Universität Leipzig
Institut für Informatik
Abteilung Datenbanken
Prof. Dr. E. Rahm
T. Böhme, U. Greiner, Dr. D. Sosna

Ausgabe:10.12.2001
Abgabe:07.01.2002

Algorithmen und Datenstrukturen 1
WS 2001/2002 - Übungsblatt 5

Hinweis: Alle *Listings* müssen in der ersten Zeile nach der Klassendefinition den *Namen des Programmautors* als Kommentar enthalten. Ohne diesen Namen werden die Listings **nicht gewertet**. Ebenfalls führt das Fehlen eines geforderten Ausdrucks der Programmausführung zur Nichtbewertung der Aufgabe.

Aufgabe 1 (Externes Sortieren)

Angenommen Sie können im Hauptspeicher maximal 20 MB Daten sortieren und wollen 1 GB Daten mittels *Merge-Sort* sortieren. Ein Datensatz sei 400 Byte groß. Die Daten können mit einer Geschwindigkeit von 2 MB/sec von der Festplatte in den Hauptspeicher eingelesen bzw. vom Hauptspeicher auf die Festplatte ausgeschrieben werden. Die Dauer der internen Sortierung kann vernachlässigt werden.
Annahme: 1 GB = 1000 MB, 1 MB = 1000 KB, 1 KB = 1000 Byte.

a) Bestimmen Sie für *ausgeglichenes 2-Wege-Merge-Sort* die Anzahl der erzeugten Runs, die Anzahl der Merge-Vorgänge und die Dauer des Sortiervorgangs!

b) Bestimmen Sie für *k-Wege-Merge-Sort* die minimale Anzahl der Merge-Vorgänge und die Gesamtdauer des Sortiervorgangs!

c) Vergleichen Sie die Ergebnisse!

Aufgabe 2 (Externes Sortieren)

Gegeben sei folgende Zahlenfolge:

5, 16, 8, 3, 9, 2, 14, 10, 7, 13, 11, 12, 23, 20, 4, 1

Sortieren Sie diese von Hand mittels *Replacement Selection Sort* (Run-Länge r=4). Schreiben Sie für jeden Run die ausgegebenen Elemente und die erzeugten Auswahlbäume auf. Schreiben Sie außerdem die Ergebnisse der Mischvorgänge auf. Wieviele Runs werden erzeugt und wieviele Mischvorgänge sind nötig?
Vergleichen Sie anschließend *Replacement Selection Sort* mit *2-Wege-Merge-Sort* hinsichtlich der Anzahl der erzeugten Runs, der Run-Länge und der nötigen Mischvorgänge!

Aufgabe 3 (Fachverteilen, Queues)

Implementieren Sie in Java eine Sortierroutine auf Basis des *Fachverteilens* (Folie 4-33). Sortiert werden sollen Zeichenketten, wobei diese nach den ersten 10 Zeichen (k=10) zu sortieren sind. Anders als die Implementierung in der Vorlesung, die einen Zähler für die Fächer verwendet, sollen Sie die Objekte selbst in die Fächer verteilen. Implementieren Sie dafür eine Klasse **Queue**, die eine Warteschlange (Folie 3-25ff) realisiert und verwenden

Sie eine Instanz der Klasse Queue pro Fach.
Das Grundgerüst der Klasse **Fachverteilen** ist Ihnen auf der Web-Übungsseite gegeben. Sie brauchen nur noch die Methode

public static void sort (OrderableString[] A)

zu implementieren. Für die Sortierung sind nur die Buchstaben a-z relevant. Sie benötigen daher 27 Fächer (26 für die Buchstaben und eines für die Fehlstellen bei kurzen Zeichenketten). In dem Grundgerüst ist eine Methode *key(int i, String str)* enthalten, die für den i-ten Buchstaben der übergebenen Zeichenkette einen Schlüsselwert zwischen 0 und 26 zurückliefert. Diese Methode können Sie für die Verteilung verwenden.

a) Implementierung der Klasse *Queue* unter Verwendung einer verketteten Liste

b) Implementierung der Methode *sort (OrderableString[] A)*

c) Test der Implementierung mittels des Programms *Ueb5Test*. Dieses ermittelt für die Sortierung einer Namensliste den Zeitaufwand unter Verwendung der von Ihnen implementierten Fachverteilen-Klasse und gibt als Vergleich den Zeitaufwand für die Sortierung mittels QuickSort an. Sie benötigen dafür die Dateien *Ueb5Test.java*, *Orderable.java*, *OrderableString.java*, *SortAlgorithm.java*, *QuickSort.java* und *names_unsorted.txt* von der Web-Übungsseite.

Abgabe:
1. dokumentierte Listings der Quellprogramme (Queue.java, Fachverteilen.java)
2. Ausgabe eines Programmlaufs von Ueb5Test

Aufgabe 4 (Münzproblem - optional)

a) Schreiben Sie ein Programm, das die minimale Anzahl von Münzen berechnet, mit denen ein bestimmter Eurobetrag in Münzen ausgegeben werden kann. Verwendbare Münzen sind 1 Cent, 2 Cent, 5 Cent, 10 Cent, 20 Cent, 50 Cent, 1 Euro (= 100 Cent) und 2 Euro (= 200 Cent). Ausgabe ist die minimale Anzahl.

(Beispiel: 5 Euro lassen sich mit zwei 2- und einem 1-Eurostück mit der minimalen Anzahl an Münzen auszahlen.)

b) Schreiben Sie ein Programm, das berechnet wieviele Möglichkeiten es gibt einen bestimmten Euro-Betrag in Münzen auszuzahlen. Verwendbare Münzen sind 1 Cent, 2 Cent, 5 Cent, 10 Cent, 20 Cent, 50 Cent, 1 Euro (= 100 Cent) und 2 Euro (= 200 Cent). Lösen Sie das Problem mit einem rekursiven Algorithmus. Ausgegeben wird die Anzahl der Möglichkeiten.

(Beispiel: 5 Cent lassen sich auf vier verschiedene Arten auszahlen: 5 Cent, 2+2+1 Cent, 2+1+1+1 Cent, 1+1+1+1+1 Cent.)

Testen Sie Ihre Programme jeweils mit den folgenden Werten: 15, 47, 84 Cent, 2, 7, 10 Euro.
Abgabe:
1. Dokumentiertes Listing des Quellcodes.
2. Für jeden Eingabewert die Anzahl der Möglichkeiten.