

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

5. Lösungsvorschlag
03. Dezember 2004

Aufgabe 1

16 Punkte

Beim externen Sortieren spielt der Aufwand für die Hauptspeichersortierung im Vergleich zum Ausschreiben und Einlesen der temporären Daten nur eine untergeordnete Rolle. Daher ziehen wir bei der Bestimmung der Komplexitätsklasse die Anzahl der I/O-Operationen heran (also die Anzahl der gelesenen und geschriebenen Tupel).

Sollen x Tupel sortiert werden, und steht im Hauptspeicher Platz für m Tupel zur Verfügung, so werden in der Partitionierungsphase $i = \lceil x/m \rceil$ Runs erzeugt. Entstehen dabei mehr als m Runs, muß die Merge-Phase rekursiv wiederholt werden, wobei je $m - 1$ Runs in einen neuen Run gemischt und geschrieben werden.

Aufgabe 1 a)

4 Punkte

Zeigen Sie, daß bei dem genannten rekursiven Mischverfahren insgesamt $2 \cdot x \cdot \lceil \log_{m-1}(i) \rceil$ Tupel geschrieben und gelesen werden müssen.

Lösung

Der Rekursionsbaum hat die Höhe $\lceil \log_{m-1}(i) \rceil$, wobei in jedem Rekursionsschritt ein Tupel gelesen und wieder geschrieben wird. Insgesamt sind also $2 \cdot x \cdot \lceil \log_{m-1}(i) \rceil$ I/O-Operationen notwendig.

Aufgabe 1 b)

6 Punkte

Erweitern Sie den Heap vom vorherigen Übungsblatt um eine Methode `extract`, die das minimale Element aus dem Heap extrahiert, und eine Methode `insert`, die ein neues Element in den Heap einfügt und die Heapeigenschaft wieder herstellt.

Lösung

(siehe Heap.java)

Aufgabe 1 c)

6 Punkte

Um die Anzahl der Runs zu vermindern, kann der Heapsort-Algorithmus folgendermaßen geändert werden. Zuerst wird der Heap gefüllt und mit `buildHeap` gebaut. Anschließend wird so lange das minimale Element entfernt und ein weiteres Element in den Heap eingefügt, bis das zuletzt extrahierte Element größer ist als das neu einzufügende Element y . In diesem Fall wird der Heapinhalt in den aktuellen Run eingefügt und ein neuer Run begonnen, der auch das Element y enthält. Erweitern Sie Ihre `HeapSort`-Implementierung wie beschrieben.

Ermitteln Sie experimentell die durchschnittliche Run-Länge und vergleichen sie diese mit der Länge bei Verwendung des Heap-Sort-Algorithmus.

Lösung

Der beschriebene Algorithmus wird *Replacement-Selection* genannt. Durch die Vergrößerung der Run-Länge um einen Faktor von etwa 2, kann die Anzahl der Mischvorgänge und somit der I/O-Aufwand verringert werden.

Implementation: siehe `Sort.java` `ExternalSort.java` `ReplaceSelectSort.java`.

Aufgabe 2

6 Punkte

Wie in der Vorlesung gezeigt wurde, benötigt ein vergleichsbasiertes Sortierverfahren zur Sortierung einer Eingabe der Länge n bei $n!$ möglichen Permutationen im Worst-Case $\Omega(n \lg n)$ Vergleiche. Wie verhält sich die Untere Schranke bei Reduktion der möglichen Permutationen der Eingabe? Geben Sie eine Untere Schranke für den Worst-Case für folgende Reduktionen an.

Aufgabe 2 a)

3 Punkte

Sei die Anzahl der möglichen Permutationen $n!/n$.

Lösung

In einem Binärbaum der Höhe h gilt, daß maximal 2^h Blätter vorhanden sein können.

$$\begin{aligned} 2^h &\geq n!/n \\ \Leftrightarrow 2^h &\geq (n-1)! \\ \Leftrightarrow h &\geq \lg((n-1)!) \\ \Rightarrow h &= \Omega(n \lg n) \quad (\text{siehe Übung 2.1}) \end{aligned}$$

Aufgabe 2 b)

3 Punkte

Sei die Anzahl der möglichen Permutationen $\frac{n!}{2^n}$.

