OPERATIONS RESEARCH
KACKERTSTR. 7
52072 AACHEN

Prof. Dr. Marco Lübbecke
Dipl.-Math. oec. Michael Bastubbe
Christian Puchert, M.Sc.
Annika Thome, M.Sc.

Johannes Gocke
Dustin Hütter
Sascha Kurowski

Programmieren, Algorithmen, Datenstrukturen
7. Programmieraufgabe

Abnahme bis spätestens 20.12.2013

In dieser Aufgabe sollt ihr mehrere Sortieralgorithmen implementieren. Alle Vergleiche sollen mittels des Java-Interfaces `Comparator` durchgeführt werden. Ein Sortieralgorithmus, der durchgehend `Comparator` verwendet, kann dann für beliebige Daten genutzt werden. Das Interface `Comparator` ist ein fester Teil von Java und liegt im Paket `java.util`. Unter `http://download.oracle.com/javase/7/docs/api/java/util/Comparator.html` könnt ihr die Beschreibung dazu finden. `Comparator<T>` ist generisch und vergleicht Objekte vom Typ `T` in der Methode `int compare(T o1, T o2)`. Diese gibt einen Wert < 0 zurück, falls $o1$ vor $o2$ kommt, > 0, falls $o1$ nach $o2$ kommt, und $= 0$, falls beide gleichberechtigt sind in der Sortierreihenfolge. Dies passiert üblichweise nur, wenn `o1.equals(o2)`. Alle Vergleiche in dieser Aufgabe sollen mittels Klassen gemacht werden, die `Comparator` implementieren. **Anmerkung:** Die Schnittstelle `Comparator<T>` definiert auch eine Methode `equals`. Diese bezieht sich aber auf den `Comparator` selbst und nicht auf die zu sortierenden Objekte. Man kann damit überprüfen, ob zwei Objekte vom Typ `Comparator` die gleiche Reihenfolge produzieren. Wenn ihr diese Methode nicht schreibt, wird sie nur für identische `Comparator`-Objekte `true` ergeben – das ist für diese Aufgabe so in Ordnung.

(a) Schreibt eine Klasse `ComparatorSize implements Comparator<Integer>`, die ganze Zahlen aufsteigend vergleicht.

(b) Schreibt ein Interface `SortAlgorithm<T>`, das eine Methode `public void sort(T[] data, Comparator<T> comp)` definiert. Dieses Interface sollen alle Sortieralgorithmen implementieren und dann `data` laut `comp` sortieren.

(c) Schreibt eine generische Klasse `SelectionSort<T> implements SortAlgorithm<T>` zum Sortieren mittels Selectionsort. Ab jetzt habt ihr alles zur Verfügung um eure Algorithmen zu testen.

(d) Schreibt den Comparator `ComparatorDigitSum implements Comparator<Integer>`, der ganze Zahlen aufsteigend nach ihrer Quersumme vergleicht. Bei Gleichheit gilt die kleinere Zahl als kleiner. Negative Zahlen haben die negative Quersumme ihres Betrags.

(e) Schreibt den Comparator `ComparatorInverse<T> implements Comparator<T>`, der im Konstruktor einen anderen Comparator entgegen nimmt und die Elemente umgekehrt anordnet.

(f) Schreibt den Comparator `CountingComparator<T> implements Comparator<T>`, der im Konstruktor einen anderen Comparator entgegennimmt, wie dieser sortiert, aber mitzählt, wie oft `compare` aufgerufen wird. Gebt ihm eine öffentliche Methode um den Zählerstand abzufragen.

(g) Implementiert Mergesort in einer generischen Klasse.

(h) Implementiert Quicksort in einer generischen Klasse und achtet dabei darauf, dass ihr bis auf einzelne Variablen keine neuen Datenstrukturen dafür anlegt.

(i) Schreibt den Comparator `ComparatorDenominator implements Comparator<Double>`, der `Double`-Werte anhand ihres (kleinsten positiven) Nenners in Bruchschreibweise anordnet. Dabei soll eine `Double`-Zahl x zu der rationalen Zahl umgewandelt werden, die den kleinsten Nenner im Bereich $(x-\varepsilon, x+\varepsilon)$ hat. Dieses ε erhält der Comparator im Konstruktor. Verwendet als Standardwert $\varepsilon = 0,01$ im Rest der Aufgabe. Bei Gleichheit wird die kleinere Zahl nach vorne sortiert.

(j) Schreibt ein Hauptprogramm, welches ein Array mit n zufälligen Werten (für `Integer` von -1000 bis 1000, für `Double` zwischen -10.0 und 10.0) füllt und dann einen Sortieralgorithmus mit einem der Comparator darauf ausführt. Am Ende soll das sortierte Array und die Anzahl der ausgeführten Vergleiche ausgegeben werden.

Das Programm soll per Kommandozeile gesteuert werden. Die Parameter werden dabei der `public static void main(String[] args)` im Array `args` als Strings übergeben. So ergibt der Aufruf

```
java Start 100 merge up digitsum
```

das Array

```
args = {"100", "merge", "up", "digitsum"}.
```

Als Parameter erwartet das Programm die Anzahl der zu erzeugenden Zufallszahlen, den Sortieralgorithmus ("select", "merge", "quick"), die Sortierrichtung ("up" oder "down") und welcher Comparator ("size", "digitsum", "denominator") verwendet werden soll. Der obige Aufruf sortiert also 100 zufällige Zahlen mit Mergesort aufsteigend anhand der Quersumme. Der Comparator bestimmt den Typ der zu sortierenden Daten, bei "size" und "digitsum" sollen `Integer`, bei "denominator" `Double` sortiert werden. Gebt im sortierten Double-Array auch den passenden Nenner zu jeder Zahl mit aus, damit man das Ergebnis nachvollziehen kann. Das Programm soll sich bei ungültigen Parametern mit einer kurzen Fehlermeldung per `return` beenden. Die Reihenfolge der Parameter könnt ihr als fest vorgegeben ansehen.

Viel Spaß und Erfolg!