

Norman May

B6, 29, Raum C0.05  
68131 Mannheim  
Telefon: (0621) 181-2517  
Email: norman@pi3.informatik.uni-mannheim.de

Matthias Brantner

B6, 29, Raum C0.05  
68131 Mannheim  
Telefon: (0621) 181-2517  
Email: msb@pi3.informatik.uni-mannheim.de

Algorithmen und Datenstrukturen  
Wintersemester 2004/05

4. Lösungsvorschlag  
26. November 2004

Aufgabe 1

5 Punkte

Sei  $a$  eine Konstante mit  $a \geq 1$ . Lösen Sie folgende Rekurrenz mit Hilfe der Iterationsmethode. Geben Sie die Lösung mittels der  $\Theta$ -Notation an.

$$T(n) = T(n - a) + T(a) + n$$

Hinweis: Die Iterationsmethode wird in zwei Schritten durchgeführt. Zunächst wird für die Rekurrenz durch wiederholtes Expandieren eine Summenformel bestimmt. Anschließend wird für diese eine asymptotische Abschätzung ermittelt. Gehen Sie davon aus, daß  $T(c) = \Theta(1)$  für eine kleine Konstante  $c > 0$ .

Lösung

Bestimmen der Summe durch wiederholtes Expandieren der Rekurrenz:

$$\begin{aligned} T(n) &= T(n - a) + T(a) + n \\ &= T(n - 2a) + T(a) + (n - a) + T(a) + n \\ &= T(n - 3a) + T(a) + (n - 2a) + T(a) + (n - a) + T(a) + n \\ &\vdots \\ &= T(c) + T(a) + (n - \lfloor n/a \rfloor a) + \dots \end{aligned}$$

Insgesamt ergeben sich  $\lfloor n/a \rfloor$  rekursive Aufrufe, bis  $T(c)$  erreicht ist. Daraus folgt die Summenformel:

$$\sum_{i=0}^{\lfloor n/a \rfloor} n - ia + T(a)$$

Bestimmen der oberen Schranke

$$\begin{aligned}\sum_{i=0}^{\lfloor n/a \rfloor} n - ia + T(a) &\leq \sum_{i=0}^{\lfloor n/a \rfloor} n + T(a) \\ &\leq n^2 + nT(a) \\ &= O(n^2)\end{aligned}$$

Bestimmen der unteren Schranke:

$$\begin{aligned}\sum_{i=0}^{\lfloor n/a \rfloor} n - ia + T(a) &\geq \sum_{i=0}^{\lfloor n/2a \rfloor} n - ia \\ &\geq \sum_{i=0}^{\lfloor n/2a \rfloor} n - \left\lfloor \frac{n}{2a} \right\rfloor a \\ &\geq \sum_{i=0}^{\lfloor n/2a \rfloor} n - \frac{n}{2a} a \\ &= \sum_{i=0}^{\lfloor n/2a \rfloor} n/2 \\ &= \Omega(n^2)\end{aligned}$$

---

Aufgabe 2

5 Punkte

---

Lösen Sie folgende Rekurrenz mittels der Substitutionsmethode. Zeigen Sie, daß für

$$T(n) = 3T(n/3 + 5) + n/2$$

gilt:  $T(n) = \Theta(n \lg n)$ .

Hinweis: Bei der Substitutionsmethode werden asymptotische Schranken für die gegebene Rekurrenz geraten. Deren Korrektheit wird mittels vollständiger Induktion nachgewiesen. Gehen Sie davon aus, daß  $T(c) = \Theta(1)$  für  $c \geq 0$ .

**Lösung**

Annahme:  $T(n) = \Theta(n \lg n)$

Beweis per vollständiger Induktion.

Induktionsanfang:

Der Induktionsanfang folgt aus der Annahme daß  $T(n)$  konstant ist für  $n \leq 2$ .

Induktionsschluß:

Sei  $T(k) \leq ck \lg k$  für  $k < n, c > 0$ . Es bleibt zu zeigen, daß  $T(n) \leq cn \lg n$

$$\begin{aligned}
T(n) &= 3T(n/3 + 5) + n/2 \\
&\leq 3c(n/3 + 5) \lg(n/3 + 5) + n/2 \quad (\text{da } n/3 + 5 < n, n > 30) \\
&\leq cn \lg(n/2) + 15c \lg(n/2) + n/2 \\
&= cn \lg n - (cn - 15c \lg(n/2) - n/2) \\
&< cn \lg n, \text{ für } c = 1, n > 220
\end{aligned}$$

