

Übungen zu Algorithmen

Wintersemester 2003/2004

Blatt 9

Aufgabe 9.1 (20 Punkte)

Bei Quicksort gibt es als Pivotstrategie den *median of three*. Hierzu wird das erste, mittlere und letzte Element des zu sortierenden Arrays betrachtet und sortiert. Somit steht der Median der drei betrachteten (und sortierten) Zahlen in der Mitte des Arrays. Danach kann Quicksort genauso wie im Skript weiterarbeiten. Implementieren Sie diese Vorgehensweise in einer Klasse `QuickSortMoT` in der Methode `public static void sort (int[] a, int unten, int oben)`.

Bei der Implementation kommt es nicht nur auf die Umsetzung der Vorgehensweise an, sondern auch auf eine sehr effiziente Programmierung.

Testen Sie die Methode `sort` aus der Klasse `QuickSortMoT` in der Klasse `Aufg9_1`.

Aufgabe 9.2 (35 Punkte)

Vergleichen Sie experimentell die Laufzeiten von `BubbleSort` (Skript 7.2), dem verbesserten `BubbleSort` aus Aufgabe 7.2a), `Shakersort` aus Aufgabe 7.2b), rekursivem `MergeSort` aus Aufgabe 7.4, den drei `QuickSort`-Implementationen (Skript 7.4, Aufgabe 8.3 b), Aufgabe 9.1) und `HeapSort` (Skript 7.6).

Erzeugen Sie dazu jeweils zehnmal eine Zufallsfolge von 2^i Zahlen mit $i = 7, 8, \dots, 13$ und sortieren Sie diese Folge mit jedem der o.g. Sortierverfahren. Führen Sie die Messung für $i = 14, \dots, 17$ mit den 5 letztgenannten Algorithmen fort. Auf schnellen Rechnern sind ggf. noch längere Zufallsfolgen zu wählen. Mitteln Sie die Zeitmeßwerte pro Verfahren und Folgenlänge. Die Klasse `Aufg9_2.java` soll eine Tabelle dieser Mittelwerte in Millisekunden ausgeben.

Erklären Sie ihrem Tutor die Ergebnisse.

Hinweis: Verwenden Sie zum zufälligen Füllen eines Arrays die Methode `Zufall.fuelleFeld(int[] feld)` aus `/home/ainf/Uebung/Blatt9/Zufall.java`. Um die Laufzeiten der Algorithmen zu messen, können die Methoden der Klasse `Stoppuhr` aus `/home/ainf/Uebung/Blatt9/Stoppuhr.java` verwendet werden. `Stoppuhr.start()` startet die Stoppuhr, `Stoppuhr.stop()` liefert die Anzahl der vergangenen Millisekunden seit dem letzten Aufruf von `Stoppuhr.start()`. Zum kopieren von Arrayinhalten sollten Sie `System.arraycopy(...)` verwenden.

Teilen Sie ihrem Tutor mit, in welchen Fällen Sie Implementationen aus der Vorlesung oder den Musterlösungen benutzt haben und wo Sie Ihre eigenen Implementationen verwendet haben.

Aufgabe 9.3 (25 Punkte)

Überprüfen Sie experimentell die Laufzeit von `BucketSort` (Skript, Abschn. 7.8) in `/home/ainf/Vorlesung/BucketSort.java`.

Sortieren Sie dazu jeweils 10-mal Zufallsfolgen von 2^i Buchstaben mit $i = 13, 14, \dots, 20$ und miteln Sie die Zeitmeßwerte für jede Folgenlänge. Die Klasse `Aufg9_3.java` soll eine Tabelle dieser Mittelwerte in Millisekunden ausgeben.

Hinweis: Verwenden Sie zum zufälligen erzeugen der Worte die Methode `Zufall.fuelleBuchstabenFeld` aus `/home/ainf/Uebung/Blatt9/Zufall.java`.

Die Zeitmessung soll wie in `Aufg9_2` erfolgen. Entsprechen die Ergebnisse Ihren Erwartungen? Starten Sie das Programm abermals mit

```
java -Xms128MB -Xmx128MB Aufg9_3
```

und erklären Sie Ihrem Tutor den Unterschied. Lesen Sie dazu die Dokumentation des Befehls `java`. Was hat es in diesem Zusammenhang mit dem *Garbage Collector* auf sich?

Aufgabe 9.4 (20 Punkte)

Sortieren Sie von Hand (schriftlich) ziffernweise die Zahlenfolge 2931, 8603, 8888, 4242, 2323, 2001, 9365, 2708, 1959, 3428, 4444, 9087, 5931, 7400 und 6661 nach der Idee des Radix Sort. Das Alphabet soll aus den Ziffern 0 bis 9 bestehen.

a) Verdeutlichen Sie die Arbeitsweise des Radix Sort. Stellen Sie die Buckets nach jeder Verteilung dar, und geben Sie die Listen W_4 bis W_0 an.

b) Wie können mit Radixsort Zahlen mit unterschiedlicher Stellenzahl sortiert werden, z.B. die Zahlen 1046, 3, 406 und 27?

Aufgabe 9.5 (0 Punkte)

Der $vi(m)$ ist individuell programmierbar. Der vi bzw. vim liest bei jedem Aufruf die Datei `.exrc` bzw. `.vimrc` im Heimatverzeichnis, in der Einstellungen automatisch vorgenommen werden können. Mit der Datei `/home/ainf/Uebung/Blatt9/.vimrc` in Ihrem Heimatverzeichnis oder im aktuellen Verzeichnis hat der $vi(m)$ zusätzliche Funktionen. Fügen Sie weitere hinzu.

Mode	Kommando	Erklärung
K	<code>Ctrl k</code>	Speichert und übersetzt das geladene Java-Programm.
K	<code>Ctrl A</code>	Führt das übersetzte Java-Programm aus.
K	<code>Ctrl w</code>	Ersetzt Tabulatorzeichen durch vier Leerzeichen.
I	<code>head_Whitespace</code>	Ein kompletter Java-Programm-Header wird eingefügt.

Um mit dem $vi(m)$ Textteile durch ein UNIX-Kommando zu filtern und das Ergebnis wieder im $vi(m)$ zu erhalten, gibt es spezielle Kommandos, die mit einem `!` beginnen und eine Fahrbewegung beinhalten. Alle dadurch eingeschlossenen Zeilen werden von dem angegebenen UNIX-Kommando bearbeitet und durch dessen Ausgabe ersetzt.

Mode	Kommando	Erklärung
K	<code>!Gsort</code>	Sortiert alle Zeilen von der aktuellen bis zum Dateende.
K	<code>!4+ls</code>	Ersetzt die nächsten 4 Zeilen durch die Ausgabe von <code>ls</code> .
K	<code>!)lpr</code>	Druckt den nächsten Satz aus, der dann aber verschwindet.
K	<code>!!date</code>	Ersetzt die aktuelle Zeile durch das Datum.

Vorsicht:

Wer sich den Editor mit `maps` konfiguriert, sollte darauf achten, nicht versehentlich wichtige Funktionen überzudefinieren.