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

Programmierung 1 (Wintersemester 2012/13)

Lösungsblatt 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!

Lösung 10.1:

(a) Die Komplexitäten der Funktionen sind

- $\mathcal{O}(0)$
- $\mathcal{O}(1)$
- $\mathcal{O}(1)$

Damit sind die Komplexitäten von (2) und (3) gleich.

(b) Die Funktion $fx = 1$ liegt $\mathcal{O}(1)$. Ebenso liegt $gx = 3$ in $\mathcal{O}(1)$, denn $4 * (fx) \geq gx$ für alle außer endlich viele x und $fx \leq gx$ für alle außer endlich viele x (d.h. f dominiert g und g dominiert f). Wenn man jetzt $hx = 0$ betrachtet so wird zwar h von g dominiert ($hx \leq gx$ für alle außer endlich viele x), aber man findet keinen Faktor $c \neq 0$ so dass $c * h(x) \geq g(x)$ für alle außer endlich viele x (denn $c * h(x)$ ist immer 0, aber $c * g(x)$ ist immer $3 * x$).

(c) Eine Komplexität von $\mathcal{O}(0)$ würde bedeuten, dass Prozedur keine Zeit zum Ausführen bräuchte. Da das auf einen realen Rechner aber nie vorkommen kann, ist die kleinstmögliche Laufzeit einer Prozedur 1 und damit liegt ihre Komplexität schon in $\mathcal{O}(1)$.

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.

Lösung 10.2:

(a) In Aufgabe 11.18 geht es um die Komplexität von Funktionen. Anschaulich verstehen wir unter der Komplexität einer Funktion deren Wachstumsverhalten: Wie schnell wächst der Funktionswert, wenn wir das Funktionsargument vergrößern? Um dieses Verhalten zu untersuchen muss man sich die Funktionen nur gut anschauen und wird schnell merken, wie der Funktionswert vom Argument abhängt. Insbesondere ist es völlig falsch, zu einer Funktion eine Größenfunktion bzw. Laufzeitfunktion aufstellen zu wollen.

In Aufgabe 11.19 dagegen geht es um die Komplexität von Prozeduren. Unter der Komplexität einer Prozedur verstehen wir die Komplexität *ihrer Laufzeitfunktion*, denn uns interessiert ja, wie stark der Aufwand der Prozedur wächst, wenn wir das Argument vergrößern. Wir müssen also erstmal eine geeignete Größenfunktion finden (in diesem Fall eignet sich immer $\lambda n.n$) und mit deren Hilfe die Laufzeitfunktion aufstellen. Und deren Komplexität bestimmen wir dann (genau so, wie wir es für die Funktionen in Aufgabe 11.19 gemacht haben).

(b) Prozeduren:

$$p : \mathbb{N} \to \mathbb{N}$$
$$p\ n = n$$

$$q : \mathbb{N} \to \mathbb{N}$$
$$q\ 0 = 1$$
$$q\ n = q(n-1) + p\ n$$

Funktionen:

$$g \in \mathbb{N} \to \mathbb{N}$$
$$g\ n = 234$$

$$h \in \mathbb{N} \to \mathbb{N}$$
$$h\ 0 = 9$$
$$h\ 1 = 90$$
$$h\ 2 = 42$$
$$h\ n = h(n-3) + 789$$

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

Gegeben sei die Prozedur *length*:

$length\ nil\ =\ nil$
$|length\ (x :: xr)\ =\ 1\ +\ length\ xr$

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	0	1	7	1	1	1
1	1	2	10	2	2	2
2	2	3	13	3	4	3
3	3	4	16	4	8	4
4	4	5	19	5	16	5
5	5	6	22	6	32	6
10	10	11	37	11	1024	11

- 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?

