

Algorithmen und Datenstrukturen

Wintersemester 2022/23

8. Vorlesung

Sortieren – mit dem Würfel!

*Prof. Dr. Alexander Wolff
und Johannes Zink*

Lehrstuhl für Informatik I

Organisatorisches

Diese Woche Donnerstag (17.11.) findet um 8:30 der 1. Zwischentest statt.

Wenn Sie mitschreiben möchten, melden Sie sich in der Abstimmung auf WueCampus bis morgen (16.11.) 15 Uhr an. Wir drucken die Exemplare nach vorhandenen Anmeldungen!

<https://wuecampus2.uni-wuerzburg.de/moodle/mod/choice/view.php?id=2381385>

The screenshot shows the WueCampus Moodle interface. At the top, there is a navigation bar with 'wuecampus', 'Dashboard', and 'Meine Kurse'. On the right, there are notification and user icons. A sidebar on the left contains a menu with categories like 'Foren', 'Vorlesung Algorithmen und ...', 'Kursbeschreibung', 'Allgemeine Informationen', 'Vorlesungsfolien', 'Zwischentests', and 'Übungen'. The main content area displays a poll titled 'Teilnahme am 1. Zwischentest am 17.11' with a deadline of 'Mittwoch, 16. November 2022, 15:05'. The poll options are 'Ja' and 'Nein'. A message states 'Die Ergebnisse werden nach Ihrer Antwort nicht veröffentlicht.' Below the poll, there is a 'Meine Auswahl speichern' button. Above the poll, a diagram shows five student icons labeled 'Z', 'y', 'Z', 'y', and 'X'. Red arrows point from the first 'Z' to the second 'y', and from the third 'Z' to the fourth 'y'. The 'X' icon is highlighted with a red circle and red lines.

Und noch einmal: Sortieren!

Zur Erinnerung: MergeSort...

- + gute Worst-Case-Laufzeit (durch Teile-und-Herrsche)
- kein in-situ-Verfahren (benötigt extra Felder beim Mergen)

Ziel: Teile-&-Herrsche-Verfahren, das trotzdem in situ sortiert!

Sortiere ein Teilfeld $A[\ell..r]$ wie folgt: `QuickSort(int[] A, int ℓ , r)`

Teile:
`int Partition(A, ℓ , r)`
 [liefert m zurück] } Bestimme einen Index $m \in \{\ell, \dots, r\}$ und teile $A[\ell..r]$ so in $A[\ell..m-1]$ und $A[m+1..r]$ auf, dass alle Elemente im ersten Teilfeld kleiner gleich $A[m]$ sind und alle im zweiten größer als $A[m]$.

Herrsche: durch rekursives Sortieren der beiden Teilfelder.

Kombiniere: —

Schreiben Sie QuickSort in Pseudocode unter Verwendung von Partition(A, ℓ , r)!

QuickSort

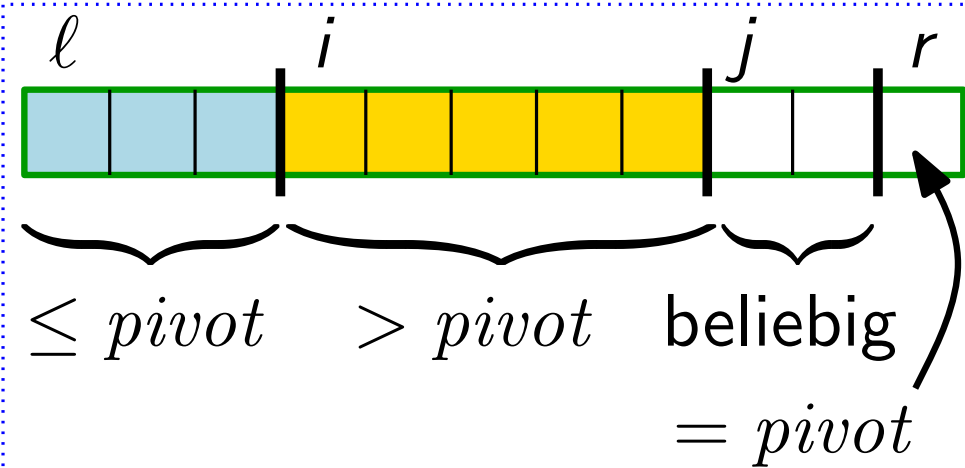
```
QuickSort( $A$ ,  $\ell = 1$ ,  $r = A.length$ )
```

```
if  $\ell < r$  then
```

```
     $m = \text{Partition}(A, \ell, r)$ 
```

```
    QuickSort( $A$ ,  $\ell$ ,  $m - 1$ )
```

```
    QuickSort( $A$ ,  $m + 1$ ,  $r$ )
```