Sei,  $T(k) \geq ck \lg k$  für  $k < n$  für  $c > 0$ . Zu zeigen,  $T(n) \geq cn \lg n$ .

$$\begin{aligned}
T(n) &= 3T(n/3 + 5) + n/2 \\
&\geq 3c(n/3 + 5) \lg(n/3 + 5) + n/2 \quad (\text{da } n/3 + 5 < n) \\
&\geq 3c(n/3 + 5) \lg(n/3) + n/2 \\
&\geq cn \lg n - cn \lg 3 + n/2 \\
&= cn \lg n - (cn \lg 3 - n/2) \\
&\geq cn \lg n, \text{ für } c = \frac{1}{2 \lg 3}, n \geq 1
\end{aligned}$$

---

Aufgabe 3

10 Punkte

---

Lösen ( $\Theta$ -Notation) Sie folgende Rekurrenzen.

Aufgabe 3 a)

2 Punkte

$$T(n) = 2T(n/2) + n^3$$

Lösung

$$T(n) = 2T(n/2) + n^3$$

Lösen mittels Master-Theorems (3. Fall), da  $\log_b a = \log_2 2 = 1, c = 1/2$ .

$$T(n) = \Theta(n^3)$$

Aufgabe 3 b)

2 Punkte

$$T(n) = T(n - 1) + \lg n$$

Lösung

$$T(n) = T(n - 1) + \lg n$$

Lösen mittels Iterationsmethode:

Obere Schranke:

$$\begin{aligned} T(n) &= T(n - 1) + \lg n \\ &= T(n - 2) + \lg(n - 1) + \lg n \\ &\vdots \\ &= T(1) + \sum_{i=2}^n \lg i \\ &\leq T(1) + T(1) + \sum_{i=2}^n \lg n \\ &= \Theta(1) + (n - 1) \lg n \\ &= O(n \lg n) \end{aligned}$$

Untere Schranke:

$$\begin{aligned} T(n) &= T(n - 1) + \lg n \\ &= T(1) + \sum_{i=2}^n \lg i \\ &= T(1) + \sum_{i=2}^{\lceil n/2 \rceil - 1} \lg i + \sum_{i=\lceil n/2 \rceil}^n \lg i \\ &\geq \sum_{i=\lceil n/2 \rceil}^n \lg(n/2) \\ &= (n - \lceil n/2 \rceil + 1) \lg(n/2) = \Omega(n \lg n) \end{aligned}$$

Aufgabe 3 c)

2 Punkte

$$T(n) = 16T(n/4) + n^2$$

Lösung

$$T(n) = 16T(n/4) + n^2$$

Lösen mittels Master Theorem (2. Fall), da  $\log_b a = \log_4 16 = 2$

$$T(n) = \Theta(n^2 \lg n)$$

Aufgabe 3 d)

2 Punkte

$$T(n) = 7T(n/2) + n^2$$

Lösung

$$T(n) = 7T(n/2) + n^2$$

Lösen mittels Master-Theorms (1. Fall), da  $\log_b a = \log_2 7 - \epsilon = 2$

$$T(n) = \Theta(n^{\lg 7})$$

Aufgabe 3 e)

2 Punkte

$$T(n) = T(n-1) + \frac{1}{n}$$

Lösung

$$T(n) = T(n-1) + \frac{1}{n}$$

Lösen mittels Iterationsmethode:

$$\begin{aligned} T(n) &= T(n-1) + \frac{1}{n} \\ &= T(n-2) + \frac{1}{n-1} + \frac{1}{n} \\ &\vdots \\ &= T(1) + \frac{1}{2} + \dots + \frac{1}{n-1} + \frac{1}{n} \\ &= \Theta(1) + \sum_{i=2}^n \frac{1}{i} \\ &= \Theta(1) + (\ln n + O(1) - 1) \\ &= \Theta(\lg n) \end{aligned}$$

Erläuterung: Harmonische Reihe  $H_n = \sum_{i=1}^n \frac{1}{i} = \ln + O(1)$

---

Aufgabe 4

14 Punkte

Bei Datenbeständen, die größer sind als der Hauptspeicher, wird das externe Sortieren angewendet. Die Sortierung des gesamten Datenbestands erfolgt in 2 Schritten:

1. *Partitionierung* des gesamten Daten und Sortierung je einer Partition im Hauptspeicher. Die sortierten Partitionen (*Runs*) werden i.a. auf den Hintergrundspeicher geschrieben.
2. *Mischen* der Runs zum sortierten Ergebnis.

