MUSTERLÖSUNGEN ÜBUNGSSERIE 5

Algorithmen & Datenstrukturen AlgDat / HS 2024

AlgDat 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.

   Siehe „BubbleSort.java"

Aufgabe 2 (Programmierung – Implementation Merge-Sort)

Es soll der *Merge-Sort* Algorithmus gemäss Skript implementiert werden.
Wir setzten voraus, dass einfachheitshalber nur Sequenzen der Längen von ganzen 2er-Potenzen (2^n) sortiert weren müssen.
Eine Ausgangslage befindet sich wiederum auf Moodle.
Es müssen darin nur die Methode `mergeSort()` und `merge()` erweitert werden.
Wir verwenden dazu ebenfalls nur *int-Arrays* (keine Listen für die Sequenzen).
Dazu 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.

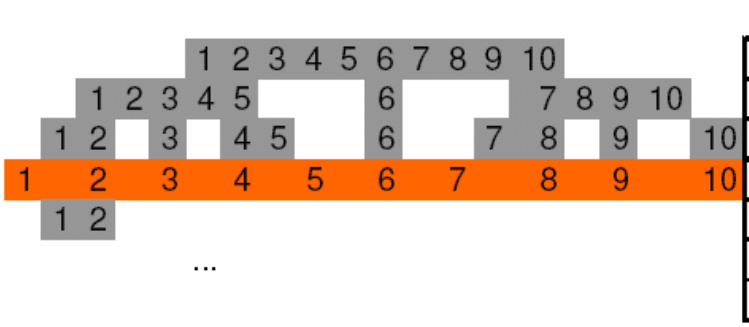
Siehe „MergeSort.java"

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.
   Pivot-Index = Sequenz-Length / 2



#Vergleiche
9
7
3
0
-
-
-

$$h = \log n$$
$$O(n \log n)$$

2. Falls als Pivot das letzte Element gewählt wird.

$$h = n - 1$$
$$O(n^2)$$

$$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = 45 \text{ Vergleiche total}$$

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.
   Hängt von der Vorsortierung ab. Laufzeit hängt von dem Input ab -> schlecht.
2. den Median als Pivot.
   Optimale Wahl, da die Sequenz in zwei annähernd gleich grosse Teile geteilt wird. Laufzeit ist dann optimal in $O(n \log n)$. Allerdings ist die Determinierung des Medianes mit z.T. erheblichem Aufwand verbunden.
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.
   Es handelt sich um eine Stichprobe der gesamten Menge. Wenn d gross genug ist um eine repräsentative Erhebung zu ermöglichen, so kann vom Median der Stichprobe auf den Median der gesamten Sequenz geschlossen werden. Siehe Aufgabe 1.c.2 für die Auswirkungen. Im Gegensatz zu Aufgabe 1.c.2 muss der Median aber nur für einen kleinen Teil gefunden werden, was Zeit spart.

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

Der inplace QuickSort Algorithmus ist nicht stabil. Wir wollen diese Behauptung mit einem Beispiel belegen. Gegeben sei folgende Sequenz: h_i, h_j, p, l_k, l_l wobei p das Pivot ist und $l_k = l_l < p < h_i = h_j$.
Inplace QuickSort würde nun die Elemente mit den Indizes i und l sowie j und k vertauschen, was zur Sequenz l_l, l_k, p, h_j, h_i führt. Offensichtlich sind die Elemente in ihrer Reihenfolge vertauscht worden. Folglich ist der inplace QuickSort nicht stabil.