

Allgemeines

Quicksort wurde 1960 von C.A.R. HOARE erfunden und gilt auch heute noch für die meisten Fälle als schnellstes internes Sortierverfahren (für das Sortieren von Daten in externen Dateien benutzt man andere Algorithmen).

Quicksort stützt sich auf die Tatsache, dass Austauschen vorzugsweise über größere Entfernungen (im Gegensatz zu Bubblesort) ausgeführt wird und dann am effizientesten ist. Sind z.B. n Elemente in umgekehrter Reihenfolge gegeben, so kann man sie mit nur $\frac{n}{2}$ Austauschvorgängen sortieren.

Idee zum Algorithmus

Man wähle willkürlich ein Element (das *Pivot-Element*) und nenne es x , durchsuche dann das Array von links, bis ein Element $a_i > x$ gefunden wird, und von rechts, bis ein Element $a_j < x$ gefunden wird. Nun werden diese beiden Elemente vertauscht, und dieser Prozess des Durchsuchens und Vertauschens wird so lange fortgesetzt, bis man sich beim Durchsuchen aus beiden Richtungen irgendwo trifft.

Als Resultat dieses Vorgangs ist jetzt das Array zerlegt in einen linken Teil, in dem alle Elemente kleiner als x sind und einen rechten Teil, in dem alle Elemente größer als x sind. Zwischen diesen beiden Teilen sind also keine Austauschvorgänge mehr nötig.

Man hat also das Problem insoweit vereinfacht, dass durch diesen Vorgang nur noch das Sortieren der beiden Teile (nach dem gleichen Verfahren) übrig bleibt. Es ist ein Verfahren nach dem Prinzip *divide and conquer* (teile und herrsche) und führt dementsprechend auf eine rekursive Methode.

Handarbeit

Um den unten ausformulierten Algorithmus zu **verstehen**, muss man ihn an Beispielen in Handarbeit selbst mit Bleistift und Papier ausführen. Mache dies ausführlichst am Beispiel der Zahlenfolge 8 2 1 7 9 5 3. Notiere jede Änderung der Indizes und schreibe jeweils bei Vertauschung von Elementen die Folge neu auf.

Programmierung

Der Algorithmus kann erstaunlich knapp formuliert werden. Er ist aber **äußerst empfindlich** gegenüber Fehlern bei den Laufindizes bzw. bei den Vergleichen. Die kleinste Änderung führt zum Absturz oder zu einer fehlerhaften Sortierung.

Als Pivot-Element wird hier der Einfachheit halber ein Element genommen, das so etwa in der Mitte liegt.

```
1 public void quicksort(int links, int rechts)
2 { int nachlinks = rechts; // Laufindex laeuft vom rechten Ende nach links
3   int nachrechts = links; // Laufindex laeuft vom linken Ende nach rechts
4   if (nachrechts < nachlinks)
5     { // Pivotelement bestimmen
6       int pivot = feld[(nachrechts + nachlinks)/2];
7
8       while (nachrechts <= nachlinks)
9         { // Links erstes Element suchen, das
10          // groesser oder gleich dem Pivotelement ist
11          while ((nachrechts < rechts) && (feld[nachrechts] < pivot))
12            nachrechts++;
13
14          // Rechts erstes Element suchen, das
15          // kleiner oder gleich dem Pivotelement ist
16          while ((nachlinks > links) && (feld[nachlinks] > pivot))
17            nachlinks--;
18
19          // Wenn nicht aneinander vorbei gelaufen, Inhalte vertauschen
20          if (nachrechts <= nachlinks)
21            { vertausche(nachrechts, nachlinks);
22              nachrechts++;
23              nachlinks--;
24            }
25          } // end while
26
27          // Linken Teil sortieren
28          if (nachlinks > links) quicksort (links, nachlinks);
29
30          // Rechten Teil sortieren
31          if (nachrechts < rechts) quicksort (nachrechts, rechts);
32
33        } // end if
34    } // quicksort
```

Testphase

Teste den Algorithmus nicht nur an Feldern mit zufälligen Inhalten, sondern auch an „fast schon“ sortierten Feldern und umgekehrt sortierten Feldern. Stelle das Zeitverhalten des Algorithmus graphisch dar und stelle fest, wie die benötigte Zeit von der Anzahl der Elemente n abhängt. Vergleiche die verschiedenen Fälle mit den anderen Sortieralgorithmen.

Teste den Algorithmus auch mit einer anderen Strategie zur Ermittlung des Pivot-Elementes, um auch die „schlimmst-möglichen“ Fälle zu entschärfen.