Schlüssel in umgekehrter Reihenfolge einsortiert werden. In der Vorlage `MergeSort.py` vom 1. Übungsblatt haben wir an dieser Stelle hingegen die Bedingung $A[i] \leq A[j]$, wodurch der Algorihmus stabil ist.

(b) Man fügt dem i-ten Element im Array als zusätzlichen Sortierschlüssel die Zahl i hinzu (d.h. man setzt $A[i].key = (A[i].key, i)$). Nun wendet man den gegebenen (nicht-stabilen) Sortieralgorithmus entsprechend der lexikographischen Ordnung[1] auf diesen neuen Schlüsseln an. D.h. man sortiert in erster Priorität nach den ursprünglichen Schlüsseln und in zweiter Priorität nach dem (ursprünglichen) Index in A.

[1] Seien $(A, <_A)$ und $(B, <_B)$ zwei geordnete Mengen. Dann ist die lexikographische Ordnung $<_{lex}$ auf $A \times B$ definiert durch $(a, b) <_{lex} (a', b') :\Leftrightarrow a <_A a' \vee (a = a' \wedge b <_B b')$

Die Änderung der Sortierschlüssel benötigt $O(n)$. Zudem verlängert sich jeder Vergleich zwischen zwei Elementen um $O(1)$. Da ein Sortieralgorithmus $\Omega(n)$ benötigt, verlängert sich die asymptotische Laufzeit nicht.