Lehrstuhl Algorithmen
und Komplexität
Prof. Dr. Fabian Kuhn
Sebastian Daum, Oleksii Saukh

Algorithmen und Datenstrukturen
(Informatik II) SS 2014

http://ac.informatik.uni-freiburg.de/teaching

Übungsblatt 2

Abgabe: Dienstag, 13. Mai, 16:00 Uhr

Aufgabe 1 (6 Punkte)

Seien f und g zwei nicht-negative Funktionen auf $\mathbb{N}$. Beweisen Sie die folgenden Aussagen:

$$\begin{aligned} O(f(n)) + O(g(n)) &= O(\max\{f(n), g(n)\}) \\ \Omega(f(n)) + \Omega(g(n)) &\subseteq \Omega(f(n)) \\ O(f(n)) \cdot O(g(n)) &= O(f(n) \cdot g(n)) \end{aligned}$$

Hinweis: Betrachten Sie $O(f(n))$ (bzw. $O(g(n))$, etc.) als *Menge* von *Funktionen* mit bestimmten Eigenschaften und weisen Sie nach, dass die LHS ("left hand side", die linke Seite der Gleichung) eine *Teilmenge* der RHS ("right hand side") ist *und im Fall "=" auch umgekehrt*. Betrachten Sie die Operation "+" zwischen zwei Mengen A und B wie folgt: $A + B = \{x + y | x \in A \wedge y \in B\}$. Analoges gilt für "$\cdot$".

Bemerkung: Es gibt noch jede Menge mehr solcher Aussagen. Z.B. lassen sich in der ersten und dritten Aussage alle O's durch Ω, ω, o oder Θ ersetzen und sie stimmen dann immer noch.

Aufgabe 2 (6 Punkte)

Gegeben sind die folgenden Funktionen (der Logarithmus ist immer zur Basis 2):

$$\begin{array}{ccc} \binom{n}{4}, & \log n, & \log(n!), \\ n \log n, & 2^{\log n^4}, & 4^n, \\ 10^{1000} n + \frac{1}{n}, & n!, & \sqrt{n}, \\ 2^n, & n^{4+\frac{1}{n}}, & \pi \end{array}$$

Ordnen Sie die Funktionen nach ihrem asymptotischem Wachstum, d.h., geben Sie eine Liste $f_1, f_2, \ldots$ wieder, so dass $f_1 = O(f_2)$, $f_2 = O(f_3)$, etc. Markieren Sie Funktionen mit gleichem asymptotischen Wachstum, d.h., Funktionen f und f', für die gilt, dass $f \in \Theta(f')$.

Sie müssen Ihre Lösungen weder beweisen noch begründen, aber achten Sie sehr auf die Korrektheit Ihrer Anordnung.

Hinweis: Schlagen Sie bei Bedarf Regeln für den Umgang mit Logarithmen nach. Es gilt außerdem, dass $n^{n/2} \leq n! \leq n^n$.

Aufgabe 3 (8 Punkte)

Nehmen Sie an, wir haben eine Methode, um für den *QuickSort* Algorithmus in jedem Rekursionsaufruf ein Pivot-Element p zu bestimmen, so dass beim Vergleichen der Array-Elemente mit p, je mindestens $1/4$ der Werte kleiner als p und mindestens $1/4$ der Werte größer als p ist. In anderen Worten, p spaltet ein zu sortierendes Array mit m Elementen in ein Array mit λm und ein Array mit $(1-\lambda)m$ Elementen auf, wobei $\lambda \in [\frac{1}{4}, \frac{1}{2}]$.

Nehmen Sie auch an, dass es eine Konstante $b > 0$ gibt, so dass das Spalten eines Arrays der Größe m, das Auffinden des Pivot-Elements p für dieses Array und das spätere Zusammenführen der zwei sortierten Sub-Arrays, zusammen maximal bm Zeitschritte benötigt.

Beweisen Sie unter diesen Annahmen, dass für die Laufzeit $T(n)$ von *QuickSort* gilt: $T(n) \leq cn \log n$ für ein konstantes c.

Hinweis: Die Aussage lässt sich am einfachsten per Induktion beweisen. Nutzen Sie als Induktionsanfang allerdings lieber $n = 2$. Starten Sie damit, eine Ungleichung für T aufzusetzen und beachten Sie im Allgemeinen, dass λ keine Konstante ist, sondern sich bei jedem Funktionsaufruf ändern kann.