Rekursion - Teil VI - Quicksort

Ein weiterer nennenswerter Algorithmus, der auf Rekursion beruht, ist Quicksort. Wir haben bereits einmal in einer Simulation gesehen, dass Quicksort deutlich schneller sortiert als die drei bisher bekannten Sortierverfahren.

Den Vergleich einiger Sortierverfahren findet man z. B. auf den Seiten des Mathe-Prisma-Projekts:

http://www.matheprisma.uni-wuppertal.de/Module/Sortieren/index.htm

Divide and Conquer

Quicksort gehört zur Klasse der "divide and conquer"-Algorithmen. Dabei bedeutet "divide and conquer" soviel wie "teile und herrsche".

Bei einem „teile und herrsche“-Ansatz wird das eigentliche Problem so lange in kleinere und einfachere Teilprobleme zerlegt, bis man diese lösen („beherrschen“) kann. Anschließend wird aus diesen Teillösungen eine Lösung für das Gesamtproblem (re-)konstruiert.

Strategie von Quicksort:

Wähle in der zu sortierenden Folge das mittlere Element (als Pivot-Element) aus und verschiebe alle Elemente, die kleiner sind als dieses mittlere Element, nach links und die, die größer sind, nach rechts. Anschließend wird der gleiche Algorithmus auf die Teilfolge links angewandt und anschließend auf die Teilfolge rechts. Wenn eine Teilfolge nur noch aus einem oder aus keinem Element mehr besteht, endet die Rekursion.

Interessanterweise kann man auch statt des mittleren Elements ein beliebiges anderes Element als Pivot-Element wählen. Eine gute Animation (allerdings wird dort das erste Element als Pivot-Element gewählt) findet man unter:

http://www.cise.ufl.edu/~ddd/cis3020/summer-97/lectures/lec17/sld003.htm

Eine ebenfalls gelungene Animation findet man auf der Seite des Mathe-Prisma-Projekts:

http://www.matheprisma.uni-wuppertal.de/Module/Sortieren/index.htm

Ganz witzig sind die beiden folgenden Animationen:

Bubblesort vs. Quicksort als Animationsfilm: http://www.youtube.com/watch?v=vxENKlcs2Tw

Quicksort als ungarischer Folkstanz: http://www.youtube.com/watch?v=ywWBy6J5gz8

Zu bemerken bleibt noch, dass der Aufwand der einfachen Sortierverfahren im average case $O(n^2)$ ist, während Quicksort im average case $O(n \log(n))$ hat.

Es lässt sich übrigens beweisen, dass ein vergleichsbasiertes Sortierverfahren nicht schneller als $O(n \log(n))$ sein kann.

Unter http://de.wikipedia.org/wiki/Sortierverfahren findet man einen Vergleich der Verfahren. Für die von uns betrachteten Sortierverfahren gilt:

Vergleich der Sortierverfahren	BC	AV	WC
Sortieren durch Auswahl	$O(n^2)$	$O(n^2)$	$O(n^2)$
Sortieren durch Einfügen	$O(n^2)$	$O(n^2)$	$O(n^2)$
Sortieren durch Vertauschen	$O(n)$	$O(n^2)$	$O(n^2)$
Quicksort	$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$

Programm Quicksort: (mittleres Element ist das Pivot-Element)

```
public static void quicksort(int[] a, int linkeGrenze, int rechteGrenze) {

  int lo = linkeGrenze;
  int hi = rechteGrenze;
  int mitte;              // Stelle des mittleren Elements
  int pivot;              // Wert des mittleren Elements

  if (rechteGrenze > linkeGrenze) {

    // ein beliebiges Pivot-Element aussuchen
    mitte = (linkeGrenze + rechteGrenze) / 2;
    pivot = a[mitte];

    // wiederholen, bis lo und hi sich kreuzen
    while (lo <= hi) {

      // finde ein Element, was größer oder gleich dem Pivot-Element ist
      while ((lo < rechteGrenze) && (a[lo] < pivot))
        lo = lo+1;

      // finde ein Element, was kleiner oder gleich dem Pivot-Element ist
      while ((hi > linkeGrenze) && (a[hi] > pivot))
        hi=hi-1;

      // wenn sich die Zeiger noch nicht überkreuzt habe, tauschen
      if (lo <= hi) {
        int dummy;
        dummy = a[hi];
        a[hi] = a[lo];
        a[lo] = dummy;

        // beide Zeiger um eins weiterrücken
        lo=lo+1;
        hi=hi-1;
      }
    }
    // wenn es links vom Pivot-Element noch Elemente gibt, Rekursion
    if (linkeGrenze < hi)
      quicksort(a, linkeGrenze, hi);

    // wenn es rechts vom Pivot-Element noch Elemente gibt, Rekursion
    if (lo < rechteGrenze)
      quicksort(a, lo, rechteGrenze);
  }
}
```