

Grundlagen der Programmierung (Vorlesung 19)

Ralf Möller, FH-Wedel

- Vorige Vorlesung
 - Elementare Sortieralgorithmen und deren Analyse
- Heute
 - Quicksort als höheres Sortierverfahren
- Lernziele
 - Grundlagen der Analyse von Algorithmen

Sortierung von Reihenungen

■ Sortierproblem - Definition:

- Gegeben sei eine Reihung a der Form array [1..n] of M und eine totale Ordnung \square definiert auf M .
- Annahme: Es seien alle $a[i]$ verschieden
- Gesucht in eine Reihung b : array [1..n] of M , so daß gilt:
 $\forall 1 \square i < n . (b[i] \square b[i+1] \wedge \exists j \in \{1, \dots, n\} . (a[j] = b[i]))$

Vergleich elementarer Sortierverfahren

■ Anzahl der Vergleiche:

Verfahren	Best Case	Average Case	Worst Case
SelectionSort	$N^2/2$	$N^2/2$	$N^2/2$
InsertionSort	N	$N^2/4$	$N^2/2$
BubbleSort	$N^2/2$	$N^2/2$	$N^2/2$

■ Anzahl der Bewegungen:

Verfahren	Best Case	Average Case	Worst Case
SelectionSort	$3(N - 1)$	$3(N - 1)$	$3(N - 1)$
InsertionSort	$2(N - 1)$	$N^2/4$	$N^2/2$
BubbleSort	0	$3N^2/4$	$3N^2/2$

Folgerungen

BubbleSort: ineffizient, da immer $N^2/2$ Vergleiche

InsertionSort: gut für fast sortierte Folgen

SelectionSort: gut für große Datensätze aufgrund konstanter Zahl der Bewegungen, jedoch stets $N^2/2$ Vergleiche

Fazit: InsertionSort und SelectionSort sollten nur für $N \leq 50$ eingesetzt werden.

Höhere Sortierverfahren: Quicksort

QuickSort wurde 1962 von C.A.R. Hoare entwickelt.

Prinzip: Das Prinzip folgt dem Divide-and-Conquer-Ansatz:

Gegeben sei eine Folge F von Schlüsselementen.

1. Zerlege F bzgl. eines partitionierenden Elementes (engl.: *pivot* = Drehpunkt) $p \in F$ in zwei Teilfolgen F_1 und F_2 , so daß gilt:

$$\begin{array}{ll} x_1 \leq p & \forall x_1 \in F_1 \\ p \leq x_2 & \forall x_2 \in F_2 \end{array}$$

2. Wende dasselbe Schema auf jede der so erzeugten Teilfolgen F_1 und F_2 an, bis diese nur noch höchstens ein Element enthalten.

Quicksort

```
procedure quicksort( $l, r : N_0$ )
begin
  var  $k : N_0$ ;
  if  $l < r$ 
  then  $k := \text{partition}(l, r)$ ;
      quicksort( $l, k-1$ );
      quicksort( $k+1, r$ );
  end if
end
```

Partition

```
function partition(l, r : N0) : N0
begin
  var i, j, p : N0;
  i, j, p := l-1, r, a[r]; (* Wähle Pivot-Wert *)
  repeat
    durchsuche Array von links (i:=i+1), solange bis a[i] >= p;
    durchsuche Array von rechts (j:=j-1), solange bis a[j] <= p;
    vertausche a[i] und a[j];
  until j <= i (* Zeiger kreuzen *)
  rückvertausche a[i] und a[j];
  vertausche a[i] und a[r]; (* positioniere Pivot-Element *)
  i; (* endgültige Position des Pivot-Elements *)
end
```

