Prof. Dr. Leif Kobbelt
Thomas Ströder, Fabian Emmes
Sven Middelberg, Michael Kremer

Informatik 8
RWTH Aachen

Datenstrukturen und Algorithmen (SS 2013)

Übungsblatt 4

Abgabe: Montag, **13.05.2013**, 14:00 Uhr

- Die Übungen sollen in Gruppen von zwei bis drei Personen bearbeitet werden.
- Schreiben Sie die Namen jedes Gruppenmitglieds sowie alle Matrikelnummern auf die abgegebenen Lösungen.
- Schreiben Sie die Namen jedes Gruppenmitglieds sowie alle Matrikelnummern auch in die Quellcode-Dateien.
- Geben Sie Ihre Lösungen am **Anfang** der Globalübung, montags, 14:00 Uhr, ab.
- Schicken Sie den jeweiligen Quellcode bitte per **E-Mail** direkt an Ihre/n Tutor/in.
- Geben Sie außerdem den ausgedruckten Quellcode zusammen mit den schriftlichen Lösungen ab.
- Zu spät abgegebene Lösungen werden nicht bewertet.
- Sofern nicht anders gefordert, müssen alle Lösungen und Zwischenschritte kommentiert werden.

Prof. Dr. Leif Kobbelt
Thomas Ströder, Fabian Emmes
Sven Middelberg, Michael Kremer

Informatik 8
RWTH Aachen

Aufgabe 1 (*Algorithmische Analyse* [**10 Punkte**])

1. Beweisen Sie die folgenden Eigenschaften der Komplexitätsklassen O, Ω und Θ: Für alle $f, g, h : \mathbb{N} \to \mathbb{R}^+$ gilt

   - **Transitivität:** $f \in O(g) \wedge g \in O(h) \Rightarrow f \in O(h)$, analog für Ω und Θ
   - **Reflexivität:** $f \in O(f)$, $f \in \Omega(f)$, $f \in \Theta(f)$
   - **Symmetrie:** $f \in \Theta(g) \Rightarrow g \in \Theta(f)$
   - **Antisymmetrie:** $f \in O(g) \wedge g \in O(f) \Rightarrow f \in \Theta(g)$ und analog $f \in \Omega(g) \wedge g \in \Omega(f) \Rightarrow f \in \Theta(g)$

2. Beschreiben Sie das asymptotische Verhalten folgender Funktionen möglichst genau:

   - $T(n) = 2T(n/2) + n^3$
   - $T(n) = 16T(n/4) + n^2$
   - $T(n) = 3T(n/4) + n \log n$
   - $T(3n) = 3T(2n) + \log n$

   Hierbei sei stets $T(1) = 1$.

3. Zeigen oder widerlegen Sie:

$$f \in \Theta(g) \wedge \lim_{n\to\infty} \frac{h(n)}{g(n)} = 0 \quad \Rightarrow \quad f + h \in \Theta(g).$$

4. Zeigen oder widerlegen Sie:

$$f \in O(n \log_a n) \quad \Leftrightarrow \quad f \in O(n \log_b n).$$

5. Lösen Sie die folgende Rekursionsgleichung so auf, dass sie keine Rekursion mehr enthält, d.h. nur noch von n, nicht aber von $T(n)$ abhängig ist. Die Ermittlung der Komplexitätsklasse ist nicht ausreichend.

$$\begin{aligned} T(0) &= 5 \\ 2T(n) &= nT(n-1) + 3 \cdot n! \end{aligned}$$

Lösungsvorschlag

1. • **Transitivität:**

$$f \in O(g) \Rightarrow \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : f(n) \leq c \cdot g(n)$$
$$g \in O(h) \Rightarrow \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : g(n) \leq c \cdot h(n)$$

Somit existieren Konstanten c_1, n_1, c_2, n_2 für welche gilt:

$$\forall n \geq n_1 : f(n) \leq c_1 \cdot g(n)$$
$$\forall n \geq n_2 : g(n) \leq c_2 \cdot h(n)$$

Es folgt:

$$\begin{aligned} & \forall n \geq \max(n_1, n_2) : f(n) \leq c_1 \cdot g(n) \leq c_1 \cdot c_2 \cdot h(m) \\ \Rightarrow\ & \forall n \geq \max(n_1, n_2) : f(n) \leq c_1 \cdot c_2 \cdot h(m) \\ \Rightarrow\ & \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : f(n) \leq c \cdot h(n) \\ \Rightarrow\ & f \in O(h) \end{aligned}$$