Schleifeninvariante:

(i) Für $k = \ell, \dots, i - 1$ gilt $A[k] \leq pivot$.

(ii) Für $k = i, \dots, j - 1$ gilt $A[k] > pivot$.

(iii) $A[r] = pivot$.

(iv) $A[\ell..j-1]$ enthält die gleichen Elemente wie zu Beginn.

```
int Partition(int[]  $A$ , int  $\ell$ , int  $r$ )
```

```
     $pivot = A[r]$ 
```

```
     $i = \ell$ 
```

```
    for  $j = \ell$  to  $r - 1$  do
```

```
        if  $A[j] \leq pivot$  then
```

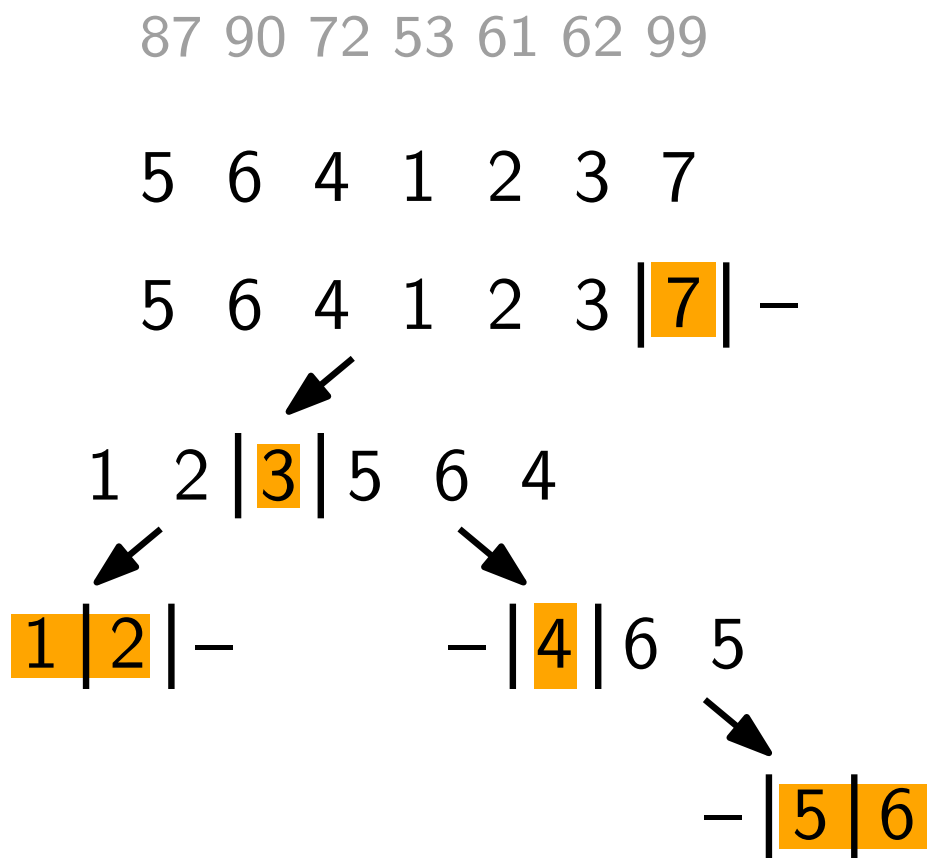
```
            Swap( $A$ ,  $i$ ,  $j$ )
```

```
             $i = i + 1$ 
```

```
    Swap( $A$ ,  $i$ ,  $r$ )
```

```
    return  $i$ 
```

