Übungen zur Vorlesung Datenstrukturen und Algorithmen

T1

Halten Sie für jede Kombination der folgenden zehn Funktionen fest, wie sie sich im Sinne der O-Notation zueinander verhalten. Tragen Sie hierzu das Symbol O, Θ bzw. Ω in eine Tabellenzelle ein, wenn $f = O(g)$, $f = \Theta(g)$ bzw. $f = \Omega(g)$ gilt.

$f \downarrow \backslash g \rightarrow$	$n/2$	$\sqrt{n+1}$	7	0.9^n	n^3+n	2^n	$\sqrt{n}$	$n!$	$500n^2$	1.1^n
$n/2$										
$\sqrt{n+1}$										
7										
0.9^n										
n^3+n										
2^n										
$\sqrt{n}$										
$n!$										
$500n^2$										
1.1^n										

Lösungsvorschlag:

Diese Aufgabe läßt sich hervorragend interaktiv lösen, indem die Teilnehmer der Übungsgruppe reihum die Zellen auffüllen. Man kann sich diese Arbeit sehr erleichtern, wenn man Identitäten und Symmetrien beachtet.

Am einfachsten ist es natürlich, die Funktionen vorher zu sortieren; das sollte aber vermieden werden, weil es den Übungseffekt drastisch verringert. Zur Kontrolle:
$0.9^n = O(7)$, $7 = O(\sqrt{n})$, $\sqrt{n} = \Theta(\sqrt{n+1})$, $\sqrt{n+1} = O(n/2)$, $n/2 = O(500n^2)$, $500n^2 = O(n^3+n)$, $n^3+n = O(1.1^n)$, $1.1^n = O(2^n)$, $2^n = O(n!)$

T2

Die Analyse eines zweiphasigen Algorithmus hat ergeben, daß die Laufzeit der ersten Phase $\Theta(n^{4\log n-1})$ beträgt, während die zweite Phase genau $2^{(\log(n^2+1))^2}$ Instruktionen umfasst. Schätzen Sie möglichst genau ab, wie viele Instruktionen der Algorithmus insgesamt benötigt.

Tip: Denken Sie bei Tag und Nacht immer ganz feste an Logarithmusgesetze, binomische Formeln, Potenzgesetze, Taylor-Entwicklungen und einen gewissen Satz aus der Vorlesung.

Lösungsvorschlag:

Zunächst formen wir die zweite Schranke um. Die Summe im Logarithmus gefällt uns nicht, weil es für diesen Fall kein Logarithmusgesetz gibt. Deshalb verwandeln wir die Summe in ein Produkt und erhalten

$$2^{(\log(n^2+1))^2} = 2^{(\log(n^2(1+1/n^2)))^2} = 2^{(2\log n + O(n^{-2}))^2}.$$

Hierbei wurden die Gleichheiten $\log(1+f(n)) = \ln(1+f(n))/\ln 2$ und – laut Taylor – $\ln(1+f(n)) = O(f(n))$ für $\lim_{n\to\infty} f(n) = 0$ ausgenutzt.
Zur Auflösung des Quadrats verwenden wir natürlich die geeignete binomische Formel. Hierbei verschwindet die dritte Komponente von $a^2 + 2ab + b^2$ dank O-Notation in der zweiten, und es gilt

$$2^{(2\log n + O(n^{-2}))^2} = 2^{4(\log n)^2 + O(n^{-2}\log n)}.$$

Nun ziehen wir den Faktor $\log n$ aus dem Exponenten in die Basis. Mit der Gleichheit $2^{\log n} = n$ und den Potenzgesetzen folgt

$$2^{4(\log n)^2 + O(n^{-2}\log n)} = n^{4\log n} \cdot 2^{O(n^{-2}\log n)} = n^{4\log n} \cdot e^{O(n^{-2}\log n)}.$$

Der letzte Schritt gestattet uns den Einsatz der Taylor-Entwicklung von e^z, die $e^{f(n)} = 1 + O(f(n))$ für $\lim_{n\to\infty} f(n) = 0$ impliziert. Damit gilt

$$n^{4\log n} \cdot e^{O(n^{-2}\log n)} = n^{4\log n} \cdot (1 + O(n^{-2}\log n)) = n^{4\log n} + O(n^{4\log n - 2}\log n).$$

Natürlich gilt

$$O(n^{4\log n-2}\log n) = O(n^{4\log n-1}).$$

Die Gesamtzahl der Instruktionen lautet also

$$n^{4\log n} + O(n^{4\log n-1}).$$

H4 (15 Punkte)