□

$$f \in \Omega(g) \Rightarrow \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : f(n) \geq c \cdot g(n)$$
$$g \in \Omega(h) \Rightarrow \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : g(n) \geq c \cdot h(n)$$

Somit existieren Konstanten c_1, n_1, c_2, n_2 für welche gilt:

$$\forall n \geq n_1 : f(n) \geq c_1 \cdot g(n)$$
$$\forall n \geq n_2 : g(n) \geq c_2 \cdot h(n)$$

Es folgt:

$$\begin{aligned} & \forall n \geq \max(n_1, n_2) : f(n) \geq c_1 \cdot g(n) \geq c_1 \cdot c_2 \cdot h(m) \\ \Rightarrow\ & \forall n \geq \max(n_1, n_2) : f(n) \geq c_1 \cdot c_2 \cdot h(m) \\ \Rightarrow\ & \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : f(n) \geq c \cdot h(n) \\ \Rightarrow\ & f \in \Omega(h) \end{aligned}$$

□

$$\begin{aligned} & f \in \Theta(g) \Rightarrow f \in O(g) \wedge f \in \Omega(g) \\ & g \in \Theta(h) \Rightarrow g \in O(h) \wedge g \in \Omega(h)) \\ \Rightarrow\ & f \in O(h) \wedge f \in \Omega(h) \\ \Rightarrow\ & f \in \Theta(h) \end{aligned}$$

□

• **Reflexivität:**
Trivialerweise gilt:

$$\begin{aligned} & \forall n \geq 1 : f(n) \leq f(n) \\ \Rightarrow\ & \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : f(n) \leq c \cdot f(n) \\ \Rightarrow\ & f \in O(f) \end{aligned}$$

□

Analog:

$$\begin{aligned} & \forall n \geq 1 : f(n) \geq f(n) \\ \Rightarrow\ & \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : f(n) \geq c \cdot f(n) \\ \Rightarrow\ & f \in \Omega(f) \end{aligned}$$

□

Es folgt:

$$f \in O(f) \wedge f \in \Omega(f) \ \Rightarrow\ f \in \Theta(f)$$

□

- **Symmetrie:**

$$f \in \Theta(g) \Rightarrow \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : \frac{f(n)}{c} \leq g(n) \leq c \cdot f(n)$$

Damit folgt:

$$\begin{aligned} & \forall n \geq n_0 : \frac{f(n)}{c} \leq g(n) \\ \Rightarrow\ & \forall n \geq n_0 : f(n) \leq c \cdot g(n) \\ & \forall n \geq n_0 : g(n) \leq c \cdot f(n) \\ \Rightarrow\ & \forall n \geq n_0 : \frac{g(n)}{c} \leq f(n) \end{aligned}$$

Und somit:

$$\forall n \geq n_0 : \frac{g(n)}{c} \leq f(n) \leq c \cdot g(n) \ \Rightarrow\ g \in \Theta(f)$$

□

- **Antisymmetrie:**

$$\begin{aligned} f \in O(g) \Rightarrow \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : f(n) \leq c \cdot g(n) \\ g \in O(f) \Rightarrow \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : g(n) \leq c \cdot f(n) \end{aligned}$$

Somit existieren Konstanten c_1, n_1, c_2, n_2 für welche gilt:

$$\begin{aligned} \forall n \geq n_1 : f(n) \leq c_1 \cdot g(n) \\ \forall n \geq n_2 : g(n) \leq c_2 \cdot f(n) \end{aligned}$$

Es folgt:

$$\begin{aligned} & \forall n \geq \max(n_1, n_2) : f(n) \leq c_1 \cdot c_2 \cdot g(n) \\ & \forall n \geq \max(n_1, n_2) : g(n) \leq c_1 \cdot c_2 \cdot f(n) \\ \Rightarrow\ & \forall n \geq \max(n_1, n_2) : \frac{g(n)}{c_1 \cdot c_2} \leq f(n) \leq c_1 \cdot c_2 \cdot g(n) \ \Rightarrow\ f \in \Theta(g) \end{aligned}$$

□

$$f \in \Omega(g) \Rightarrow \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : f(n) \geq c \cdot g(n)$$
$$g \in \Omega(f) \Rightarrow \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 : g(n) \geq c \cdot f(n)$$

