Algorithmen und Datenstrukturen
Tutorium VI

Michael R. Jung

01. - 06. 06. 2016

Aufgabe 1

Wählen sie für die folgenden Eingaben ein geeignetes Sortierverfahren aus und begründen Sie Ihre Antwort!

1. Array aller Binärzahlen von 0 bis $2^9 - 1$.
2. Liste großer Objekte mit beliebigen Strings als Schlüssel.
3. Liste großer Objekte mit sechsstelligen Strings als Schlüssel.
4. 2.000.000 Würfelwürfe.

Lösung (Beispiele):

1. RadixExchangeSort. Gleiche Asymptotische Laufzeit wie MergeSort und QuickSort(BestCase) aber weniger bzw. nicht mehr Speicher.
2. BubbleSort oder InsertionSort, da das Swappen benachbarter Elemente auch in einfach verketten Listen leicht zu realisieren ist.
3. BucketSort, da es das Paradebeispiel für die Anwendung von Bucketsort ist, auch wegen der Verwendung von Listen.
4. BucketSort oder Linearsort (Übungsaufgabe). Lineare Laufzeit.

SwapSort

```
1  Input: Array A der Länge n
2  Output: sortiertes Array
3  for i=1 to ⌊n/2⌋ do
4    if A[i]>A[n-i+1] then swap(i,n-i+1);
5    endif
6  endfor
7  if (n mod 2) = 1 then
8    m:=⌈n/2⌉
9    if A[m]>A[m+1] then swap(m,m+1);
10   else if A[m-1]>A[m] then swap(m-1,m);
11   endif
12 endif
13 SwapSort(A[1..⌊n/2⌋]);
14 SwapSort(A[⌊n/2⌋+1..n]);
```

Aufgabe 2

Analysieren Sie die Laufzeit des Algorithmus "SwapSort". Zeigen oder widerlegen Sie dessen Korrektheit!

Korrektheitsbeweis

Induktion über n:

IA $n \leq 3$ klar.

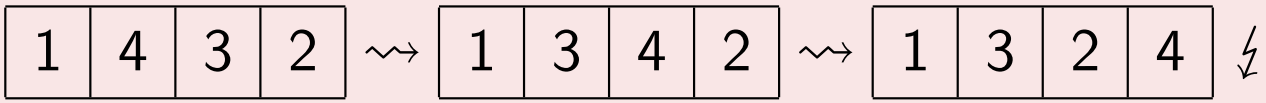
IV SwapSort sortiert Instanzen der Größe $< n$ korrekt.

IS Der Algorithmus vergleicht jeweils ein Element aus der vorderen mit einem Element aus der hinteren Hälfte. Sollte das vordere größer sein, so gehört es in die hintere Hälfte und umgekehrt, und es wird geswappt. Zum Schluss wird auch noch das mittlere Element, falls vorhanden, in die richtige Hälfte geschoben.
Da nun alle Elemente in der ersten Hälfte kleiner sind als die Elemente in der zweiten Hälfte und SwapSort auf kleineren Instanzen korrekt arbeitet (IV), folgt die Korrektheit von SwapSort.
□

Achtung!

Im vorangehenden Beweis befinden sich Fehler. Die Behauptung, am Ende eines Durchganges befände sich ein jedes Element in der richtigen Hälfte, ist falsch!

Gegenbeispiel:



BackSort

```
1  Input: Array A der Länge n
2  Output: sortiertes Array
3  for i=n downto 2 do
4    if A[i]<A[i-1] then
5      swap(i,i-1);
6      j=i;
7      while j<n and A[j]>A[j+1] do
8        swap(j,j+1);
9        j:=j+1;
10     endwhile
11   endif
12 endfor
```

Aufgabe 3

Analysieren Sie die Laufzeit des Algorithmus "BackSort". Zeigen oder widerlegen Sie dessen Korrektheit

Behauptung: Nach dem k-ten Durchlauf sind die letzten $k+1$ Stellen korrekt sortiert. (Da wir $n-1$ Durchläufe machen, folgt somit schlussendlich die Korrektheit von BackSort.)
Induktion über k:

IA $k = 1$ klar.