Ein Beispiel



```
QuickSort( $A, \ell = 1, r = \dots$ )
```

```
if  $\ell < r$  then
```

```
     $m = \text{Partition}(A, \ell, r)$ 
```

```
    QuickSort( $A, \ell, m - 1$ )
```

```
    QuickSort( $A, m + 1, r$ )
```

```
int Partition( $A, \ell, r$ )
```

```
     $pivot = A[r]$ 
```

```
     $i = \ell$ 
```

```
    for  $j = \ell$  to  $r - 1$  do
```

```
        if  $A[j] \leq pivot$  then
```

```
            Swap( $A, i, j$ )
```

```
             $i = i + 1$ 
```

```
    Swap( $A, i, r$ )
```

```
    return  $i$ 
```

Laufzeit

Zähle Anzahl der Vergleiche!

Beob. Partition benötigt *immer* $r - \ell$ Vergleiche.

Wovon hängt dann die Laufzeit ab?

$$T_{QS}(n) = T_{QS}(m - 1) + T_{QS}(n - m) + n - 1$$

1. Extremfall: m immer erstes Element

$$\begin{aligned} T_{QS}(n) &= T_{QS}(0) + T_{QS}(n - 1) + n - 1 \\ &= (T_{QS}(n - 2) + n - 2) + n - 1 \\ &\vdots \\ &= T_{QS}(1) + 1 + 2 + \dots + n - 2 + n - 1 \\ &\in \Theta(n^2) \end{aligned}$$

2. Extremfall: m immer mittleres Element

$$T_{QS}(n) \approx 2T_{QS}(n/2) + n - 1 \in \Theta(n \log n)$$

siehe MergeSort

```
QuickSort(A, ℓ = 1, r = ...)
```

```
  if ℓ < r then
```

```
    m = Partition(A, ℓ, r)
```

```
    QuickSort(A, ℓ, m - 1)
```

```
    QuickSort(A, m + 1, r)
```

```
int Partition(A, ℓ, r)
```

```
  pivot = A[r]
```

```
  i = ℓ
```

```
  for j = ℓ to r - 1 do
```

```
    if A[j] ≤ pivot then
```

```
      Swap(A, i, j)
```

```
      i = i + 1
```

```
  Swap(A, i, r)
```

```
  return i
```

Wo ist die Wahrheit?

M.a.W. was passiert im Durchschnittsfall (*average case*)?

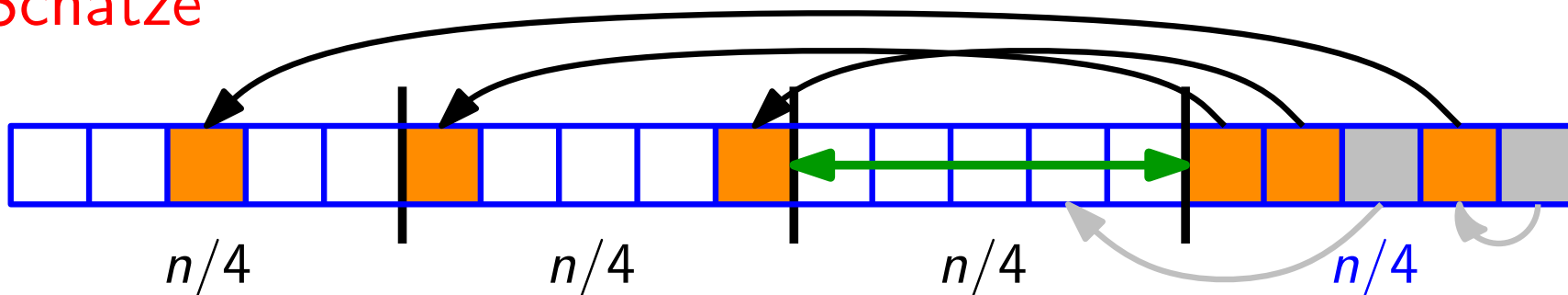
Vgl. InsertionSort: Bester Fall = $n - 1 \in \Theta(n)$ Vergleiche
 Schlechtester Fall = $\binom{n}{2} \in \Theta(n^2)$ Vergleiche
 Durchschnittsfall = $\Theta(n^2)$ ←

Mittle die Laufzeit über alle Permutationen der Eingabe!

