QuickSort: Sortieren durch Zerlegen

Weiterführende Literatur:

- *Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19*, Seite 55
- *Wikipedia-Artikel „Quicksort“*
- Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 135-139 (PDF 153-157)

Funktionsweise

Listen mit maximal *einem* Element sind *trivialerweise sortiert*. Falls die zu sortierende Liste mehr als ein Element beinhaltet wird ein sogenanntes *Pivot-Element* (vom Französischen *pivot „Dreh-/Angelpunkt“*) ausgewählt. Alle *kleineren* Elemente werden *vor* und alle *größeren hinter* das Pivot-Element verschoben. Der Algorithmus verfährt *rekursiv* mit den beiden Teillisten. Der Algorithmus arbeitet nach dem *Teile-Und-Herrsche-Prinzip*.

einem
trivialerweise sortiert
Pivot-Element
Dreh-/Angelpunkt
kleineren
vor
größeren
hinter
rekursiv
Teile-Und-Herrsche-Prinzip

Eigenschaften

- Laufzeitkomplexität:
  - $\mathcal{O}(n \cdot log(n))$ (im Best-/Average-Case)
  - $\mathcal{O}(n^2)$ (im Worst-Case)
- in „klassischer“ Variante *instabil*
- durch Rekursion wachsender Aufrufstapel $\rightarrow$ out-of-place

Minimales Code-Beispiel zum Auswendiglernen

```
private int zerlege(int l, int r) {
  int i, j;
  int pw = a[(l + r) / 2];
  i = l - 1;
  j = r + 1;
  while (true) {
    do {
      i++;
    } while (a[i] < pw);

    do {
      j--;
    } while (a[j] > pw);

    if (i < j) {
      vertausche(i, j);
    } else {
      return j;
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/sortier/QuickMinimal.java`

```
private int[] sortiereRekursiv(int l, int r) {
  int p;
  if (l < r) {
    p = zerlege(l, r);
    sortiereRekursiv(l, p);
    sortiereRekursiv(p + 1, r);
  }
  return zahlen;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/sortier/QuickMinimal.java`

Implementation nach Saake[1]

```
/**
 * Hilfsmethode zum Zerlegen der Folge. Diese Methode heißt im
↪  Englischen auch
 * oft „partition".
 *
```

[1] Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 138 (PDF 156).

```
 * @param links  Die Index-Nummer der unteren Grenze.
 * @param rechts Die Index-Nummer der oberen Grenze.
 *
 * @return Die endgültige Index-Nummer des Pivot-Elements.
 */
private int zerlege(int links, int rechts) {
  berichte.feldAusschnitt(links, rechts, "zerlege");
  int pivotIndex = bestimmePivot(links, rechts);

  int pivotWert = zahlen[pivotIndex];
  int pivotIndexEndgültig = links;
  // Pivot-Element an das Ende verschieben
  if (pivotIndex != rechts) {
    vertausche(links, rechts, pivotIndex, rechts);
  }
  for (int i = links; i < rechts; i++) {
    if (zahlen[i] <= pivotWert) {
      vertausche(links, rechts, pivotIndexEndgültig, i);
      pivotIndexEndgültig++;
    }
  }
  // Pivot-Element an die richtige Position kopieren
  vertausche(links, rechts, rechts, pivotIndexEndgültig);
  // neue Pivot-Position zurückgeben
  return pivotIndexEndgültig;
}

/**
 * Hilfsmethode zum rekursiven Sortieren
 *
 * @param links  Die Index-Nummer der unteren Grenze.
 * @param rechts Die Index-Nummer der oberen Grenze.
 */
private void sortiereRekursiv(int links, int rechts) {
  if (rechts > links) {
    // Feld zerlegen
    int pivotIndexEndgültig = zerlege(links, rechts);
    // und zerlegeen sortieren
    sortiereRekursiv(links, pivotIndexEndgültig - 1);
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/sortier/QuickSaake.java`

Weitere Implementation

```
 * verschoben wird.
 */
public class QuickHorare extends Quick {

  /**
   * Zerlege das Zahlen-Feld.
   *
   * @param links  Die Index-Nummer ab dem das Zahlen-Feld zerlegt werden
↪  soll.
   * @param rechts Die Index-Nummer bis zu dem das Zahlen-Feld zerlegt
↪  werden
   *               soll.
   *
   * @return Die Index-Nummer, an dem das Feld zerlegt werden soll.
   */
  private int zerlege(int links, int rechts) {
    berichte.feldAusschnitt(links, rechts, "zerlege");
    int i, j;
    int pivotIndex = bestimmePivot(links, rechts);
    int pivotWert = zahlen[pivotIndex];
    i = links - 1;
    j = rechts + 1;
    while (true) {
      do {
        i++;
      } while (zahlen[i] < pivotWert);

      do {
        j--;
      } while (zahlen[j] > pivotWert);

      if (i < j) {
        vertausche(links, rechts, i, j);
      } else {
        return j;
      }
    }
  }

  /**
   * Sortiere ein Zahlen-Feld mit Hilfe des Quicksort-Algorithmus.
   *
   * @param links  Die Index-Nummer ab dem das Zahlen-Feld sortiert werden
↪  soll.
   * @param rechts Die Index-Nummer bis zu dem das Zahlen-Feld sortiert
↪  werden
```

```
 *                soll.
 *
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/sortier/QuickHorare.java`

Im Gegensatz zu der Implementation von Saake wird hier der Pivot-Wert nicht an den oberen Rand und dann wieder zurück kopiert.

Literatur

[1] *Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19*. `https://www.studon.fau.de/file2567217_download.html`. FAU: Lehrstuhl für Informatik 2 (Programmiersysteme).

[2] Gunter Saake und Kai-Uwe Sattler. *Algorithmen und Datenstrukturen. Eine Einführung in Java*. 2014.

[3] *Wikipedia-Artikel „Quicksort“*. `https://de.wikipedia.org/wiki/Quicksort`.