

Quicksort

Quicksort ist ein allgemeines Sortierverfahren vom Typ «Teile und Herrsche». Es beruht auf dem Zerlegen eines zu sortierenden Feldes in zwei Teile und dem anschließenden Sortieren der beiden Teile unabhängig voneinander. Der Algorithmus hat folgende Gestalt:

```
void quicksort (int A[], int l, int r)
{
    if(l >= r) return;
    int i = l;
    int j = r+1;
    int v = A[l];
    for (;;)
    {
        while(A[++i] < v && i < r);
        while(A[--j] > v);
        if(i >= j) {
            swap(A, l, j);
            break;
        }
        swap(A, i, j);
    }
    quicksort(A, l, j-1);
    quicksort(A, j+1, r);
}
```

Die Parameter `l` und `r` begrenzen das Teilfeld innerhalb des ursprünglichen Feldes, welches zu sortieren ist; der Aufruf `quicksort(A, 0, A.size()-1)` sortiert das gesamte Feld.

Das entscheidende Element der Methode ist der Programmcode innerhalb der unendlich Schleife `for(;;)`, der das Feld so umordnet, daß die folgenden Bedingungen erfüllt sind:

- i. Für ein beliebiges `j` befindet sich das Element `A[j]` an seinem endgültigen Platz im Feld.
- ii. Alle Elemente `A[l]`, `...`, `A[j-1]` sind kleiner oder gleich `A[j]`.
- iii. Alle Elemente `A[j+1]`, `...`, `A[r]` sind größer oder gleich `A[j]`.

Im ersten Schritt wird das zerlegende Element `v = A[l]` ausgewählt, das in seine endgültige Position gebracht werden soll. Jetzt beginnt die Suche von links, bis ein Element gefunden wurde, welches größer als `v` ist, anschließend startet die Suche von rechts, bis ein Element gefunden wurde, das kleiner als `v` ist.

Haben sich die Zeiger `i` und `j` noch nicht getroffen, dann werden die Elemente `A[i]` und `A[j]` miteinander vertauscht, andernfalls wird durch einen weiteren Austausch die Bedingung i. sichergestellt und die unendlich Schleife durch ein `break` verlassen.

Im folgenden wird die Funktion für das linke Teilfeld und danach für das rechte Teilfeld rekursiv aufgerufen. Am Ende ist das Feld aufsteigend sortiert.

Korrektheit des Vefahrens

Beweis.

Zu zeigen ist, daß in jedem Iterationsschritt gilt:

$$A[1], \dots, A[i] \leq A[j], \dots, A[r].$$

Induktionsanfang

Vor der ersten Iteration ist $i = 1$ und $j = r+1$.

Induktionsschritt

D.h. der Übergang von $i_{old} \rightarrow i_{new}$ und $j_{old} \rightarrow j_{new}$.

Nach Induktionsannahme gilt:

$$A[1], \dots, A[i_{old}] \leq A[j_{old}], \dots, A[r]$$

Nach Konstruktion gilt:

$$A[i_{old}+1], \dots, A[i_{new}-1] \leq A[j_{new}+1], \dots, A[j_{old}+1]$$

hieraus folgt

$$A[1], \dots, A[i_{new}-1] \leq A[j_{new}+1], \dots, A[r]$$

weiter gilt:

$$A[i_{new}] \geq x \text{ und } A[j_{new}] \leq x$$

Nach Vertauschen von $A[i_{new}]$ mit $A[j_{new}]$ gilt:

$$A[1], \dots, A[i_{new}] \leq A[j_{new}], \dots, A[r]$$

□

Laufzeitverhalten (worst-case)

Die Vermutung ist, daß ein schlimmster Fall eintritt, wenn Quicksort eine aufsteigend sortierte Liste sortieren soll. Dann fällt nämlich in jedem Rekursionsaufruf immer nur ein Element aus der Liste der noch zu sortierenden Folge (siehe Animation). Die resultierende Laufzeit $T(n)$ hat die Komplexität

$$\begin{aligned} T(n) &= \mathcal{O} \left(\sum_{i=2}^n i + n \cdot c \right) \\ &= \mathcal{O} (n^2) \end{aligned}$$

Beweis.

