Algorithmen und Datenstrukturen

Dr. M. Lüthi, Dr. G. Röger
Frühjahrssemester 2018

Universität Basel
Fachbereich Informatik

Übungsblatt 3

Abgabe: 23. März 2018

Aufgabe 3.1 (Quicksort, 2+2+2 Punkte)

Betrachten Sie Sequenz $a = \langle 5, 1, 4, 5, 0, 3, 2, 7 \rangle$.

(a) Simulieren Sie den Aufruf von `partition(a, 0, 7)` und geben Sie jeweils den Stand des Arrays und der i- und j-Pointer nach Zeile 4, 9 und 14 an. Was ist der Rückgabewert und wie sieht die Sequenz zum Terminierungszeitpunkt aus?

(b) Welche Eigenschaft sollte das Pivotelement allgemein günstigenfalls haben? Wählen Sie im folgenden das Pivotelement jeweils optimal, so dass die Anzahl der Aufrufe von `sort_aux` minimiert wird. Geben Sie die Aufrufe von `sort_aux` (jeweils mit dem Zustand des Arrays) an, wie sie in einem Durchlauf von `sort(a)` geschehen.

Geben Sie jeweils an, welches Pivotelement gewählt wird. Bei gleich gut geeigneten Pivotelementen wählen Sie bitte immer die kleinere Zahl und bei gleichen Zahlen, das Element, das gerade weiter vorne in der Sequenz steht.

Hinweis: Vergessen Sie bitte nicht, dass das Pivotelement in Zeile 8 des Algorithmus an Position lo geswapt wird. Vergessen Sie auch nicht die rekursiven Aufrufe für leere und einelementige Bereiche. Wenn Sie alles richtig machen, sollten Sie insgesamt neun Aufrufe von `sort_aux` benötigen.

(c) Was ist allgemein die schlechtestmögliche Wahl für das Pivotelement? Geben Sie die ersten sieben Aufrufe von `sort_aux` für den Fall an, dass immer ein möglichst ungünstiges Pivotelement gewählt wird. Bei mehreren Kandidaten wählen Sie bitte wieder das erste.

Erkennen Sie ein Muster? Wie viele Aufrufe wird Quicksort in diesem Fall insgesamt benötigen?

Aufgabe 3.2 (Radixsort, 1 + 3 Punkte)

Im Rahmen der Aufgabe sollen Sie Radixsort implementieren. Als Grundlage finden Sie im ADAM-Workspace der Vorlesung die Datei `blatt03.zip`.

- Erweitern Sie `src/test/java/SortTests.java` um mindestens drei weitere Testcases für Radixsort. Die Tests sollen das Verfahren für verschiedene Basen testen und auch das Betrachten von mehr als nur einer Stelle erfordern.
- Vervollständigen Sie Radixsort in der Datei `src/main/java/RadixSort.java`. Verwenden Sie dabei jeweils eine `ArrayList<Integer>` für die einzelnen Buckets.

  Stellen Sie sicher, dass ihre Lösung die Testcases und den Stylecheck (vgl. Blatt 1) besteht.

Die Übungsblätter dürfen in Gruppen von zwei Studierenden bearbeitet werden. Bitte schreiben Sie beide Namen auf Ihre Lösung.

1a)

```
1  def partition(array, lo, hi):
2      pivot = array[lo]
3      i = lo + 1
4      j = hi
5      while (True):
6          while i < hi and array[i] < pivot:
7              i += 1
8          while array[j] > pivot:
9              j -= 1
10         if i >= j:
11             break
12
13         array[i], array[j] = array[j], array[i]
14         i, j = i + 1, j - 1
15     array[0], array[j] = array[j], array[0]
16     return j
```

[5, 1, 4, 5, 0, 3, 2, 7]

[5, 1, 4, 2, 0, 3, 5, 7]

[3, 1, 4, 2, 0, 5, 5, 7]