Algorithmus

Die Idee des Quicksort ist ähnlich wie die des Merge Sort. Die Liste wird rekursiv in immer kleinere Teile zerlegt. Der Quicksort gehört damit auch zu den Divide-and-Conquer-Algorithmen. Während beim Merge Sort auf dem **Rückweg** die geteilten Listen sortiert wieder zusammengeführt werden, wird die Liste beim Quicksort jeweils **vor dem Teilen** umsortiert.

1. Wir gehen aus von einer unsortierten Liste von Zahlen, die als Array implementiert ist:

10	5	8	4	1	7	3	9

2. Der Quicksort wählt ein Element des Arrays aus. Dieses Element nennt man „**Pivot**“-Element. Zur Frage, welches Element als Pivot ausgewählt wird, gibt es verschiedene Strategien. Man kann z.B. das erste, das mittlere oder ein zufälliges Element auswählen.
   Wir nehmen an dieser Stelle das Element in der **Mitte**.

10	5	8	4	1	7	3	9

3. Das Array wird **umsortiert**, so dass alle Elemente, die kleiner als das Pivot-Element sind, auf die linke Seite, und alle, die größer sind, auf die rechte Seite kommen. Für das Pivot-Element selbst (oder andere Elemente, die den gleichen Wert haben) ist es prinzipiell egal, auf welcher Seite sie stehen.

   Eine Möglichkeit, das Array umzusortieren ist, von links nach Elementen zu suchen, die größer oder gleich, und von rechts nach Elementen, die kleiner oder gleich dem Pivot-Element sind. Sobald man links und rechts je eins gefunden hat, werden sie getauscht.
   In dieser Variante tauscht also ggf. auch das Pivot-Element seinen Platz.
   Das wird solange wiederholt, bis man sich in der Mitte trifft.

   Im Beispiel würden zuerst links die 10 und rechts die 3 getauscht. Danach links 5 und rechts 1, und zuletzt links 8 und rechts 4 :

10	5	8	4	1	7	3	9

3	1	4	8	5	7	10	9

4. Die Suchvorgänge von links bzw. rechts haben sich zwischen 4 und 8 getroffen.
   An dieser Stelle wird das Array geteilt:

3	1	4

8	5	7	10	9

5. Dann wird der Quicksort **rekursiv für beide Teile** aufgerufen. Für jedes Teil-Array wird ein neues Pivot-Element gewählt, die kleineren Elemente werden nach links und die größeren nach rechts getauscht, das Array wird geteilt, und so weiter.

3	1	4

8	5	7	10	9

Implementierung des Quicksort

Die folgende Implementierung sieht recht übersichtlich aus. Den Quicksort korrekt zu implementieren ist allerdings eine Kunst, da schon minimale Änderungen dazu führen können, dass er nicht korrekt sortiert oder nicht „terminiert“ (also sich in einer endlosen Rekursion verliert).

```
public void quickSort(int[] array, int left, int right)
{
   if (left < right)
   {
      // wähle Element in der Mitte als Pivot
      int pivot = array[(left+right)/2];

      int i = left, j = right;
      while (i < j)
      {
         // Suche Element von links, das >= Pivot ist
         while (array[i] < pivot)
         {
            i++;
         }
         // Suche Element von rechts, das <= als Pivot ist
         while (array[j] > pivot)
         {
            j--;
         }
         // Wenn i links von j liegt, tausche die Elemente
         if (i < j)
         {
            tausche(array, i, j);
            j--;
         }
      }
      quickSort(array, left, j);
      quickSort(array, j+1, right);
   }
}
```

Aufgabe

Analysiere den Aufruf der Quicksort-Methode mit dem folgenden Beispiel für die erste, zweite und dritte Rekursionstiefe. Protokolliere dabei die Werte der Parameter und Variablen und die Änderungen des Arrays durch die while-Schleife.

array

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
2	7	11	19	5	18	21	28	6	3

Tiefe 1

left	right	pivot

array

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

Tiefe 2 (links)

left	right	pivot

Tiefe 2 (rechts)

left	right	pivot

array

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*

Tiefe 3 (li-li)

left	right	pivot

Tiefe 3 (li-re)

left	right	pivot

Tiefe 3 (re-li)

left	right	pivot

Tiefe 3 (re-re)

left	right	pivot

array

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*