Prof. Dr. V. Linnemann
Nils Hoeller
Jana Neumann
Universität zu Lübeck
Institut für Informationssysteme

Lübeck, den 20. Mai 2009

Algorithmen und Datenstrukturen

Sommersemester 2009

7. Übungsblatt

Aufgabe 1: Einfache Sortieralgorithmen

Die Sortieralgorithmen in der Vorlesung basieren auf einem bereits existierenden Array mit unsortierten Werten. Um beliebig große Datenmengen zu sortieren, sollen jetzt die zu sortierenden Werte aus einer Datei gelesen werden. Nutzen Sie zum Einlesen das info1package, wie in den Hinweisen beschrieben. Die sortierten Werte sollen anschließend in eine zweite Datei geschrieben werden (siehe Hinweis 3). Um die Leistungsfähigkeit der Algorithmen zu beurteilen, sollen die benötigten Zeiten gemessen werden (siehe Hinweis 4). Es stehen Ihnen 3 unterschiedlich große Dateien mit unsortierten Datensätzen zum Download auf der Übungsseite zur Verfügung.

Im Folgenden sollen Sie verschiedene Sortieralgorithmen implementieren. Messen Sie für jede Implementierung und für jede der drei Dateien die benötigte Gesamtzeit (inkl. Lesen und Schreiben der Dateien) und die Zeit, die nur zum Sortieren benötigt wird. Geben Sie diese Zeiten und die jeweilige Anzahl der sortierten Datensätze an.

a) Implementieren Sie einen Selection-Sort Algorithmus (kleinstes Element nach vorne, Direktes Aussuchen), der ein Array nutzt. Das Array muss dynamisch angelegt werden, wenn die Anzahl der zu sortierenden Werte bestimmt wurde. (2 Punkte)

b) Implementieren Sie analog zu a) den Bubble-Sort Algorithmus. (2 Punkte)

c) Implementieren Sie einen Selection-Sort Algorithmus, der eine Lineare Liste nutzt. Beim Vertauschen sollen die Werte in den Listenelementen vertauscht werden.

(2 Punkte)

d) Ändern Sie die Implementierung von c), so dass die Listenelemente vertauscht werden. (2 Punkte)

Aufgabe 2: Sortierzeiten

Alle oben implementierten Algorithmen besitzen quadratische Laufzeit zum Sortieren von n Elementen. Die Sortierzeit $t(n)$ soll durch folgende Funktion abgeschätzt werden:

$$t(n) = a \cdot n^b$$

a) Bestimmen Sie anhand Ihrer gemessenen Zeiten beim Sortieren der mittleren und langen Datei die Koeffizienten a und b für jede Ihrer 4 Implementierungen. (2 Punkte)

b) Extrapolieren Sie den benötigten Zeitaufwand zum Sortieren von 80 Mio Einträgen (entspricht ungefähr der Einwohnerzahl Deutschlands). Geben Sie die errechneten Zeiten für jede der 4 Implementierungen in sinnvollen Einheiten an. (1 Punkt)

Aufgabe 3: Quicksort

Die in der Aufgabe 1 angegebenen Dateien sollen jetzt mit dem Quicksort-Algorithmus sortiert werden. Dazu soll der in der Vorlesung auf einem Array operierende Algorithmus auf eine Liste wie folgt angewandt werden:

Gegeben sei eine zu sortierende Liste l; das erste Element der Liste sei das sogenannte *Pivot*-Element p, mit dem alle weiteren Elemente der Liste verglichen werden. Elemente die kleiner als p sind, werden in eine Liste k eingefügt, Elemente die größer sind in eine Liste g. Auf diese Listen wird rekursiv der Sortieralgorithmus ausgeführt. Das Abbruchkriterium der Rekursion ist erreicht, wenn eine Liste leer ist oder nur noch ein Element enthält. Der Algorithmus gibt jeweils die zusammengesetzte Liste bestehend aus k, p und g zurück.

a) Implementieren Sie den Algorithmus in Java. Sortieren Sie die drei Dateien, die auf der Übungsseite zu finden sind und messen Sie dabei die Sortierzeiten wie in Aufgabe 1. (5 Punkte)

b) Zeichnen Sie die Sortierzeiten in Abhängigkeit der zu sortierenden Elemente n in ein Koordinatensystem. (1 Punkt)

c) Laut Vorlesung besitzt Quicksort im Mittel eine Laufzeit von $n \cdot log(n)$. Die Sortierzeit $t(n)$ soll durch folgende Funktion abgeschätzt werden:

$$t(n) = a \cdot n \cdot log(b \cdot n)$$

Bestimmen Sie anhand Ihrer gemessenen Zeiten beim Sortieren der mittleren und langen Datei die Koeffizienten a und b. Extrapolieren Sie analog zu Aufgabe 2 den benötigten Zeitaufwand zum Sortieren von 80 Mio Einträgen. (2 Punkte)

d) Wie muss die zu sortierende Liste aufgebaut sein, damit der beschriebene Quicksort-Algorithmus nicht mehr effizient sortiert ? Begründen Sie Ihre Antwort. (1 Punkt)

Hinweis 1: Abgabe Senden Sie ihre Lösung zusätzlich per Email an die bekannten Email-Adressen.

Hinweis 2: Ein und Ausgabe Um aus bzw. in Dateien zu lesen/schreiben können Sie das info1package nutzen, welches Ihnen die Arbeit erleichtert. Laden Sie es hierzu von der Übungsseite herunter. Die zu sortierenden Dateien enthalten einen Eintrag pro Zeile. Folgender Beispielcode zeigt, wie Zeilen aus einer Datei gelesen werden können:

```
1 Eingabe in =  Eingabe.oeffnen("c:\\eintraege.txt");
2 String zeile = in.readString();
3 while (!zeile.equals("")){  // Gibt es noch Zeichen?
4
5   // zeile verarbeiten, z.B:
6   System.out.println(zeile);
7
8   zeile = in.readString;  // Nächste Zeile lesen
9 }
```

Hinweis 3: Schreiben in eine Datei Der folgende Beispielcode erstellt eine Datei mit dem Namen *ausgabe.txt* und schreibt 2 Zeilen.

```
1 Ausgabe a = Ausgabe.oeffnen("ausgabe.txt");
2 a.println("tritra");
3 a.println("trulala");
```

Hinweis 4: Zeitmessungen Die Klasse *java.lang.System* stellt eine Methode *currentTimeMillis* zur Verfügung, die die seit dem Systemstart vergangene Zeit in Millisekunden angibt. Der folgende Beispielcode errechnet die benötigte Zeit für einen Methodenaufruf.

```
1 long a = System.currentTimeMillis();
2
3 Methodenaufruf();
4
5 long b = System.currentTimeMillis();
6 System.out.println("Benötigte Zeit in ms: "+ (b-a));
```

Abgabetermin: Donnerstag, den 28. Mai bis 11 Uhr Abgabekasten IFIS