Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität

Institut für Informatik

Algorithmen und Datenstrukturen

Wintersemester 2023/24
8. Vorlesung

Sortieren – mit dem Würfel!

Prof. Dr. Alexander Wolff *Lehrstuhl für Informatik I*

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: 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 $A[m]$.

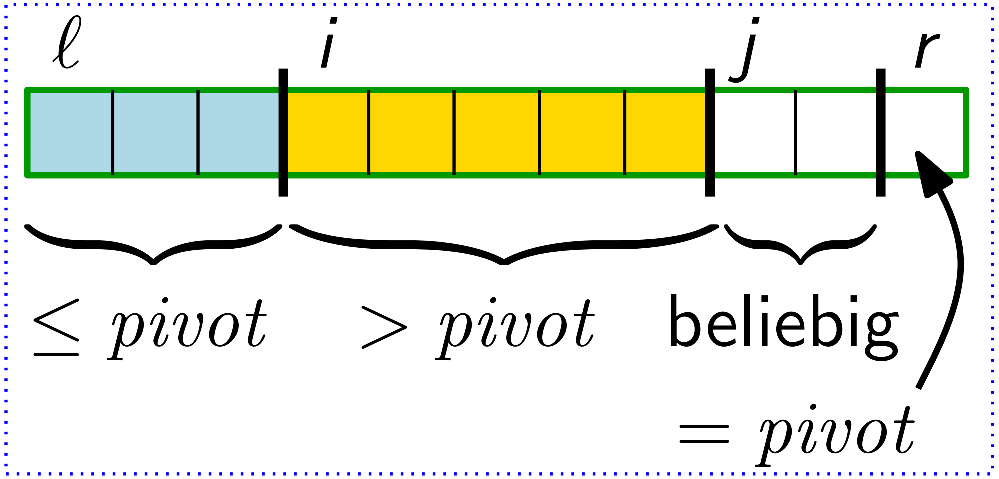
int Partition(A, ℓ, r) [liefert m zurück]

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($A, m+1, r$)

int Partition(int[] A, int ℓ, int r)

  $pivot = A[r]$
  $i = \ell$
  for $j = \ell$ **to** $r - 1$ **do**
    if $A[j] \le pivot$ **then**
      Swap(A, i, j)
      $i = i + 1$
  Swap(A, i, r)
  return i

Schleifeninvariante:

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

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

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

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

Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | $\boxed{7}$ | –

1 2 | $\boxed{3}$ | 5 6 4

$\boxed{1}$ | $\boxed{2}$ | –

– | $\boxed{4}$ | 6 5

– | $\boxed{5}$ | $\boxed{6}$

QuickSort($A, \ell = 1, r = \ldots$)

```
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
```

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

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

2. Extremfall: m immer mittleres Element

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

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

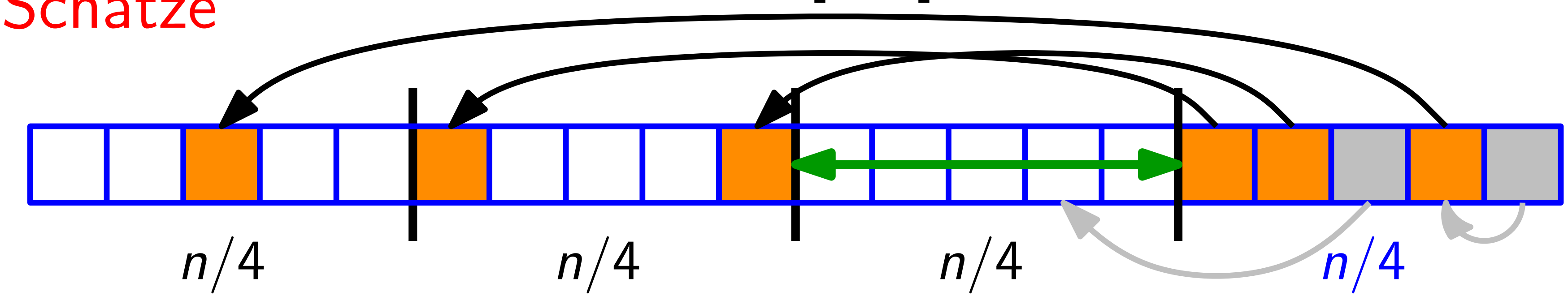
Durchschnittsfall $= \Theta(n^2)$

Mittle die Laufzeit über alle Permutationen der Eingabe!

Schwierig...

Statt dessen:

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



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

Zurück zu QuickSort

Idee: *Steck Zufall in den Algorithmus!*

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

Wann vergleicht Alg. z_i und z_j?

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

5 6 4 1 2 3 7

6 Vergleiche

5 6 4 1 2 3 | 7 | –

5 Vergleiche

1 2 | 3 | 5 6 4

1 Vgl.

1 | 2 | –

2 Vgl.

– | 4 | 6 5

1 Vgl.

– | 5 | 6

Wie viele Vergleiche bräuchte InsertionSort für diese Instanz?

RandomizedPartition(A, ℓ, r)

$k = \mathrm{Random}(\ell, r)$ Liefert Zufallszahl $\in \{\ell, \ldots, r\}$.
Swap(A, r, k)
return Partition(A, ℓ, r)

Partition(A, ℓ, r)

$\mathit{pivot} = A[r]$
$i = \ell$
for $j = \ell$ **to** $r - 1$ **do**
  if $A[j] \le \mathit{pivot}$ **then**
    Swap(A, i, j)
    $i = i + 1$
Swap(A, i, r)
return i

Zurück zu QuickSort

Idee: *Steck Zufall in den Algorithmus!*

Seien $z_1, z_2, \ldots, z_n$ 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.

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

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

Linearität des Erwartungswerts!

RandomizedPartition(A, ℓ, r)

  $k = \text{Random}(\ell, r)$ Liefert Zufallszahl $\in \{\ell, \ldots, r\}$.
  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 $A[j] \le pivot$ **then**
      Swap(A, i, j)
      $i = i + 1$
  Swap(A, i, r)
  return i

First come, first serve

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

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

Sei $z^\star$ die erste Zahl in Z_{ij}, die Pivot wird.

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

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

Auf zum letzten Gefecht...

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

Wir wissen:

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

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

Trick: ersetze $j - i$ durch k!

$$-i$$

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

Auf zum letzten Gefecht...

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

Wir wissen:

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

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

Trick: ersetze $j - i$ durch k!

harmonische Reihe!

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

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] (*in place*)	✓	✗	✓	(✓)*
stabil[2]	✓	✓	✗	✗

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

[2]) Sortieralg. *stabil*, wenn er gleiche Schlüssel in Ursprungsreihenf. 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.