3 - Rekursion

Montag, 14. April 2014 08:09

Kann man effizienter sortieren?

Divide and conquer Strategie:

Divide: Zerlege das Problem in Unterprobleme

Conquer: Löse die Unterprobleme
- entweder durch Zerlegung in weitere Unterprobleme, also rekursiv.
- oder direkt, wenn das Problem klein genug ist.

Combine: Kombiniere die Teillösungen zu einer Gesamtlösung.

Anwendung:

Divide: Zerlege Folge in zwei Teilfolgen halber Länge

Conquer: Sortiere die Teilfolgen

Merge: Füge zwei sortierte Teilfolgen zusammen (Merge)

Spezifikation von Merge (A, p, q, r)

Eingabe
* $p \leq q < r$
* $A[p..q]$ sortiert

* $A[q+1..r]$ sortiert

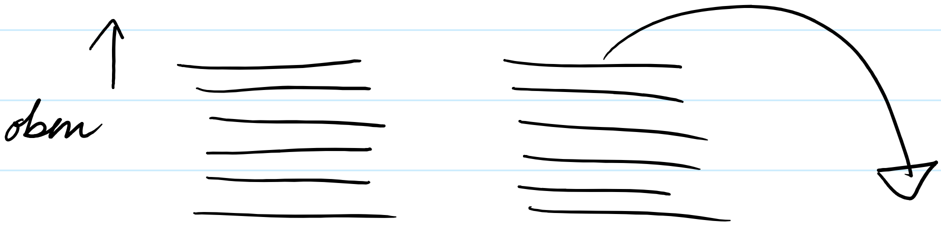
Ausgabe

$A[p..r]$ sortiert

Wenn ein solches Programm Merge existiert, kann man Merge Sort folgendermaßen implementieren:

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Jetzt diskutieren wir Merge. Hier ist die Idee.



zwei Kartenstapel.

* Vergleiche die beiden oberen Karten und wähle die kleinere

* Lege sie mit dem Gesicht nach unten auf den Ausgabestapel.

Optimierung:

Füge an die Teilfolgen das Element ∞ an. Dann braucht man nicht jedesmal abfragen, ob der Stapel leer ist.

Merge (A, p, q, r)

hier der Algorithmus.

```
MERGE(A, p, q, r)
1  n₁ = q − p + 1
2  n₂ = r − q
3  let L[1 .. n₁ + 1] and R[1 .. n₂ + 1] be new arrays
4  for i = 1 to n₁
5      L[i] = A[p + i − 1]
6  for j = 1 to n₂
7      R[j] = A[q + j]
8  L[n₁ + 1] = ∞
9  R[n₂ + 1] = ∞
10 i = 1
11 j = 1
12 for k = p to r
13     if L[i] ≤ R[j]
14         A[k] = L[i]
15         i = i + 1
16     else A[k] = R[j]
17         j = j + 1
```

Schleifeninvariante:

Am Anfang der for-Schleife in Zeile 12 – 17 enthält

$A[p, \ldots, k-1]$ die $k-p$ kleinsten Elemente von $L[1..n_1+1]$, $R[1..n_2+1]$

in sortierte Form.
$L[i]$, $R[j]$ sind die kleinsten bis jetzt nicht kopierten Elemente in L und R.

Wir beweisen diese Invariante induktiv und leiten daraus die Eigenschaften des Algorithmus ab.

Erster Durchlauf:

$k = p$, $i = 1$, $j = 1$
$L[1]$ ist das kleinste Element in L
$R[1]$ ist das kleinste Element in R
$k = p$ impliziert, dass die Folge $A[p, k-1]$ leer ist. Also ist die erste Aussage trivial und die zweite Aussage wahr.

Angenommen die Invariante gilt vor der Schleife, dann auch nachher.

Das sieht man so:

Wir wissen bereits, dass $L[i]$, $R[j]$ die beiden noch nicht eingeordneten Elemente von L bzw. R sind. Wir wissen auch, dass alle eingeordneten Elemente von L höchstens so groß wie $L[i]$ sind und dass alle eingeordneten Elemente von R höchstens so groß wie $R[j]$ sind. Das liegt daran, dass L und R sortierte Folgen sind. Wenn $L[i] \leq R[j]$ ist, wird $L[i]$ als Element $A[k]$ eingeordnet und i um 1 erhöht. Also ist die Invariante erfüllt. Im anderen Fall gilt das Entsprechende.

Terminierung:

Bei Terminierung ist $k = r+1$. Die Invariante impliziert, dass $A[p,..,r]$ die Elemente von L und R enthält und sortiert ist. Nach Konstruktion gilt aber

$$L[1,..,n_1] = A[p,..,q-1]$$

$$R[1,..,n_2] = A[q,..,r]$$

Also ist das neue A das alte A in sortierter Reihenfolge.

Wir analysieren Merge.

<u>Behauptung:</u> Merge benötigt Zeit $\Theta(n)$. mit $n = r-p+1$.

Um das zu beweisen, stellen wir zunächst fest, dass Zeilen 1-3 und 8-11 konstante Zeit benötigen.

Die for-Schleifen in Zeilen 4-7 benötigen Zeit $\Theta(n_1+n_2) = \Theta(q-p+1+r-q)$
$= \Theta(r-p+1) = \Theta(n)$.

Die for-Schleife in Zeilen 12-17 benötigt $\Theta(p-r+1) = \Theta(n)$ Durchläufe. Die Anweisungen in dieser Schleife benötigen Zeit $\Theta(1)$. Also ist die gesamte Laufzeit $\Theta(n)$.

Jetzt wird der gesamte Algorithmus analysiert.

Die Laufzeit wird mit $T(n)$ bezeichnet. In der Analyse benutzen wir folgendes

1. Wenn die Anzahl der zu sortierenden Objekte hinreichend klein ist, also unterhalb einer Schranke c, so ist die Laufzeit konstant ($\Theta(1)$)

2. Bei der Rekursion setzt sich die Laufzeit zusammen aus den Laufzeiten der kleineren Probleme (in unserem Fall sind das 2), der Laufzeit

$D(n)$ für die Aufteilung des Problems und der Laufzeit $C(n)$ für die Zusammensetzung der kleineren Lösungen. Allgemein bekommt man damit die Formel

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ \underbrace{a\,T(n/b)}_{\substack{a \text{ Teilprobleme} \\ \text{der Größe } n/b}} + \underbrace{D(n)}_{\text{Divide}} + \underbrace{C(n)}_{\text{Combine}} & \text{otherwise} \end{cases}$$

Für Merge Sort ergibt sich speziell

$$T(n) = \begin{cases} c & n = 1 \\ 2T(n/2) + c \cdot n \end{cases}$$

Die Analyse vereinfacht sich sehr, wenn wir $n = 2^k$ untersuchen. Dann erhält man:

$$\begin{aligned} T(n) &= 2\,T(n/2) + c \cdot n \\ &= 2^2\,T(n/2^2) + 2\,c \cdot n \\ &= \ldots = \\ &= 2^k\,T(1) + k \cdot c \cdot n \end{aligned}$$

$$= (n + n \lg n) \cdot c$$

$$= \Theta(n \lg n)$$