Implementieren Sie eine Unterklasse StatList<K,D> von List<K,D>, die kumulativ mitzählt, wie oft Knoten während einer Suche besucht werden.
Implementieren Sie eine Unterklasse MTFList<K,D> von StatList<K,D>, die das bei einer erfolgreichen Suche gefundene Element an den Anfang der Liste verschiebt („*move to front*“).
Implementieren Sie ein Testprogramm, das mit Hilfe einer StatList<String, Integer> zählt, welche Wörter in *War of the Worlds* von H. G. Wells wie oft vorkommen. Benutzen Sie als Quelle die Datei `http://www.gutenberg.org/files/36/36.txt`. Wieviele Listenknoten werden beim Suchen besucht, wenn Sie eine *MTFList* und wenn Sie eine *StatList* verwenden?
Erläutern Sie, warum die Laufzeit des Programms proportional zu dieser Zahl ist.
Hinweis: Als Wörter verstehen wir maximale zusammenhängende Zeichenketten aus den Buchstaben A bis Z, wobei wir zwischen kleinen und großen Buchstaben nicht unterscheiden. Alle anderen Zeichen gelten als Sonderzeichen und müssen ignoriert werden.
Beispiel: Zu der Eingabe „Der Hund, der der Katze 17 nicht traute.“ sollte Ihr Programm am Ende die Liste

(traute, 1), (nicht, 1), (katze, 1), (der, 3), (hund, 1)

ausgeben.
Auf der Übungsseite stellen wir Ihnen eine Datei zur Verfügung, die die Wörter des Textes bereits in Kleinschreibung mit einem Wort pro Zeile enthält.
Lösungsvorschlag:

Auf dem Webserver finden Sie als Beispiellösung die Dateien „StatList.java“, „MTFList.java“ und „H4.java“.

H5 (10 Punkte)
Wir betrachten zwei Algorithmen für das Durchsuchen einer doppelt verketteten Liste: Algorithmus A sucht linear vom Anfang der Liste bis zum Ende. Algorithmus B dagegen durchsucht die Liste rückwärts und beginnt beim letzten Knoten.
Wieviele Knoten besuchen die Algorithmen jeweils im schlechtesten Fall bei einer erfolgreichen Suche?
Algorithmus C ist ein randomisierter Algorithmus, der ebenfalls nach einem Element in einer doppelt verketteten Liste sucht. Er ruft mit Wahrscheinlichkeit $1/2$ Algorithmus A auf und mit Wahrscheinlichkeit $1/2$ Algorithmus B.
Die Laufzeit im schlechtesten Fall eines deterministischen Algorithmus ist einfach zu definieren: Über alle Eingaben der Länge n wählen wir die längste Laufzeit, die auftritt.
Für einen randomisierten Algorithmus definieren wir die erwartete Laufzeit im schlechtesten Fall so: Über alle Eingaben x der Länge n berechnen wir den Erwartungswert der Laufzeit für die Eingabe x und nehmen wieder die längste.
Was ergibt sich für die erwartete Anzahl besuchter Knoten im schlechtesten Fall für eine erfolgreiche Suche bei Algorithmus C?
Haben Sie einen Kommentar zum Verhältnis der drei Algorithmen zueinander?

Lösungsvorschlag: Algorithmus A und B benötigen im schlechtesten Fall n Vergleiche, wenn das gesuchte Element an letzter bzw. erster Stelle steht.
Für Algorithmus C müssen wir den Erwartungswert berechnen. Nehmen wir an, daß gesuchte Element steht an Postition k. Dann wird mit Wahrscheinlichkeit $1/2$ von vorn gesucht und k Vergleiche durchgeführt. Mit Wahrscheinlichkeit $1/2$ wird von hinten gesucht und es werden $n-k+1$ Vergleiche benötigt. Der Erwartungswert ist damit $(n+1)/2$ und das unabhängig von k. Also werden auch im schlechtesten Fall im Mittel genau $(n+1)/2$ Vergleiche durchgeführt.
Algorithmus C scheint besser als A und B zu sein, wenn ein bösartiger Benutzer es darauf anlegt, ihm schlechte Eingaben vorzusetzen. Ist die Eingabe zufällig, dann hat C keinen Vorteil. Wenn es auch keinen bösartigen Benutzer gibt, aber keinerlei Informationen über die zu erwartenden Eingaben existieren, mag das Modell der *worst-case average time* aber vernünftig sein.

Hinweise
Es wird eine Zentralübung und Fragestunde geben! Wir konnten den Hörsaal AH II im Informatikgebäude gewinnen und die Zentralübung findet dort montags von 14:00–15:30 statt.