Lösung

$$\begin{aligned} 2^h &\geq n!/(2^n) \\ \Leftrightarrow 2^{h+n} &\geq n! \\ \Leftrightarrow h + n &\geq \lg(n!) \\ \Rightarrow h + n &\geq \lg(n/e)^n \\ \Leftrightarrow h &\geq n \lg n - n \lg e - n \\ \Rightarrow h &= \Omega(n \lg n) \end{aligned}$$

Aufgabe 3	4 Punkte
-----------	----------

Es sei ein Array A mit n Datensätzen gegeben. Die Schlüssel haben den Wert 0 oder 1.

Aufgabe 3 a)	2 Punkte
--------------	----------

Entwerfen Sie einen In-Place Algorithmus mit linearer Laufzeit, der die Datensätze sortiert.

Lösung

Verwenden von PARTITION mit vorgegebenen Pivot-Element $x = 0.5$.

Aufgabe 3 b)	2 Punkte
--------------	----------

Kann der Algorithmus dazu verwendet werden, n Datensätze mit einem b -Bit langen Schlüssel mit Hilfe von *Radix-Sort* in-place zu sortieren? Die Laufzeit soll $O(nb)$ betragen. Begründen Sie ihre Aussage.

Lösung

Nein, da der Algorithmus nicht stabil ist.

Aufgabe 4	6 Punkte
-----------	----------

Es seien n Datensätze gegeben. Der Wertebereich der Schlüssel soll $1, \dots, k$ sein. Geben Sie eine *Counting-Sort*-Variante an, deren Speicherplatzbedarf, abgesehen von der Eingabe, $O(k)$ beträgt. Die zum Sortieren der Datensätze benötigte Laufzeit soll $O(n + k)$ betragen.

Lösung

Bei der In-Place-Variante von COUNTING-SORT werden die Elemente innerhalb des Arrays umkopiert. Die Stelle wohin ein Element kopiert werden muß, wird im Hilfs-

Array D gespeichert. Mit dem Hilfs-Array C wird bestimmt, ob ein Element umkopiert werden muß. Für die In-Place-Variante des COUNTING-SORT Algorithmus werden somit zwei Hilfsarrays (C, D) der Größe k verwendet.

COUNTING-SORT' (A, k)

```

1 for  $i \leftarrow 1$  to  $k$ 
2   do  $C[i] \leftarrow 0$ 
3 for  $j \leftarrow 1$  to  $length[A]$ 
4   do  $C[A[j]] \leftarrow C[A[j]] + 1$ 
5  $\triangleright C[i]$  enthält nun die Elemente  $= i$ .
6 for  $i \leftarrow 2$  to  $k$ 
7   do  $C[i] \leftarrow C[i] + C[i - 1]$ 
8  $\triangleright C[i]$  enthält nun die Elemente  $\leq i$ .
9 for  $i \leftarrow 1$  to  $k$ 
10  do  $D[i] \leftarrow C[i]$ 
11  $i \leftarrow length[A]$ 
12 while  $i > 1$ 
13  do
14    if  $C[A[i]] \neq i$ 
15      then  $x \leftarrow A[i]$ 
16         $A[D[x]] \leftrightarrow A[i]$ 
17         $D[x] \leftarrow D[x] - 1$ 
18         $\triangleright x$  befindet sich jetzt an der richtigen Stelle in A,
19         $\triangleright D[x]$  zeigt nun an die Stelle wohin das nächste Element  $= x$ 
20         $\triangleright$  kopiert werden muß
21    else
22       $C[A[i]] \leftarrow C[A[i]] - 1$ 
23       $D[A[i]] \leftarrow \min(C[A[i]], D[A[i]])$ 
24       $i \leftarrow i - 1$ 

```