Lösung 04: Quicksort, Mergesort

Aufgabe 1: Quicksort

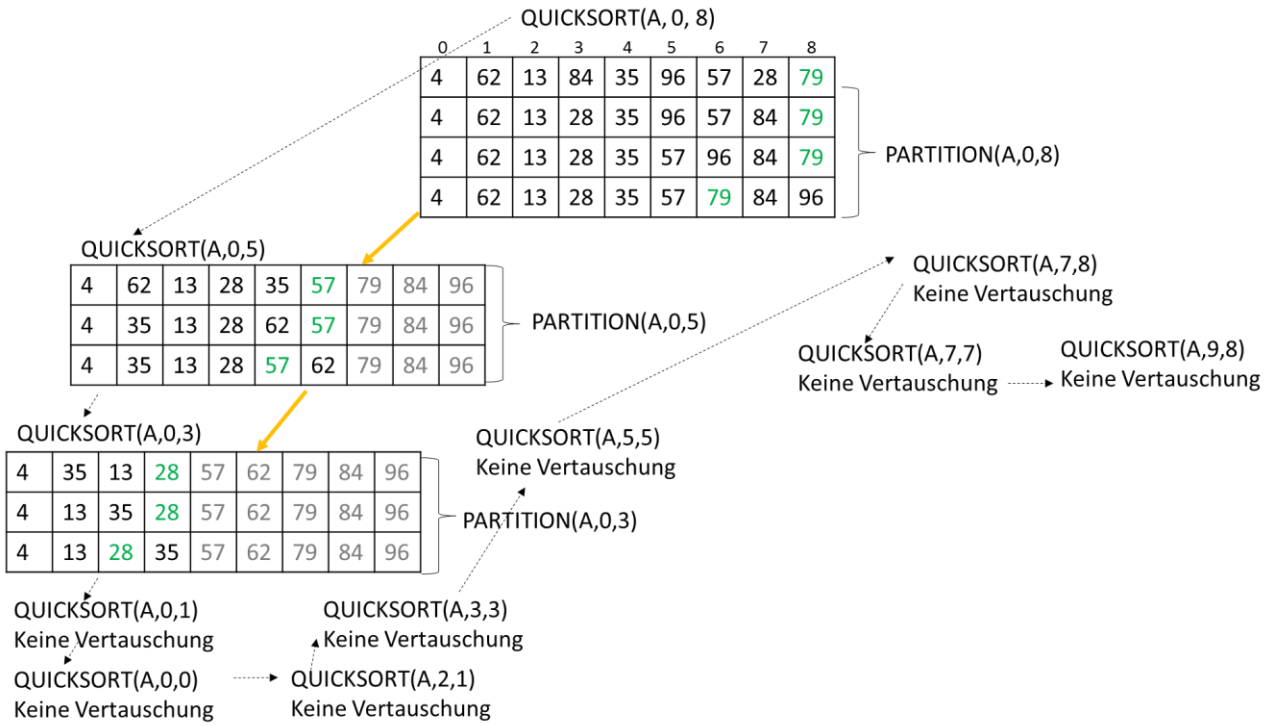
a) Die folgende Tabelle zeigt das Array nach jeder Vertauschung. Die grüne Zahl ist das Pivot, die blauen Zahlen markieren den *i*- und den *j*-Index. Wie man erkennt, sind insgesamt 3 Vertauschungen notwendig (am Schluss muss das Pivot an die richtige Stelle gesetzt werden).

4	62	13	84	35	96	57	28	79
4	62	13	28	35	96	57	84	79
4	62	13	28	35	57	96	84	79
4	62	13	28	35	57	79	84	96

b) Es sind insgesamt 7 Vertauschungen notwendig. 3 davon beim Aufruf von PARTITION(A,0,8), 2 beim Aufruf von PARTITION(A,0,5) und 2 beim Aufruf von PARTITION(A,0,3). Die Belegungen verändern sich wie folgt:

