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

Informatik 8
RWTH Aachen

Datenstrukturen und Algorithmen (SS 2013)

Präsenzübung
Musterlösung
Dienstag, **28.05.2013**

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

Informatik 8
RWTH Aachen

Aufgabe 1 (*Allgemeine Fragen* [20 Punkte])

1. Tragen Sie in der folgenden Tabelle die *Best-*, *Average-* und *Worst-Case-*Komplexitäten der jeweiligen Sortierverfahren in O-Notation ein. [3 Punkte]

	Best-Case-Kompl.	*Avg.-Case*-Kompl.	*Worst-Case*-Kompl.
Bubble-Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection-Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion-Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Quick-Sort	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n^2)$
Merge-Sort	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n \cdot \log n)$
Heap-Sort	$O(n)$	$O(n \cdot \log n)$	$O(n \cdot \log n)$

2. Gegeben sei eine nahezu vorsortierte Folge, von der Sie wissen, dass nur wenige Elemente unsortiert sind. Ihnen stehen vier Sortierverfahren zur Verfügung: **Bubble-Sort**, **Insertion-Sort**, **Quick-Sort** und **Merge-Sort**. Welches der vier Verfahren würden Sie zur Sortierung der Folge empfehlen? Begründen Sie Ihre Antwort in einem Satz. [3 Punkte]

   Insertion Sort, da dieses Verfahren bei vorsortierten Folgen lineare Laufzeitkomplexität hat.

3. Teilen Sie die folgenden Funktionen derart in Äquivalenzklassen auf, dass f und g genau dann in derselben Äquivalenzklasse sind, wenn $f \in \Theta(g)$ gilt. [3 Punkte]

$$
\begin{aligned}
f_1(x) &= x^2 \\
f_2(x) &= x \log x \\
f_3(x) &= x^2 + x \log x \\
f_4(x) &= x \\
f_5(x) &= 2^x \\
f_6(x) &= 1/x \\
f_7(x) &= \sqrt{x} \\
f_8(x) &= x(2 + \sin x) \\
f_9(x) &= 1
\end{aligned}
$$

$\{f_1, f_3\}$
$\{f_2\}$
$\{f_4, f_8\}$
$\{f_5\}$
$\{f_6\}$
$\{f_7\}$
$\{f_9\}$

4. Sei $f \in O(n^2 \log n)$ und $g \in O(2^n)$. Geben Sie eine *möglichst kleine* obere Schranke für die Funktion $f \circ g$ in O-Notation an. [3 Punkte]

$$f \circ g \in O((2^n)^2 \cdot \log 2^n) = O(4^n \cdot n)$$

5. Ein Ternärbaum ist ein Baum, in dem jeder Knoten maximal drei Söhne hat. Wieviele Knoten hat ein Ternärbaum der Höhe h minimal und maximal? [3 Punkte]

   Minimal: $h + 1$.

   Maximal: $\sum_{i=0}^{h} 3^i$.

6. Ist *dynamische Programmierung* ein *top-down*- oder ein *bottom-up*-Verfahren? [2 Punkte]

   Bottom-up.

7. In der Vorlesung haben Sie verschiedene Entwurfsparadigmen kennengelernt. Nach welchem Entwurfsparadigma wurde der folgende Algorithmus entworfen? Sie müssen Ihre Antwort nicht begründen. [3 Punkte]

```
Sum(A[l..r])
    if (r == l) then
        return A[r]
    m ← ⌊(l+r)/2⌋
    return Sum(A[l..m]) + Sum(A[m+1..r])
```

   Divide-and-Conquer.

