HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

ÜBUNGSSERIE 5

Algorithmen & Datenstrukturen AD2 / HS 2018

AD2 Team

Aufgabe 1 (Programmierung – Implementation Bubble-Sort)

Neben den Sortieralgorithmen, welche Ihnen in der Vorlesung vorgestellt wurden, gibt es noch eine Vielzahl weiterer. Ein bekannter Algorithmus ist der so genannte *Bubble-Sort*. Er verdankt seinen Namen der Analogie zu Luftblasen, welche im Wasser empor „bubblen". Dabei steigen die grösseren Blasen schneller als die Kleineren.

a) Implementieren Sie den Algorithmus. Er ist denkbar einfach. In jeder Iteration beginnt man mit dem ersten Element und vergleicht es mit dem Zweiten. Ist das erste Element grösser, so werden die beiden vertauscht. Daraufhin wird das selbe mit dem zweiten und dem dritten Element gemacht, etc.. Der Algorithmus wird so lange ausgeführt, bis keine Vertauschungen mehr vorgenommen werden.
b) Es gibt viele Verbesserungen an dem Verfahren. Eine simple Optimierung ist die obere Grenze bei jeder Iteration zu dekrementieren. Dies ist möglich, weil sich nach jeder Iteration bei aufsteigender Sortierung die gegenwärtig grösste Blase am Ende der Sequenz befindet. Messen Sie die Auswirkung, welche dieser simple Eingriff auf das Laufzeitverhalten hat.

Aufgabe 2 (Programmierung – Implementation Merge-Sort)

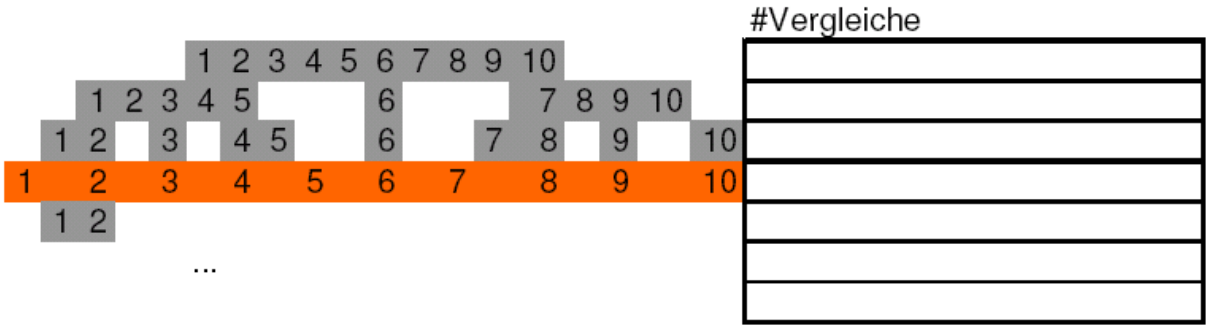
Es soll der *Merge-Sort* Algorithmus gemäss Skript implementiert werden.
Wir setzten voraus, dass nur Sequenzen der Längen von ganzen 2er-Potenzen (2^n) sortiert weren müssen.
Eine Ausgangslage befindet sich wiederum auf dem Skripte-Server.
Es müssen darin nur die Methode `mergeSort()` und `merge()` erweitert werden.
Wir verwenden dazu ebenfalls nur *int-Arrays* (keine Listen für die Sequenzen).
Dazu benutzen wir u.a. `System.arraycopy()` um neue Arrays aus bestehenden zu erzeugen und führen zusätzliche Hilfs-Indexes ein um eine Sequenz zu simulieren (siehe `ai, bi, si` in `merge()`).
Mit dem JUnit-Test *MergeSortJUnitTest* soll die Implementation getestet werden.
Nach der Implementierung sollen die gemessenen Laufzeiten beobachtet werden.
Die Java-Virtual-Machine soll dabei mit den Parameter gemäss Session-Log zur Ausführung gebracht werden (`-Xint -Xms100M -Xmx100M`).
Entsprechen die Laufzeiten den Erwartungen?
Vergleichen Sie diese auch mit der Bubble-Sort-Implementation der vorherigen Aufgabe.

Aufgabe 3 (Quick-Sort)

Wieviele Vergleiche brauchen Sie zum Sortieren der Folge {1,2,3,4,5,6,7,8,9,10} mit Quicksort.

a)

1. falls diese in geordneter Reihenfolge vorliegt.



2. wählen sie als Pivot das letzte Element.

b) Wie empfindlich ist der Algorithmus in Bezug auf die Auswahl des Pivot-Elements? Wählen Sie zum Vergleich:
   1. ein Element möglichst am Anfang als Pivot.
   2. den Median als Pivot.
   3. das Pivot-Element wird als Median der letzten d Elemente (d >=3 und ungerade) gewählt. Argumentieren Sie, warum dies eine gute Pivot-Auswahl sein könnte.

c) Ist der inplace QuickSort Algorithmus *stabil* ? Begründung?
Stabilität (bei der Sortierung): Die Reihenfolge gleicher Elemente bleibt mit der Sortierung erhalten.