Allgemein gilt:

$$T(n) = \max_{1 \leq q \leq n-1} \{T(q) + T(n-q) + c_1 \cdot n\}$$

Es wird nun gezeigt, daß $T(n) \leq (c \cdot n^2)$ für $c \geq \frac{c_1}{2} \cdot \frac{n_0}{n-1}$ für n_0 groß genug gilt.

$$\begin{aligned} T(n) &\leq \max_{1 \leq q \leq n-1} \{T(q) + T(n-q) + c_1 \cdot n\} \\ &\stackrel{\text{i.A.}}{\leq} \max_{1 \leq q \leq n-1} \{c \cdot q^2 + c \cdot (n-q)^2 + c_1 \cdot n\} \\ &= c \cdot \max_{1 \leq q \leq n-1} \{q + (n-q)^2 - 2q \cdot (n-q)\} + c_1 \cdot n \\ &= c \cdot n^2 - 2c \cdot \min_{1 \leq q \leq n-1} \{q \cdot (n-q)\} + c_1 \cdot n \\ &= c \cdot n^2 - 2c \cdot 1(n-1) + c_1 \cdot n \end{aligned}$$

für n groß genug folgt

$$\begin{aligned} T(n) &= c \cdot n - 2 \cdot \frac{c_1}{2} \cdot \frac{n}{n-1} \cdot (n-1) + c_1 \cdot n \\ &= c \cdot n^2 \end{aligned}$$

□

In der Praxis kommt der Algorithmus sehr oft zum Einsatz, dies liegt unter anderem an der Laufzeit im Mittel, die im folgenden betrachtet wird.

Satz 1. *Quicksort benötigt im Mittel $2 \log n + \Theta(1)$ viele Vergleiche. Durch randomisiertes Quicksort (zufälliges Vertauschen von zwei Elementen in dem noch zu sortierenden Feld) wird der schlechteste Fall mit hoher Wahrscheinlichkeit vermieden.*

Beweis.

$$QSA(n) = n + 2 + \frac{1}{n-1} \cdot \sum_{1 \leq q \leq n-1} (QSA(q) + QSA(n-q))$$

mit

$$\sum_{1 \leq q \leq n-1} QSA(q) = \sum_{1 \leq q \leq n-1} QSA(n-q)$$

folgt

$$QSA(n) = n + 2 + \frac{2}{n-1} \cdot \sum_{1 \leq q \leq n-1} QSA(q)$$

$$(n-1) \cdot QSA(n) = (n+2) \cdot (n-1) + 2 \cdot \sum_{1 \leq q \leq n-1} QSA(q) \quad | -$$

$$(n-2) \cdot QSA(n) = (n+1) \cdot (n-2) + 2 \cdot \sum_{1 \leq q \leq n-1} QSA(q)$$

$$(n-1) \cdot QSA(n) - (n-2) \cdot QSA(n-1) = \underbrace{(n+2) \cdot (n-1)}_{n^2-n+2} - \underbrace{(n+1) \cdot (n-2)}_{n^2+n-2} + 2 \cdot QSA(n-1)$$

$$(n-1) \cdot QSA(n) = 2 \cdot n + n \cdot QSA(n-1) \quad | : n \cdot (n-1)$$

$$\begin{aligned} \frac{QSA(n)}{n} &= \frac{QSA(n-1)}{n-1} + \frac{2}{n-1} \\ &= \frac{QSA(n-2)}{n-2} + \frac{2}{n-2} + \frac{2}{n-1} \\ &= \frac{QSA(n-2)}{n-3} + \frac{2}{n-3} + \frac{2}{n-2} + \frac{2}{n-1} \\ &\dots \end{aligned}$$

$QSA(k) = c$ für ein festes k folgt

$$\begin{aligned} \frac{QSA(n)}{n} &= \Theta \left(c + 2 \cdot \sum_{i=1}^{n-1} \frac{1}{i} \right) \\ &= 2 \log n + \Theta(1) \\ &= \Theta(\log n) \end{aligned}$$

$$QSA(n) = \Theta(n \log n)$$

□