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

Informatik 8
RWTH Aachen

Aufgabe 2 (*Binärbäume* [20 Punkte])

1. In der Vorlesung haben Sie die Methoden `ReadTerm`, `ReadProduct` und `ReadFactor` kennengelernt, die aus einem arithmetischen Term einen Binärbaum konstruieren. Geben Sie den Binärbaum an, welcher bei Aufruf der Methode `ReadTerm` für den Term

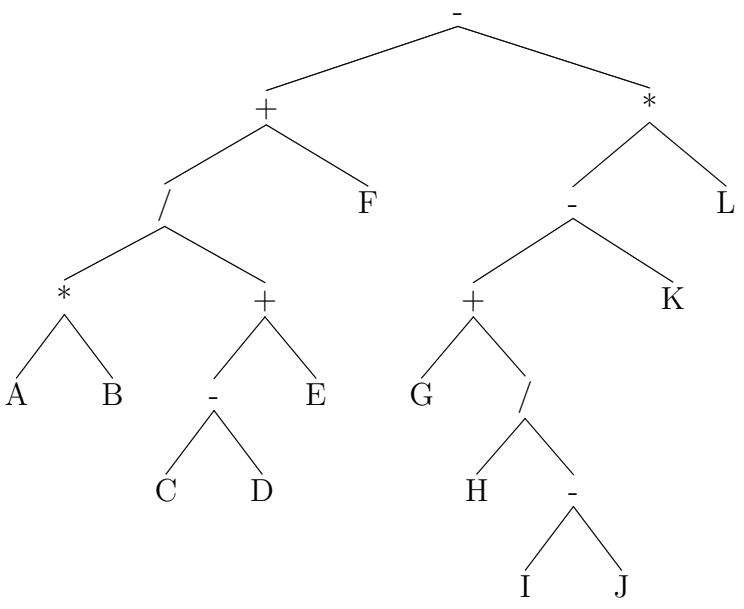
$$A * B/(C - D + E) + F - (G + H/(I - J) - K) * L$$

   entsteht. [14 Punkte]

2. In der Vorlesung haben Sie weiterhin Algorithmen zur Traversierung von Binärbäumen kennengelernt, die sich darin unterscheiden, ob der zugrundeliegende arithmetische Term in *Präfix-*, *Infix-* oder *Postfixreihenfolge* traversiert wird. Geben Sie für alle drei Traversierungsmethoden an, in welcher Reihenfolge die Knoten des von Ihnen konstruierten Binärbaums aus Aufgabenteil 1 traversiert werden. [6 Punkte]

Lösungsvorschlag

1.



2. *Präfix*: - + / * A B + - C D E F * - + G / H - I J K L

   Infix: A * B / C - D + E + F - G + H / I - J - K * L

   Anmerkung: Dieser Term ist offensichtlich aufgrund der fehlenden Klammerung nicht äquivalent zum ursprünglichen Term. Die Angabe der Klammerung ist hier aber nicht notwendig, da ausschließlich nach der Reihenfolge, in welcher die Knoten traversiert werden, gefragt wurde.

   Postfix: A B * C D - E + / F + G H I J - / + K - L * -

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

Informatik 8
RWTH Aachen

Aufgabe 3 (*Sortierverfahren* [20 Punkte])

1. Sortieren Sie das Array `[5,5,6,2,4,8]` mit **Heap-Sort**. Tragen Sie hierfür den Zustand des Arrays nach jeder Swap-Operation in der folgenden Tabelle ein. Markieren Sie, welche Elemente in jedem Schritt vertauscht werden. **Hinweis**: Die Tabelle enthält mehr Zeilen, als bei der korrekten Anwendung des Algorithmus benötigt werden. [10 Punkte]

5	5	6	2	4	8
5	5	**8**	2	4	**6**
8	5	**5**	2	4	6
8	5	**6**	2	4	**5**
5	5	6	2	4	**8**
6	5	**5**	2	4	8
4	5	5	2	**6**	8
5	**4**	5	2	6	8
2	4	5	**5**	6	8
5	4	**2**	5	6	8
2	4	**5**	5	6	8
4	**2**	5	5	6	8
2	**4**	5	5	6	8
2	4	5	5	6	8

