Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Tutorenteam der Vorlesung Programmierung 1

Programmierung 1 (Wintersemester 2012/13)

Aufgaben aus den Übungsgruppen 10

(Kapitel 11)

Hinweis: Dieses Übungsblatt enthält von den Tutoren für die Übungsgruppe erstellte Aufgaben.

Die Aufgaben und die damit abgedeckten Themenbereiche sind für die Klausur weder relevant, noch irrelevant.

1 Zum Aufwärmen

Aufgabe 10.1 (*Komplexitätsklassen*)
Betrachten Sie die folgenden rekursiv definierten Funktionen und bestimmen Sie deren Komplexitäten!

- $f\ n = if\ n = 0\ then\ 0\ else\ f(n-1)$
- $fn = if\ n = 0\ then\ 1\ else\ f(n-1)$
- $fn = if\ n = 0\ then\ 2\ else\ f(n-1)$

(a) Welche Funktionen besitzen die gleiche Komplexitätsklasse?

(b) Erklären Sie, warum $\mathcal{O}(0) \neq \mathcal{O}(1)$ gelten muss!

(c) Kann die Laufzeit einer Prozedur in $\mathcal{O}(0)$ liegen? Begründen Sie!

Aufgabe 10.2 (*Vergleich der Komplexität von Prozeduren und Funktionen*)

(a) Betrachten Sie noch einmal die Aufgaben 11.18 und 11.19 auf dem regulären Übungsblatt. Was ist der Unterschied zwischen den beiden Aufgabenstellungen? Erklären Sie in eigenen Worten!

(b) Bestimmen Sie (von Aufgabe 11.18 verschiedene) Prozeduren der Komplexität 1 und n^2.
Bestimmen Sie (von Aufgabe 11.18 verschiedene) Funktionen der Komplexität 1 und n^2.

Aufgabe 10.3 (*Vergleich der Komplexität von Prozeduren und Funktionen*)

Gegeben sei die Prozedur *length*:

$$\begin{aligned} &length\ nil\ =\ nil \\ |&length\ (x :: xr)\ =\ 1\ +\ length\ xr \end{aligned}$$

Betrachten Sie die folgenden Größenfunktionen:

(a) $|xs|$

(b) $3|xs| + 7$

(c) $2^{|xs|}$

- Füllen Sie nun die entsprechende Tabelle aus:

  Größe a bedeutet hier: Bestimmen Sie die Größe einer Liste nach Größenfunktion a).

  max. LZ a bedeutet: Bestimmen Sie die maximale Laufzeit einer Liste der entsprechenden Größe nach Größenfunktion a).

$\|Liste\|$	Größe a	max. LZ a	Größe b	max. LZ b	Größe c	max. LZ c
0						
1						
2						
3						
4						
5						
10						

- Die Laufzeitfunktion ist eine Funktion, die jeweils von der Größe eines Arguments auf die entsprechende Laufzeit abbildet.

  Definieren Sie die Laufzeitfunktionen der Prozedur *length* für die obigen Größenfunktionen mit Hilfe der Tabelle!

- Geben Sie die Komplexität der verschiedenen Laufzeitfunktionen an!

- Was sagt dies über die Komplexität einer Prozedur aus? Kann man diese getrennt von der Größenfunktion betrachten?

2 Zum Üben

Aufgabe 10.4 (*Exponentieller Rekurrenzsatz*)
Die Rekurrenzsätze sind zwar nützlich und vermitteln ein gutes Bild von der Einteilung in Komplexitätsklassen, schränken in der Praxis allerdings oftmals zu sehr ein.
Betrachten Sie den *exponentiellen Rekurrenzsatz*: Finden Sie ein Beispiel für eine Prozedur, die offensichtlich exponentielle Komplexität aufweist, allerdings wegen einer kleinen Unstimmigkeit nicht vom *exponentiellen Rekurrenzsatz* erfasst wird.

Tipp: Schauen Sie sich an, warum die Definition nicht direkt auf die Funktion der Fibonacci-Folge anwendbar ist.

3 Knobelecke

Aufgabe 10.5 (*Die Sinnfrage*)
Wie aussagekräftig ist es, wenn eine Prozedur eine geringe Komplexität hat? Welche Faktoren sorgen dafür, dass eine Prozedur, die z.B. nur lineare Komplexität hat, für ein doppelt so großes Argument trotzdem um ein Vielfaches „schwerer“ zu berechnen sein kann?
Sie sollten mindestens 3 Argumente finden.

Aufgabe 10.6 (*mathematisch komplex*)

(a) Die folgende Definition der *Komplexität einer Funktion* entspricht der Ihnen bekannten Definition aus dem Buch:

$$\mathcal{O}(f) \quad := \quad \{g \in OF \mid \exists n_0 \in \mathbb{N} : \exists c \in \mathbb{N} : \forall n \geq n_0 : g\, n \leq c \cdot (f\, n)\}$$

Beschreiben Sie knapp, wann $g \in \mathcal{O}(f)$ gilt. Welche der folgenden Analogien greift am Besten?

(i) „$g < f$“ (g wächst echt langsamer als f)
(ii) „$g > f$“ (g wächst echt schneller als f)
(iii) „$g \leq f$“ (g wächst nicht schneller als f)
(iv) „$g \geq f$“ (g wächst nicht langsamer als f)

(b) Wir definieren nun zwei weitere, ähnliche Mengen:

$$\mathcal{A}(f) \quad := \quad \{g \in OF \mid \forall c \in \mathbb{R}^+ : \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : g\, n \leq c \cdot (f\, n)\}$$

$$\mathcal{B}(f) \quad := \quad \left\{g \in OF \;\middle|\; \exists c \geq 0 : lim_{n\to\infty} \tfrac{g(n)}{f(n)} = c\right\}$$

Eine dieser beiden Definitionen ist äquivalent zur Definition von $\mathcal{O}(f)$. Welche?

Bonusaufgabe: Beweisen Sie diese Äquivalenz.

Die andere Definition formalisiert eine andere Analogie aus Teil *a*. Welche?

(c) Finden Sie eine formale Definition für $\Theta(f)$, so dass $g \in \Theta(f)$ genau dann gilt, wenn f und g (bis auf endliche viele Ausnahmen und einen konstanten Faktor) gleich schnell wachsen.

Aufgabe 10.7 (*Dieter, der Große*)
Gegeben sei die folgende mathematische Prozedur:

$$\begin{array}{lcl} superDieter : \mathcal{L}(\mathbb{N}) \times X \times (X \to X) \to X & & \\ superDieter(nil, s, f) & = & s \\ superDieter(x :: xr, s, f) & = & superDieter(xr, iter(x, s, f), f) \end{array}$$

Sie können davon ausgehen, dass die Nebenkosten, die *iter* verursacht, linear vom Argument x abhängen.

(a) Geben Sie eine Größenfunktion an.

(b) Begründen Sie, warum Ihre Größenfunktion gültig ist.

(c) Geben Sie die Komplexität der Laufzeitfunktion an.

Aufgabe 10.8 (*Eine wunderhübsche Prozedur*)
Gegeben sei die folgende mathematische Prozedur:

$$\begin{array}{lcl} mystery : \mathbb{N} \to \mathbb{N} & & \\ mystery(0) & = & 2 \\ mystery(1) & = & 1 \\ mystery(n) & = & mystery(\lfloor \frac{n}{2} \rfloor) \cdot mystery(\lfloor \frac{n}{2} \rfloor) \end{array}$$

Bestimmen Sie die Komplexität der Laufzeitfunktion dieser Prozedur zur Größenfunktion $\lambda n \in \mathbb{N}.n$.