Schwierig...

Statt dessen:

~~Berechne~~ *Schätze* erwartete Laufzeit $E[T_{IS}]$ einer zufälligen Permutation *ab!*



$$E[T_{IS}] \geq E[\text{Aufwand für letzte } \frac{n}{4} \text{ Elem.}] \geq \frac{n}{4} \cdot \frac{1}{2} \cdot \frac{n}{4} \in \Omega(n^2)$$

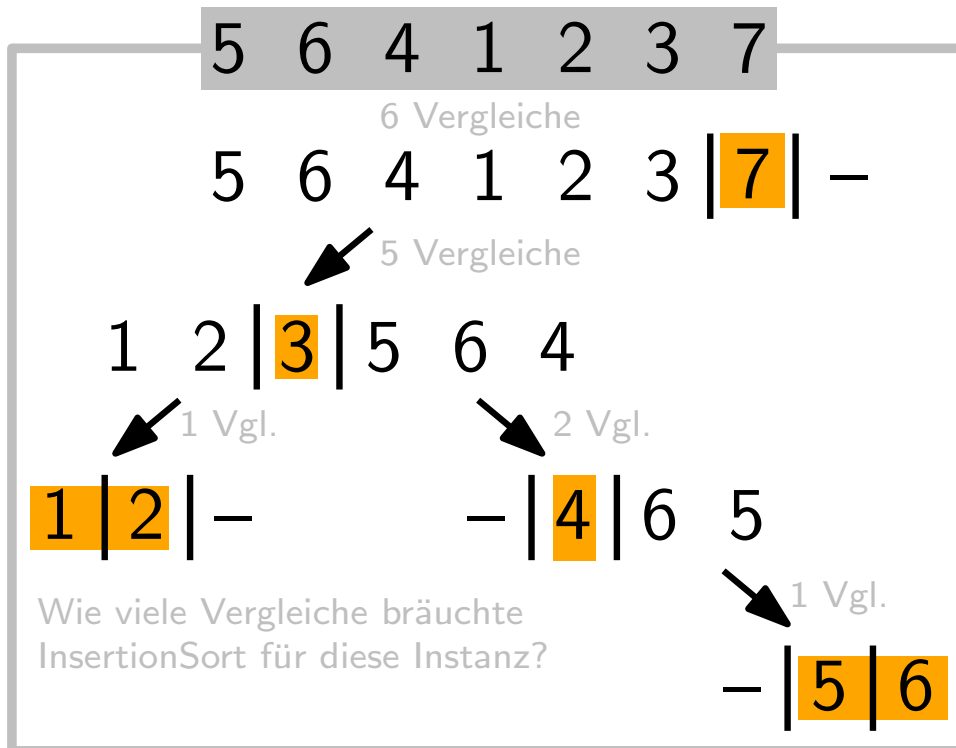
Zurück zu QuickSort

Idee: *Steck Zufall in den Algorithmus!*

Seien z_1, z_2, \dots, z_n die Elemente von A in sortierter Reihenfolge.

Wann vergleicht Alg. z_i und z_j ?

* höchstens ein Mal:
wenn eins von beiden *pivot* ist.



RandomizedPartition(A, ℓ, r)

$k = \text{Random}(\ell, r)$ Liefert Zufallszahl $\in \{\ell, \dots, r\}$.

Swap(A, r, k)

return Partition(A, ℓ, r)

Partition(A, ℓ, r)

$pivot = A[r]$

$i = \ell$

for $j = \ell$ **to** $r - 1$ **do**

if $A[j] \leq pivot$ **then**

 Swap(A, i, j)

$i = i + 1$

Swap(A, i, r)

return i

Zurück zu QuickSort

Idee: *Steck Zufall in den Algorithmus!*

Seien z_1, z_2, \dots, z_n die Elemente von A in sortierter Reihenfolge.

Wann vergleicht Alg. z_i und z_j ?