Implementieren Sie externes Sortieren, wobei ein Parameter die Größe des Hauptspeichers in Anzahl der Elemente angibt.

Aufgabe 4 a)

6 Punkte

Implementieren Sie Heapsort. Der Sortieralgorithmus bekommt ein Array bzw. eine Liste als Eingabe und liefert die sortierte Datenstruktur als Ergebnis.

Lösung

(siehe Heap.java und HeapSort.java)

Aufgabe 4 b)

2 Punkte

Gegeben Sei folgender Sortieralgorithmus:

AnotherSort(array A)

```
% bugfix: j>2 ist falsch , es muss j>=2 heissen
for (j = A.length; j >= 2; --j)
  for (i = 1; i < j; ++i)
    if (A[i] > A[i+1])
      swap(A[i], A[i+1])
```

Leiten Sie die Komplexität des Algorithmus in  $\Theta$ -Notation her. Vergleichen Sie die Komplexität mit der des Heapsort-Algorithmus. Welche Algorithmus ist besser?

Lösung

Für ein Array der Länge  $n$  wird die äußere Schleife  $n - 1$ -mal ausgeführt. Die innere Schleife wird  $\sum_{i=1}^{n-1} i$  mal durchlaufen. Daraus folgt:

$$\begin{aligned}
 T(n) &= \left( \sum_{i=1}^{n-1} i \right) \cdot C \\
 &= \frac{(n-1)(n)}{2} \cdot C \\
 &= C/2 \cdot (n^2 - n) \\
 &= \Theta(n^2)
 \end{aligned}$$

wenn  $C = \Theta(1)$  und  $C$  die Kosten der der Rumpfes der inneren Schleife sind.

Aus der Vorlesung ist bekannt, daß Heapsort eine Komplexität von  $O(n \lg n)$  hat. Deshalb ist der Heapsort-Algorithmus vorzuziehen.

Der Algorithmus kann jedoch leicht so erweitert werden, daß er im besten Fall in  $\Theta(n)$  arbeitet. Dazu muß ein Flag eingeführt werden, daß sich merkt, ob bei einer Ausführung der inneren Schleife Vertauschungen vorgenommen wurden. Ist das nicht der Fall, kann die Sortierung vorzeitig beendet werden. Wenn die Eingabedaten bereits sortiert sind, muß die innere Schleife nur genau einmal durchlaufen werden. Dann ist dieser Sortier-Algorithmus dem Heapsort-Algorithmus vorzuziehen.

Anmerkung: Der Sortieralgorithmus wird als *BubbleSort* bezeichnet.

Aufgabe 4 c)

6 Punkte

Implementieren Sie die Partitionierung und den Mischvorgang. Gehen Sie davon aus, dass nur ein Mischvorgang nötig ist. Orientieren Sie sich an folgender Java-Schnittstelle:

```
public class ExternalSort {  
  
    /* constructor */  
    public ExternalSort(int memSize) { ... }  
  
    **  
    * partition the input and sort each partition  
    * @param filename the input file to read from  
    */  
    public void partition(String filename) { ... }  
  
    **  
    * merge the partitions into the sorted result  
    * @param filename the output file for the sorted result  
    */  
    public void merge(String filename) { ... }  
    ...  
}
```

Hinweis: Lesen von Daten aus einer Datei kann z.B. mit folgendem Codefragment durchgeführt werden. Sie können annehmen, dass nur atomare Werte eines vorher bekannten Typs (z.B. int) gelesen und geschrieben werden.

```
DataInputStream input =  
    new DataInputStream(new FileInputStream(filename));  
try {  
    while (true) {  
        int val = input.readInt();  
    }  
} catch (EOFException e) {  
} finally {  
    input.close ();  
}
```

Schreiben von Daten erfolgt analog (ohne Ausnahmebehandlung):

```
DataOutputStream output =  
    new DataOutputStream(new FileOutputStream(filename));
```

```
for (int i = 0; i < size; ++i) {  
    int val = 5;  
    input.writeInt(val);  
}  
input.close();
```

Wenn Sie mit der Ein- und Ausgabe in Dateien nicht zurecht kommen, verwenden Sie Arrays oder Listen sowohl für die Ein- und Ausgabe als auch für die sortierten Runs.

Wenn Sie keinen der Sortieralgorithmen implementiert haben, können sie z.B. in Java `java.util.Collections.sort(List)` oder `java.util.Arrays.sort(Array)` verwenden oder die unsortierten Runs mischen (aber unter der Annahme, daß sie sortiert sind).

## Lösung

(siehe ExternalSort.java)