Somit existieren Konstanten c_1, n_1, c_2, n_2 für welche gilt:

$$\forall n \geq n_1 : f(n) \geq c_1 \cdot g(n)$$
$$\forall n \geq n_2 : g(n) \geq c_2 \cdot f(n)$$

Es folgt:

$$\begin{aligned}
&\forall n \geq \max(n_1, n_2) : f(n) \geq c_1 \cdot c_2 \cdot g(n) \\
&\forall n \geq \max(n_1, n_2) : g(n) \geq c_1 \cdot c_2 \cdot f(n) \\
\Rightarrow\ &\forall n \geq \max(n_1, n_2) : \frac{g(n)}{c_1 \cdot c_2} \geq f(n) \geq c_1 \cdot c_2 \cdot g(n) \\
\Rightarrow\ &\forall n \geq \max(n_1, n_2) : \frac{g(n)}{c_1^{-1} \cdot c_2^{-1}} \leq f(n) \leq c_1^{-1} \cdot c_2^{-1} \cdot g(n) \ \Rightarrow\ f \in \Theta(g)
\end{aligned}$$

□

2.
- $T(n) = 2T(n/2) + n^3 = 2T(n/2) + O(n^3)$
  Master Theorem: $a = b = 2$, $p = 3 \Rightarrow a = 2 < b^p = 8$ (1. Fall), und somit $T(n) = O(n^3)$.
- $T(n) = 16T(n/4) + n^2 = 16T(n/4) + O(n^2)$
  Master Theorem: $a = 16$, $b = 4$, $p = 2 \Rightarrow a = 16 = b^p$ (2. Fall), und somit $T(n) = O(n^2 \cdot \log n)$.
- $T(n) = 3T(n/4) + n \log n = 3T(n/4) + O(n^2)$
  Master Theorem: $a = 3$, $b = 4$, $p = 2 \Rightarrow a = 3 < b^p = 16$ (1. Fall), und somit $T(n) = O(n^2)$.
- $T(3n) = 3T(2n) + \log n$
  Substitutiere $m = 3n$: $T(m) = 3T(\frac{2}{3}m) + \log \frac{m}{3} = 3T(m/1.5) + O(m)$
  Master Theorem: $a = 3$, $b = 1.5$, $p = 1 \Rightarrow a = 3 > b^p = 1.5$ (3. Fall), und somit $T(m) = O(m^{\log_{1.5} 3})$. Rücksubstitution liefert $T(3n) = O((3n)^{\log_{1.5} 3}) = O(n^{\log_{1.5} 3})$

3.
$$\lim_{n \to \infty} \frac{h(n)}{g(n)} = 0 \ \Rightarrow\ h \in O(g) \ \Rightarrow\ \exists c_1 : h(n) \leq c_1 \cdot g(n) \ \ \forall n \geq n_1$$
$$f \in \Theta(g) \ \Rightarrow\ \exists c_2 : \frac{1}{c_2} g(n) \leq f(n) \leq c_2 \cdot g(n) \ \ \forall n \geq n_2$$

Dann:

$$\frac{1}{c_1 + c_2} g(n) \leq \frac{1}{c_2} g(n) \leq f(n) \leq f(n) + h(n) \leq c_2 g(n) + c_1 g(n) = (c_1 + c_2) g(n)$$

Es folgt $f + h \in \Theta(g)$ mit $c := c_1 + c_2$ und $n_0 := \max\{n_1, n_2\}$.

4.
$$
\begin{aligned}
f \in O(n \log_a n) \quad &\Leftrightarrow \quad f(n) \leq c \cdot n \log_a n, \forall n \geq n_0 \\
&\Leftrightarrow f(n) \leq c \cdot n \cdot \frac{\ln n}{\ln a} \\
&= c \cdot n \cdot \frac{\ln b}{\ln b} \cdot \frac{\ln n}{\ln a} \\
&= c \cdot \frac{\ln b}{\ln a} \cdot n \cdot \frac{\ln n}{\ln b} \\
&= c \cdot \log_a b \cdot n \cdot \log_b n
\end{aligned}
$$

Damit folgt $f \in O(n \log_b n)$ mit $c' := c \cdot \log_a b$ und $n_0' = n_0$.