2. Sortieren Sie das Array `[9,5,6,5,8,4,5,7,3]` mit der **rekursiven** Variante des **Merge-Sort**. Tragen Sie hierfür den Zustand des Arrays nach jeder Merge-Operation in der folgenden Tabelle ein. Markieren Sie, welche Elemente in jeder Merge-Operation betrachtet werden. **Hinweis**: Die Tabelle enthält mehr Zeilen, als bei der korrekten Anwendung des Algorithmus benötigt werden. [10 Punkte]

9	5	6	5	8	4	5	7	3
5	**9**	6	5	8	4	5	7	3
5	9	**5**	**6**	8	4	5	7	3
5	**5**	**6**	**9**	8	4	5	7	3
5	5	6	9	**4**	**8**	5	7	3
5	5	6	9	4	8	5	**3**	**7**
5	5	6	9	8	4	**3**	**5**	**7**
5	5	6	9	**3**	**4**	**5**	**7**	**8**
3	**4**	**5**	**5**	**5**	**6**	**7**	**8**	**9**

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

Informatik 8
RWTH Aachen

Aufgabe 4 (*Laufzeitanalyse/Rekursionsgleichungen* [20 Punkte])

1. Gegeben sei folgender Java-Code:

```
public static int f(int n) {
    if (n == 1) {
        return 3;
    } else {
        return f(0.25*n) + Math.sqrt(n) - 1;
    }
}
```

   Überführen Sie die Funktion `f` in eine Rekursionsgleichung für die Laufzeitabschätzung unter der Annahme, dass der Eingabeparameter n stets positiv ist und alle verwendeten arithmetischen Operationen und Vergleiche konstante Laufzeit c haben. [6 Punkte]

   Lösungsvorschlag

$$T(1) = c$$
$$T(n) = T(n/4) + c$$

2. Gegeben sei folgende Rekursionsgleichung:

$$T(1) = c$$
$$T(n) = T(\lfloor n/2 \rfloor) + 2n + c,$$

   wobei $c \in \mathbb{R}$ eine Konstante ist. Bestimmen Sie eine möglichst kleine, nichtrekursive obere Schranke für $T(n)$ ohne Zuhilfenahme des Master-Theorems und ohne Verwendung der O-Notation. **Hinweis**: Es gilt $\sum_{i=0}^{n} q^i = \frac{1-q^{n+1}}{1-q}$. [10 Punkte]

Lösungsvorschlag

Der Einfachheit halber setzen wir voraus, dass $n = 2^k, k \in \mathbb{N}_0$. Dies ändert an der asymptotischen Laufzeitkomplexität von $T(n)$ nichts. Dann gilt:

$$
\begin{aligned}
T(n) &\leq T(n/2) + 2n + c \\
&= [T(n/4) + n + c] + 2n + c \\
&= \left[\left(T(n/8) + \frac{1}{2}n + c\right) + n + c\right] + 2n + c \\
&\vdots \\
&= (\log_2 n + 1)c + 2n + n \sum_{i=0}^{\log_2 n - 2} 2^{-i} \\
&= (\log_2 n + 1)c + 2n + n \sum_{i=0}^{\log_2 n - 2} \left(\frac{1}{2}\right)^i \\
&= (\log_2 n + 1)c + 2n + n \left(\frac{1 - (\frac{1}{2})^{\log_2 n - 1}}{1 - (\frac{1}{2})}\right) \\
&= (\log_2 n + 1)c + 2n + 2n \left(1 - \left(\frac{1}{2}\right)^{\log_2 n - 1}\right) \\
&= (\log_2 n + 1)c + 2n + n \left(2 - 2 \cdot 2^{-\log_2 n + 1}\right) \\
&= (\log_2 n + 1)c + 2n + n \left(2 - 2^{-\log_2 n + 2}\right) \\
&= (\log_2 n + 1)c + 2n + 2n - 2^{-\log_2 n + 2} \cdot n \\
&= (\log_2 n + 1)c + 4n - 2^{-\log_2 n + 2} \cdot n \\
&= (\log_2 n + 1)c + 4n - \frac{n}{2^{\log_2 n - 2}} \\
&= (\log_2 n + 1)c + 4n - \frac{n}{\frac{1}{4} \cdot 2^{\log_2 n}} \\
&= (\log_2 n + 1)c + 4n - \frac{n}{\frac{1}{4} \cdot n} \\
&= 4n + (\log_2 n + 1)c - 4 \in O(n)
\end{aligned}
$$

