Übungsblatt 1

Algorithmen und Datenstrukturen (WS 2013, Ulrike von Luxburg)

Präsenzaufgabe 1 (Landau-Notation) Bestimmen Sie für alle folgenden Beispiele, welcher der drei folgenden Fälle vorliegt: $f \in o(g)$, oder $f \in \omega(g)$, oder $f \in \Theta(g)$.

	$f(n)$	$g(n)$		$f(n)$	$g(n)$
(a)	$n/2$	$4n + 250$	(e)	$n^{1.01}$	$n \log(n)^5$
(b)	$10n^2 + 8n + 100$	n^3	(f)	2^n	2^{n+1}
(c)	$10 \cdot \log(n)$	$\log(n^2)$	(g)	$n!$	2^n
(d)	n	$n \log(n)$	(h)	$(\log_2 n)^{\log_2 n}$	$2^{(\log_2 n)^2}$

Präsenzaufgabe 2 (Induktion) Folgender rekursiver Algorithmus wird bei Eingabe $n \geq 1$ insgesamt $C(n)$ Mal aufgerufen (d.h. Zeile 1 ausgeführt).

```
1: function StupidAlg(n)
2:     if n = 1 then
3:         return 1
4:     else
5:         return StupidAlg(n − 1) · StupidAlg(n − 1)
6:     end if
7: end function
```

(a) Was berechnet der Algorithmus?

(b) Ermitteln Sie $C(n)$. Beweisen Sie durch vollständige Induktion.

Präsenzaufgabe 3 (Laufzeitanalyse I) Geben Sie scharfe asymptotische Schranken (d.h. von der Form $\Theta(\cdot)$) für die Laufzeit folgender Code-Fragmente in Abhängigkeit von n an. Gehen Sie davon aus, dass alle Zuweisungs- und Vergleichsoperationen konstante Zeit benötigen.

```
for i = 0 to n do
    for j = n downto i do
        sum ← sum + j
    end for
end for
```

```
i ← 1
while i · i < n do
    sum ← sum + 2
    i ← i + 1
end while
```

```
for i = 1 to n do
    j ← n
    while j > 1 do
        sum ← sum + i
        j ← j/2
    end while
end for
```

Präsenzaufgabe 4 (Laufzeitanalyse II) Sei A ein Array bestehend aus n reellen Zahlen $A[1], \ldots, A[n]$. Betrachten Sie folgenden Algorithmus, welcher A als Eingabe erhält.

```
1: function MagicAlgorithm(A)
2:     for k = length(A) downto 2 do
3:         for i = 2 to k do
4:             if A[i] > A[i − 1] then
5:                 swap A[i] and A[i − 1]
6:             end if
7:         end for
8:     end for
9:     return A
10: end function
```

(a) Welche (asymptotische) Laufzeit hat der Algorithmus?

(b) In welcher Weise ist A bei der Ausgabe verändert worden?

(c) Beweisen Sie die Korrektheit des Algorithmus.

Hausaufgaben zum 22. Oktober, 24:00, Abgabe per Email an Tutor

---

Aufgabe 0 (Wiederholung) Wiederholen Sie die Rechengesetze für Logarithmen und Matrizen, und lesen Sie die Wikipedia-Artikel zu "vollständige Induktion" und "Landau-Notation".

Aufgabe 1 (Landau-Notation, 2+3 Punkte) Die folgenden 14 Funktionen $f_i : \mathbb{N} \to \mathbb{R}$ stehen in keiner speziellen Reihenfolge:

$$2^n,\quad n^{0.01},\quad \log n,\quad \log(n^3),\quad n\log n,\quad n^n,\quad 1,\quad \log\log n,\quad \sqrt{n},\quad 1/n,\quad n!,\quad \log(n^{\log n}),\quad 8^n,\quad n^8$$

(a) Sortieren Sie obige Funktionen in asymptotisch aufsteigender Reihenfolge und begründen Sie kurz jede aufeinanderfolgende Paarung. Nutzen Sie die Schreibweise $f_1 \prec f_2$ für $f_1 \in o(f_2)$ und $f_1 \asymp f_2$ für $f_1 \in \Theta(f_2)$, also beispielsweise $f_1 \prec f_2 \asymp f_3 \prec f_4 \prec \ldots \prec f_{12} \asymp f_{13} \prec f_{14}$.

(b) Zeigen oder widerlegen Sie:

(i) für beliebige $b > 1$ gilt: $\log_b(n) \in \Theta(\log_2 n)$

(ii) $f \in \mathcal{O}(g) \;\Rightarrow\; g \in \omega(f)$

(iii) für $f_c(n) := \sum_{i=0}^{n} c^i$ und positives reelles c gilt: $f_c(n) \in \Theta(n) \Leftrightarrow c = 1$

Aufgabe 2 (Fibonacci I, 2+2 Punkte) Die Fibonacci-Folge $F_0, F_1, F_2, \ldots$ ist durch folgende Rekursion definiert:

$$F_0 := 0, \quad F_1 := 1, \quad F_n := F_{n-1} + F_{n-2}.$$

In dieser Aufgabe zeigen wir das exponentielle Wachstum dieser Folge.

(a) Zeigen Sie per Induktion, dass $F_n \geq 2^{0.5n}$ für alle $n \geq 6$.

(b) Finden Sie ein $c < 1$ so, dass $F_n \leq 2^{cn}$ für alle $n \geq 0$ ist, und beweisen Sie Ihre Antwort.

Aufgabe 3 (Fibonacci II, 2+2+2 Punkte) Eine alternative Berechnung der Fibonacci-Folge ist mittels 2×2 Matrizen möglich. Überzeugen Sie sich, dass gilt:

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} \quad \text{, sowie} \quad \begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^2 \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}.$$

Es folgt allgemein, dass:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \cdot \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}. \qquad (\star)$$

Zur Berechnung von F_n muss im Wesentlichen also "lediglich" $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 1 \end{smallmatrix}\right)^n$ berechnet werden.

(a) Beweisen Sie $(\star)$ für $n \geq 0$. (Tipp: vollständige Induktion).

(b) Sei X Element irgendeines Ringes (z.B. eine Matrix, oder eine Zahl). Zeigen Sie, dass $\mathcal{O}(\log n)$ Multiplikationen ausreichen um X^n zu berechnen. (Tipp: Wie lässt sich z.B. X^{64} effizienter berechnen, als mittels $X \cdot X \cdot X \cdot \ldots \cdot X$?)

(c) Die Addition zweier ℓ-Bit-Zahlen benötigt Zeit $\mathcal{O}(\ell)$, deren Multiplikation mittels geschickter Verfahren jedoch Zeit $\mathcal{O}(\ell^{1.59})$. Zeigen Sie, dass das Matrizen-Verfahren $(\star)$ zur Berechnung von F_n asymptotisch echt schneller als das in der Vorlesung mit Laufzeit $\mathcal{O}(n^2)$ vorgestellte Verfahren ist. (Tipp: Bestimmen Sie die Gesamtanzahl benötigter skalarer Rechenoperationen und deren Bitlängen)