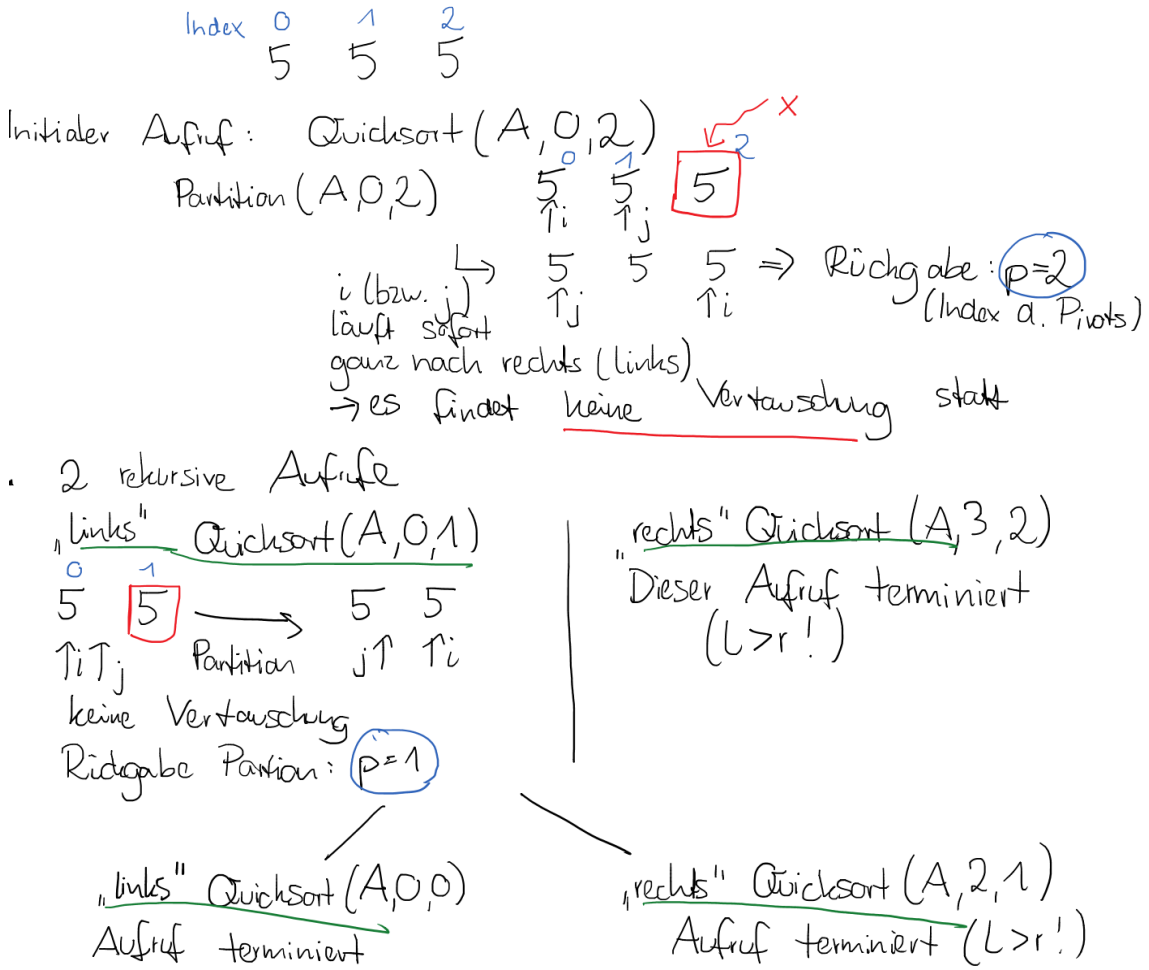
- 4, 62, 13, 84, 35, 96, 57, 28, 79
- 4, 62, 13, 28, 35, 96, 57, 84, 79
- 4, 62, 13, 28, 35, 57, 96, 84, 79
- 4, 62, 13, 28, 35, 57, 79, 84, 96
- 4, 35, 13, 28, 62, 57, 79, 84, 96
- 4, 35, 13, 28, 57, 62, 79, 84, 96
- 4, 13, 35, 28, 57, 62, 79, 84, 96
- 4, 13, 28, 35, 57, 62, 79, 84, 96

Die folgende Skizze (nicht verlangt) zeigt in welcher Reihenfolge diese 7 Vertauschungen stattfinden und wie das Array jedes Mal aussieht. Die gestrichelten schwarzen Pfeile illustrieren in welcher Reihenfolge die rekursiven Aufrufe von Quicksort auftreten.



c) Falls alle Elemente der Eingabe `A[l..r]` den gleichen Wert haben, wird `PARTITION` keine einzige Vertauschung vornehmen. Das Pivot liegt nach `PARTITION` am rechten Rand des Arrays A[r]. Deshalb wird im Quicksort rekursiv auf `A[l..(r-1)]` aufgerufen, die Eingabegröße wird also nur um 1 kleiner. Im nächsten Rekursionsschritt nimmt `PARTITION` aber wieder keine Vertauschung vor, usw. Insgesamt gibt es **keine einzige Vertauschung** statt, aber rekursive Aufrufe.

- Es wird nie ein Element getauscht.
- Allerdings gibt es 4 rekursive Aufrufe (siehe grüne Markierung). Ausführliche Illustration, siehe Skizze.



d) Man könnte meinen, dass das Array dann bereits sortiert ist. Das ist aber nicht immer der Fall. Gegenbeispiel:

- Array 1 2 1 2 2
- Als Pivot wird das rechte 2 gewählt.
- Nach dem Partitionieren sieht das Array immer noch gleich aus: 1 2 1 2 2 (der i-Zeiger steht bei 2).
- Rekursion würde dann auf 1 2 1 2 erfolgen. Hier erfolgt auch die erste Vertauschung.

Aufgabe 2: Iterativer MergeSort mit Queues

a) Siehe Lösung im Gitlab: src/de/th_rosenheim/ad/uebung04/
b) Siehe Lösung im Gitlab: src/de/th_rosenheim/ad/uebung04/
c) Die Laufzeit ist wie beim (normalen) iterativen MergeSort und wie beim rekursiven MergeSort O(n log n). Die Verwendung von Queues stellt sicher, dass zunächst alle 1-elementige Queues/Listen zu 2-elementige Queues/Listen gemischt werden. Erst wenn keine Einzelqueue weniger als 2 Elemente enthält, kommen die bereits „gemischten" 2-elementigen Queues an den Head der zentralen Queue.