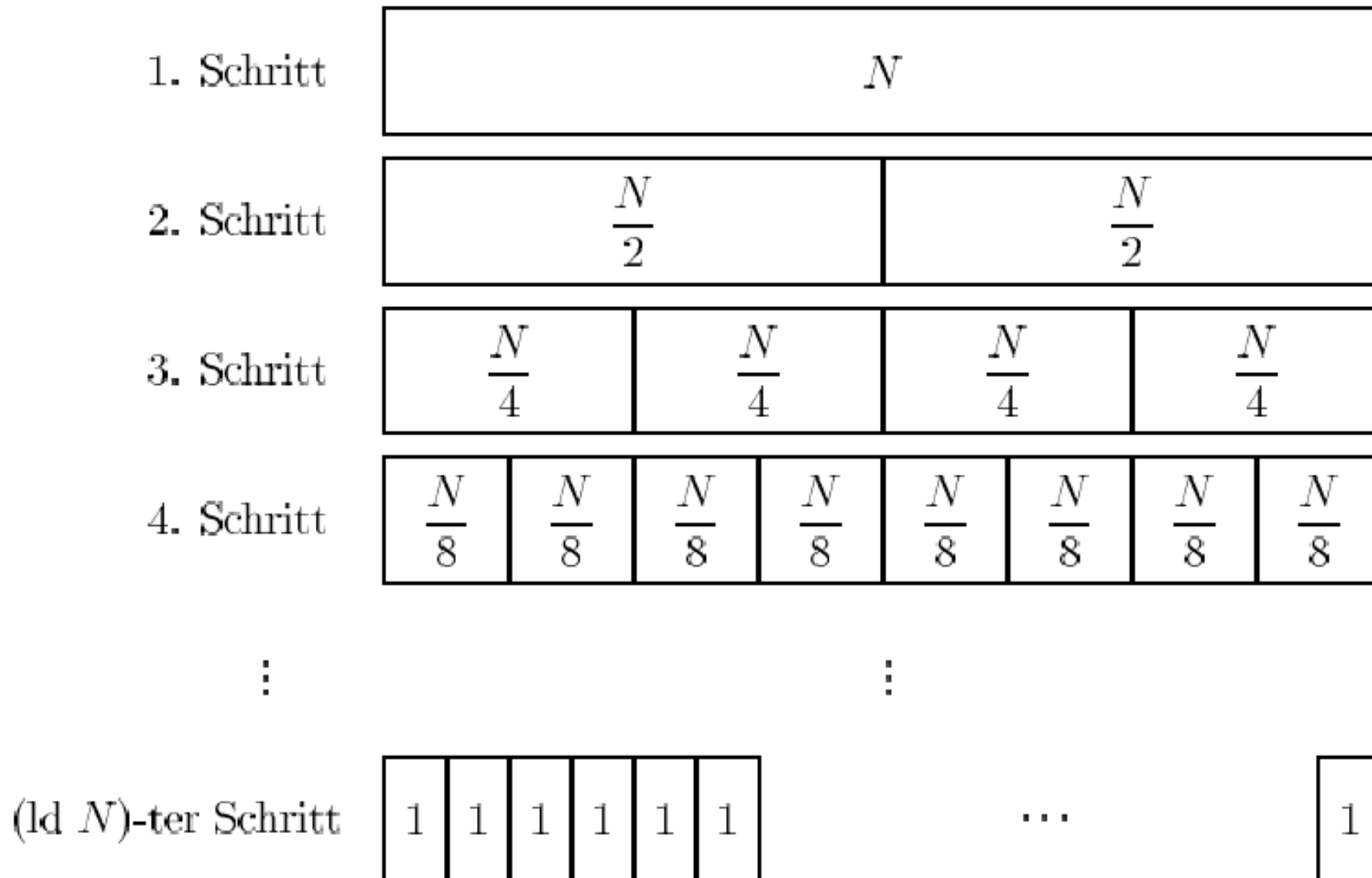
Partition (2)

```
function partition(l, r : N0) : N0
begin
  var i, j, p : N0;
  i, j, p := l-1, r, a[r]; (* Wähle Pivot-Wert *)
  repeat
    repeat i:=i+1 until a[i] >= p;
    repeat j:=j-1 until a[j] <= p;
    a[i], a[j] := a[j], a[i];
  until j <= i (* Zeiger kreuzen *)
  a[j], a[i] := a[i], a[j];
  a[i], a[r] := a[r], a[i];
  i; (* endgültige Position des Pivot-Elements *)
end
```

Partition (3)

- Wahl des Pivot-Wertes im Prinzip willkürlich
- Korrektheit des hier besprochenen Algorithmus ist von der Wahl $a[r]$ abhängig

Komplexitätsabschätzung



Zusammenfassung, Kernpunkte



■ Einfache Sortierverfahren

- Sortieren durch Auswahl
- Sortieren durch Einfügen
- Sortieren durch paarweises Vertauschen (Bubblesort)

■ Höhere Sortierverfahren

- Quicksort

■ Komplexitätsabschätzung

- n^2 vs. $n \log n$
- Teile-und-herrsche-Prinzip

Was kommt beim nächsten Mal?



-
- Abstrakte Maschinen für spezielle Aufgaben
 - Automatentheorie und Formale Sprachen