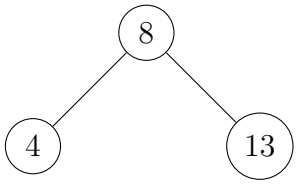
3. Benutzen Sie nun das **Master-Theorem** zur Bestimmung der asymptotischen Laufzeit der in Teilaufgabe 2 gegebenen Rekursionsgleichung. [4 Punkte]

Lösungsvorschlag

$$a = 1, b = 2, p = 1 \Rightarrow a < b^p \Rightarrow T(n) \in O(n^p) = O(n)$$

Aufgabe 5 (*Abstrakte Datentypen* [20 Punkte])

Entwerfen Sie einen abstrakten Datentyp Set, der Mengen von natürlichen Zahlen als binäre Suchbäume darstellt. Beispielsweise entspricht der folgende binäre Suchbaum der Menge $\{4, 8, 13\}$:



Der ADT soll insbesondere aus den folgenden Funktionen bestehen:

create	erzeugt eine leere Menge.
node	erzeugt einen Knoten des Binärbaums.
insert	fügt eine Zahl in eine Menge ein.
union	erzeugt die Vereinigung von zwei Mengen.

Der oben abgebildete Baum würde zum Beispiel dem folgenden Term entsprechen:

$$\text{node}\big(\text{node}(\text{create}(), 4, \text{create}()), 8, \text{node}(\text{create}(), 13, \text{create}())\big).$$

1. Geben Sie geeignete Sorten (Datentypen) und Signaturen für die Funktionen create, node, insert und union an. [5 Punkte]

2. Geben Sie geeignete Axiome an, die die Funktionen insert und union definieren. Beachten Sie, dass eine Menge keine doppelten Elemente enthalten darf. Der Vergleich von natürlichen Zahlen kann als elementare Operation betrachtet werden. [15 Punkte]

Lösungsvorschlag

1.

Sorten: $\mathbb{N}$, Set

Funktionen:

$$\begin{array}{lllll} \text{create} & : & & \to & \text{Set} \\ \text{node} & : & \text{Set} \times \mathbb{N} \times \text{Set} & \to & \text{Set} \\ \text{insert} & : & \text{Set} \times \mathbb{N} & \to & \text{Set} \\ \text{union} & : & \text{Set} \times \text{Set} & \to & \text{Set} \end{array}$$

2.

Axiome: $\forall a \in \text{A} \;\; \forall i, j \in \mathbb{R}$

$$\begin{array}{lll} (1) & \text{insert}(\text{create}(), i) & = \text{node}(\text{create}(), i, \text{create}()) \\ (2) & \text{insert}(\text{node}(s_1, j, s_2), i) & = \begin{cases} \text{node}(s_1, j, s_2) & \text{, falls } i = j \\ \text{node}(\text{insert}(s_1, i), j, s_2) & \text{, falls } i < j \\ \text{node}(s_1, j, \text{insert}(s_2, i)) & \text{, falls } i > j \end{cases} \\ (3) & \text{union}(\text{create}(), s) & = s \\ (3) & \text{union}(\text{node}(s_1, j, s_2), s) & = \text{union}(\text{union}(s_1, s_2), \text{insert}(s, j)) \end{array}$$