Lösung 10.3:
Laufzeitfunktion für length:

- (a)

$$r0 = 1$$
$$rn = r(n-1) + 1 \qquad n \neq 0$$

(b)

$$rn = 1 \qquad n \leq 7$$
$$rn = r(n-3) + 1 \qquad n > 7$$

(c)

$$rn = 1 \qquad n \leq 1$$
$$rn = r(\left\lfloor \frac{n}{2} \right\rfloor) + 1 \qquad n > 1$$

- (a) $\mathcal{O}(n)$
  (b) $\mathcal{O}(n)$
  (c) $\mathcal{O}(log(n))$
- Wie man an der Aufgabe erkennen kann, beeinflussen die Größenfunktionen die Laufzeitfunktionen und somit auch deren Komplexität. Es ist also sehr wichtig sich eine sinnvolle Größenfunktion zu definieren um eine realistische Komplexität zu erhalten.

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.

Lösung 10.4:

$$p\,n = if\ n = 0\ then\ 1\ else\ p(n-2) + p(n-2) + p(n-2) + p(n-2) + p(n-2) + n^4$$

Der *exponentielle Rekurrenzsatz* lässt explizit nur $n - 1$ als Rekursionsargument zu, selbst wenn die Prozedur in 100-fache Rekursion mit n-2 gehen würde, so wäre der Satz nicht anwendbar, die Prozedur aber offensichtlich von exponentieller Komplexität.

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.

Lösung 10.5:

- Die Komplexität hängt von der Wahl der Größenfunktion ab. Durch Anpassung der Größenfunktion können Sie die Komplexität jeder Prozedur scheinbar verbessern.
- Bei der Einteilung in Komplexitätsklassen können beliebig viele Ausnahmen (durch die Wahl von n_0) ignoriert werden, so dass die Komplexität immer nur Aussagen über das asymptotische Verhalten macht.
- Für unsere mathematische Prozeduren gibt es keine Größenbeschränkung für Variablen. Eine Prozedur, die eine lineare Laufzeit hat, kann für größere Argumente trotzdem exponentiell mehr Speicherplatz brauchen, was sie oft genauso unbrauchbar macht wie eine langsame Prozedur.

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.

Lösung 10.6:

(a) Die beste Analogie ist „$g \leq f$“ (g wächst nicht schneller als f). Eine Funktion g ist genau dann in $\mathcal{O}(f)$, wenn g bis auf endlich viele Ausnahmen und einen konstanten Faktor kleinere oder gleich große Werte liefert wie f.

(b) Es gilt $\mathcal{B}(f) = \mathcal{O}(f)$.

Es gilt weiterhin: $g \in \mathcal{A}(f) \Rightarrow g \in \mathcal{O}(f)$, dann $\mathcal{A}(f)$ entspricht der Analogie „$g < f$“ (g wächst echt langsamer als f). Für $\mathcal{A}(f)$ wird üblicherweise die Notation $o(f)$ gewählt.

(c)

$$\Theta(f) := \left\{g \in OF \;\middle|\; \exists c > 0 : lim_{n\to\infty} \frac{f(n)}{g(n)} = c\right\}$$

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 $\lambda n \in \mathbb{N}.n$.

Lösung 10.8:
Wir bestimmen eine rekursive Darstellung der Laufzeitfunktion:

$$\begin{array}{lcl} r(0) & = & 1 \\ r(1) & = & 1 \\ r(n) & = & 2 \cdot (\lfloor \frac{n}{2} \rfloor) + 1 \end{array}$$

Da wir nur an der Komplexität interessiert sind, können wir den Fall $r = 0$ ignorieren und die Abrundung entfernen:

$$\begin{array}{lcl} r(1) & = & 1 \\ r(n) & = & 2 \cdot (\frac{n}{2}) + 1 \end{array}$$

Leider ist kein Rekurrenzsatz anwendbar. Wir versuchen, stattdessen eine explizite Darstellung zu erraten:

$$r(n) = 2n - 1$$

Die Korrektheit dieser Vermutung lässt sich per Induktion beweisen, worauf wir hier aber verzichten. Die gesuchte Komplexität ist also $\mathcal{O}(n)$.