IS Betrachte $k+1$. Nach IV sind die letzten k Stellen korrekt sortiert. Gilt $A[n-(k+1)] \leq A[n-k]$, so sind die letzten k Stellen korrekt sortiert und es nichts zu tun. Anderenfalls werden diese beiden Elemente geswappt. Danach beginnen wir die while-Schleife. Hier wird unser ursprüngliches $A[n-(k+1)]$ solange nach hinten durchgetauscht, bis das dahinter stehende Element nicht kleiner ist. Da die relative Reihenfolge der anderen Elemente (welche nach IV ja korrekt ist) nicht verändert wird, steht das betrachtete Element an der richtigen Stelle.

□

AverageQuickSort

```
1  Input: Array A der Länge n
2  Output: sortiertes Array
3  if n>1 then
4    m:=0;
5    for i=1 to n do
6      m:=m+(A[i]/n);
7    endfor
8    repeat
9      i:=1;j:=n;
10     while i<j and A[i]≤m do i:=i+1; endwhile
11     while i<j and A[j]>m do j:=j+1; endwhile
12     if A[i]>A[j] then swap(i,j); endif
13   until i≥j
14   AverageQuickSort(A[1..i-1]);
15   AverageQuickSort(A[i..n]);
16 endif
```

Aufgabe 4

Zeigen Sie die Korrektheit des Algorithmus "AverageQuickSort"!
Geben Sie eine WorstCase-Instanz an und analysieren Sie die Laufzeit!

Korrektheitsbeweis:

Induktion über die Länge des Arrays:

IA: $n = 1$ klar.

IV: `AverageQuickSort` sortiert Arrays der Länge $< n$ korrekt.

IS: Dass der Algorithmus immer terminiert, ist klar, da er sich auf immer kleiner werdenden Instanzen aufruft.
Zuerst wird das arithmetische Mittel aller Werte im Array errechnet. Sollte es also verschiedene Werte im Array geben, so exitiert mindestens ein Wert $\leq m$ und ein Wert $> m$ (falls nicht muss der Algorithmus korrekt sein , da er terminiert und höchstens Vertauschungen vornimmt).

Der Algorithmus verschiebt nun alle Elemente die $\leq m$ sind, in den vorderen Teil (sie stehen am Ende alle vor `i` und dort stehen nur solche Werte) und die anderen in den hinteren Teil. Folglich reicht es nun die beiden Bereiche zu sortieren, was der Algorithmus nach IV nun auch tut. □

WorstCase-Instanz:

2	4	16	256	...	2^{2^n}

Es gilt: $\frac{\sum_{i=0}^{n+1} 2^{2^i}}{n+1} > 2^{2^n}$, d.h. der Mittelwert liegt zwischen dem vorletzten und dem letzten Wert im Array[1]. Somit wird jedesmal nur ein Element abgetrennt und wir erhalten $\Omega(n^2)$ Vergeleiche.

[1] Beweis folgt auf der nächsten Folie

Zu zeigen: $\frac{\sum_{i=0}^{n+1} 2^{2^i}}{n+1} > 2^{2^n} \Leftrightarrow \sum_{i=0}^{n+1} 2^{2^i} > (n+1)2^{2^n}$.

Es gilt aber sogar schon: $2^{2^{n+1}} > (n+1)2^{2^n}$.

Denn: $2^{2^{n+1}} = \left(2^{2^n}\right)^2$

Und offensichtlich: $2^{2^n} > n+1$.

IA $n = 0$ klar.

IS $n \rightsquigarrow n+1$:.

$$
\begin{aligned}
2^{2^{n+1}} &= \left(2^{2^n}\right)^2 \\
&\overset{\text{IV}}{>} 2^{2^n} \cdot (n+1) \\
&\geq 2(n+1) \\
&\geq (1 + \frac{1}{n+1})(n+1) \\
&= n+1+\frac{n+1}{n+1} \\
&= n+2
\end{aligned}
$$

□