5. Multipliziere beide Seiten mit $\frac{2^{n-1}}{n!}$:

$$
\frac{2^n}{n!} T(n) = \frac{2^{n-1}}{(n-1)!} T(n-1) + 3 \cdot 2^{n-1}
$$

$$
S(n) := 2^n \frac{T(n)}{n!}
$$

Damit folgt

$$
\begin{aligned}
S(n) &= S(n-1) + 3 \cdot 2^{n-1} \\
&= S(n-2) + 3 \cdot 2^{n-2} + 3 \cdot 2^{n-1} \\
&\vdots \\
&= S(0) + \sum_{i=0}^{n-1} 3 \cdot 2^i \\
&= 5 + 3(2^n - 1) \\
&= 3 \cdot 2^n + 2
\end{aligned}
$$

und es ist

$$
\begin{aligned}
T(n) &= \frac{n!}{2^n} S(n) = \frac{n!}{2^n}(3 \cdot 2^n + 2) \\
&= 3(n!) + \frac{n!}{2^{n-1}}
\end{aligned}
$$

Aufgabe 2 (Rekursionsgleichungen [**10 Punkte**])

Gegeben sei folgende Java-Methode (der Einfachheit halber kann angenommen werden, dass der Parameter `i` stets positiv oder null ist):

```
public static int func(int i) {

  if(i == 0 || i == 1) return i;

  int f1 = func((i+1)/2);
  int f2 = func((i+1)/2 - 1);

  if(i % 2 == 1) {
    return f1 * f1 + f2 * f2;
  }

  return f1 * f1 + 2 * f1 * f2;
}
```

Beachten Sie, dass bei der Ganzzahldivision immer auf die nächstniedrigere Ganzzahl *abgerundet* wird.

(a) Bestimmen Sie die Rekursionsgleichung für die Funktion. [4 Punkte]

(b) Schätzen Sie die asymptotische Komplexität mit Hilfe des Master-Theorems ab. [4 Punkte]

(c) Beschreiben Sie, was die Funktion berechnet. [2 Punkte]

Lösungsvorschlag

(a) Wenn $i = 0$ oder $i = 1$, dann steigt der Funktionsaufruf in den ersten `if`-Zweig ab. Für die Vergleichsoperation ist der Aufwand c konstant. Für die Summe aller Vergleiche und arithmetischer Operationen im Falle $i > 1$ ist der Aufwand d ebenfalls konstant. Im zweiten Fall wird in jedem Aufruf die Funktion `func` zweimal rekursiv aufgerufen. Dabei wird der Parameter i stets für beide Aufrufe halbiert (und ggf. um eins verringert). Dies führt uns zu folgender Rekursionsgleichung:

$$\begin{aligned} T(1) &= c \\ T(n) &= 2 \cdot T(n/2) + d \\ &= 2 \cdot T(n/2) + O(n^0) \end{aligned}$$

(b) Mit Hilfe der zuvor bestimmten Darstellung lassen sich die im Master-Theorem verwendeten Koeffizienten leicht bestimmen:

$$a = 2, b = 2, p = 0$$

$$\Rightarrow 2 = a > b^p = 1$$

$$\Rightarrow T(n) \in O(n^{\log_2 2}) = O(n)$$

Dass sich die Komplexität des beschriebenen Algorithmus linear in der Eingabegröße verhält, ist auch durch Substitution, ohne Hilfe des Master-Theorems, ersichtlich. Sei der Einfachheit halber $c = d = 1$ (das können wir machen, da wir ja wissen, dass ein konstanter Vorfaktor keinen Einfluss auf die asymptotische Komplexität einer Funktion hat). Dann gilt

$$\begin{aligned}
T(n) &= 2 \cdot T(n/2) + 1 \\
&= 2 \cdot \big[2 \cdot T(n/4) + 1\big] + 1 \\
&= 4 \cdot T(n/4) + 2 + 1 \\
&= 8 \cdot T(n/8) + 4 + 2 + 1 \\
&\vdots \\
&= 2^k \cdot T(n/2^k) + \sum_{i=0}^{k-1} 2^i \\
&\vdots \\
&= n \cdot T(1) + n - 1 \quad (\text{da } n = 2^k) \\
\\
&\Rightarrow T(n) = 2 \cdot n - 1 \in O(n)
\end{aligned}$$

(c) Die Funktion berechnet rekursiv die Folge der Fibonacci-Zahlen in linearem Zeitaufwand.