* höchstens ein Mal:
wenn eins von beiden *pivot* ist.

Definiere Indikator-Zufallsvariable:

$$V_{ij} = \begin{cases} 1, & \text{falls Alg. } z_i \text{ und } z_j \text{ vergleicht,} \\ 0 & \text{sonst.} \end{cases}$$

Sei V ZV für Gesamtanz. von Vgl.

$$\text{Dann gilt } V = \sum_{1 \leq i < j \leq n} V_{ij}.$$

$$\Rightarrow E[V] = \sum_{1 \leq i < j \leq n} E[V_{ij}]$$

Linearität des Erwartungswerts!

RandomizedPartition(A, ℓ, r)

$k = \text{Random}(\ell, r)$ Liefert Zufallszahl $\in \{\ell, \dots, r\}$.

Swap(A, r, k)

return Partition(A, ℓ, r)

Partition(A, ℓ, r)

$pivot = A[r]$

$i = \ell$

for $j = \ell$ **to** $r - 1$ **do**

if $A[j] \leq pivot$ **then**

 Swap(A, i, j)

$i = i + 1$

Swap(A, i, r)

return i

First come, first serve

$$E[V_{ij}] = \Pr[\text{Alg. vergleicht } z_i \text{ und } z_j] =$$



Betrachte die Menge $Z_{ij} := \{z_i, z_{i+1}, \dots, z_j\}$.

Sei z^* die erste Zahl in Z_{ij} , die Pivot wird.

Es gilt: Alg. vergleicht z_i und $z_j \iff z^* = z_i$ oder $z^* = z_j$.

$$\begin{aligned} \Rightarrow \Pr[\text{Alg. vergleicht } z_i \text{ und } z_j] &= \Pr[z^* = z_i \text{ oder } z^* = z_j] \\ &= \Pr[z^* = z_i] + \Pr[z^* = z_j] \\ &= \frac{1}{|Z_{ij}|} + \frac{1}{|Z_{ij}|} \\ &= \frac{2}{j - i + 1} \end{aligned}$$

Auf zum letzten Gefecht...

$$E[V_{ij}] = \Pr[\text{Alg. vergleicht } z_i \text{ und } z_j] = \frac{2}{j-i+1}$$

Wir wissen:

$$E[V] = \sum_{1 \leq i < j \leq n} E[V_{ij}] = \sum_{1 \leq i < j \leq n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \left(\sum_{j=i+1}^n \frac{2}{j-i+1} \right)$$

Trick: ersetze $j - i$ durch k !

$$= \sum_{i=1}^{n-1} \left(\sum_{k=1}^{n-i} \frac{2}{k+1} \right)$$

Auf zum letzten Gefecht...

$$E[V_{ij}] = \Pr[\text{Alg. vergleicht } z_i \text{ und } z_j] = \frac{2}{j-i+1}$$

Wir wissen:

$$E[V] = \sum_{1 \leq i < j \leq n} E[V_{ij}] = \sum_{1 \leq i < j \leq n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1}$$

Trick: ersetze $j - i$ durch k !

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^n \frac{2}{k} \in O(n \log n)$$

harmonische Reihe!

Satz: RandomizedQuickSort sortiert n Zahlen in $O(n \log n)$ erwarteter Zeit.

Zusammenfassung Sortierverfahren

	InsertionSort	MergeSort	HeapSort	QuickSort
Worst-Case-Laufzeit	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$
Avg.-Case-Laufzeit	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Best-Case-Laufzeit	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
in situ ¹ (<i>in place</i>)	✓	✗	✓	(✓)*
stabil ²	✓	✓	✗	✗

¹) Ein *in-situ*-Algorithmus benötigt nur $O(1)$ extra Speicher.

²) Sortieralg. *stabil*, wenn er gleiche Schlüssel in Ursprungsreihenfolge belässt.

*) QuickSort muss für jeden rekursiven Aufruf die Variable m zwischenspeichern. Dafür wird im worst case $\Omega(n)$ zusätzlicher Speicherplatz benötigt. Mit Tricks kann man dieses Problem umgehen und so QuickSort in-situ machen.