Übung 4 Abgabe 21.5.2012

Aufgabe 1 – Asymptotische Komplexität 16 Punkte

a) Beweisen Sie die Vereinfachungsregeln für Ausdrücke in *O*-Notation:

$f(x)$	vereinfacht sich zu	$O\big(f(x)\big)$
$O\big(c\, f(x)\big)$	vereinfacht sich zu	$O\big(f(x)\big)$
$O\big(f(x)\big)\, O\big(g(x)\big)$	vereinfacht sich zu	$O\big(f(x)g(x)\big)$
$O\big(f(x)\big) + O\big(g(x)\big)$	vereinfacht sich zu	$O\big(f(x)\big)$ falls $g(x) \in O\big(f(x)\big)$

b) Ordnen Sie die folgenden Funktionen nach aufsteigender asymptotischer Komplexität:

$$f_A(x) = 3^x$$
$$f_B(x) = \sqrt{x} + \log_2 x$$
$$f_C(x) = x^x$$
$$f_D(x) = x \log_2 x$$
$$f_E(x) = 2^{\sqrt{\log_2 x}}$$
$$f_F(x) = x^3 + 12x^2 + 200x + 999$$

und begründen Sie Ihre Entscheidung (z.B. mit Hilfe der Regel von l'Hospital). Das heißt, bringen Sie die Funktionen in eine Reihenfolge $f_1 < f_2 < \cdots < f_5 < f_6$, und beweisen Sie, dass für alle $i < 6$ und geeignet gewählte Konstanten c_{i+1} gilt:

$$\lim_{x\to\infty} \frac{f_i(x)}{c_{i+1}\, f_{i+1}(x)} \le 1$$

Aufgabe 2 – Komplexität vs. Laufzeit 6 Punkte

Gegeben seien drei Algorithmen mit Laufzeit

$$a_1(N) = N^2 + N + 10$$
$$a_2(N) = 15\, N \log_2 N$$
$$a_3(N) = 2^N$$

(jeweils für *N* > 1). Geben Sie Intervalle für *N* an, wo jeder der drei Algorithmen am schnellsten abläuft. Bestimmen Sie außerdem die asymptotische Komplexität der Algorithmen und die daraus folgende Ordnung. In welchem Intervall stimmt diese Ordnung mit der Ordnung nach Laufzeit überein?

Aufgabe 3 – Komplexität und Laufzeit des Sieb des Eratosthenes 6 Punkte

Beweisen Sie, dass das Sieb des Eratosthenes (siehe Übung 1) in $O(N \log\log N)$ liegt (wobei *N* die größte betrachtete Zahl ist). Benutzen Sie dazu die Formel $\sum_{p\le N} \frac{1}{p} = O(\log\log N)$ (wobei die Summe sich über die Primzahlen *p* bis maximal *N* erstreckt) sowie die Schachtelungsregel der *O*-Notation. Bauen Sie in den Code aus Übung 1 (`"sieve.py"`) eine Zählervariable ein, die die Gesamtzahl der Iterationen der inneren Schleife zählt, und überprüfen Sie experimentell die Korrektheit des obigen Resultats.

Aufgabe 4 – Experimente zu Laufzeit und Optimierung 12 + 5 Punkte

a) Überprüfen Sie experimentell, ob das Python-Array (Typ `"list"`) ein dynamisches Array ist, d.h. ob die amortisierten Kosten von `array.append(x)` konstant sind. Benutzen Sie dafür das "timeit"-Modul (siehe Übung 2). *4 Punkte*

Bonusaufgabe: Wer die Programmiersprache C beherrscht, kann zusätzlich den Quellcode *5 Bonusp.*

von Python anschauen und die relevanten Teile der Implementation von `array.append(x)` kurz erläutern. Der Code befindet sich in der Python-source distribution in der Funktion "`PyList_Append()`" im File "`Objects/listobject.c`".

b) Eine naive Implementation der Multiplikation von zwei `size*size` Matrizen lautet[1] *8 Punkte*

```
for i in range(size):
    for j in range(size):
        for k in range(size):
            C[i+j*size] += A[i+k*size]*B[k+j*size]
```

Diese Implementation ist relativ langsam, weil die innere Schleife redundante Berechnungen enthält und der Cache schlecht ausgenutzt wird. Optimieren Sie die Implementation (z.B. durch Umstellung der Reihenfolge der Schleifen, Verschiebung von invarianten Teilausdrücken), begründen Sie Ihre Optimierungen und messen Sie die Verbesserung für `size`=100. (Verwenden Sie wieder das "timeit"-Modul. Die optimierte Version sollte ungefähr 40% schneller sein als die naive Implementation.) Ändert sich durch die Optimierung die Komplexität des Algorithmus?

[1] Matrizen sollen durch eindimensionale Arrays implementiert werden, so dass z.B. das Matrixelement C_{ij} als `C[i+j*size]` indexiert wird. Für die Zeitmessung nehmen wir an, dass alle C_{ij} bereits auf 0 initialisiert wurden.