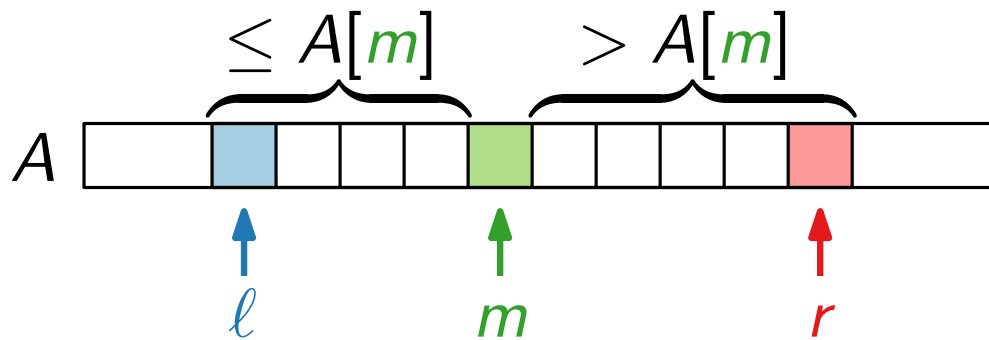
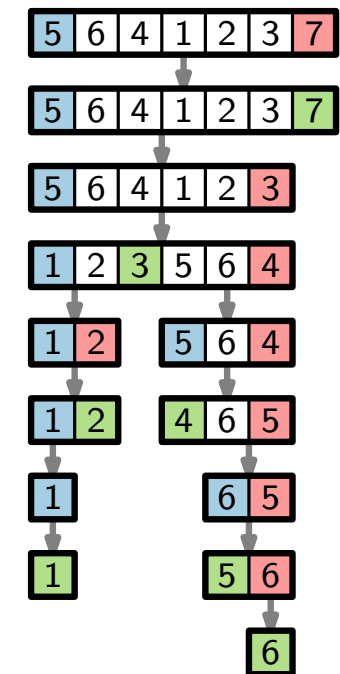


Algorithmen und Datenstrukturen

Vorlesung 8: QUICKSORT



Alexander Wolff



Wintersemester 2025

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-und-Herrsche**-Verfahren, das trotzdem **in-situ** sortiert!

QUICKSORT(int[] A, int ℓ , r)

Sortiere ein Teilfeld $A[\ell \dots r]$ wie folgt:

Teile:

int PARTITION(A, ℓ , r)

liefert m zurück

Bestimme einen Index $m \in \{\ell, \dots, r\}$ und teile $A[\ell \dots r]$ so auf, dass
 alle Elemente in $A[\ell \dots m - 1]$ kleiner gleich $A[m]$ sind und
 alle Elemente in $A[m + 1 \dots r]$ größer als $A[m]$ sind.

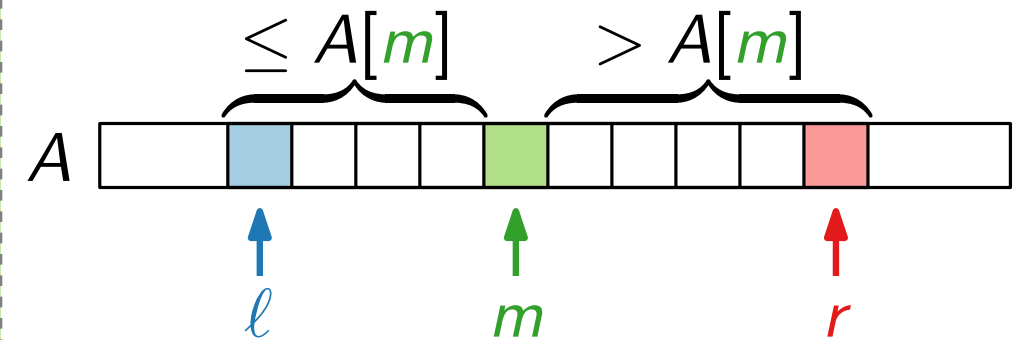
Herrsche:

durch rekursives Sortieren der beiden Teilfelder.

Kombiniere:

Aufgabe.

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  $\ell$ , int  $r$ )
```

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

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

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

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

```
             $A[i] \leftrightarrow A[j]$ 
```

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

```
     $A[i] \leftrightarrow A[r]$ 
```

```
    return  $i$ 
```

Was passiert hier?



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  $\ell$ , int  $r$ )
```

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

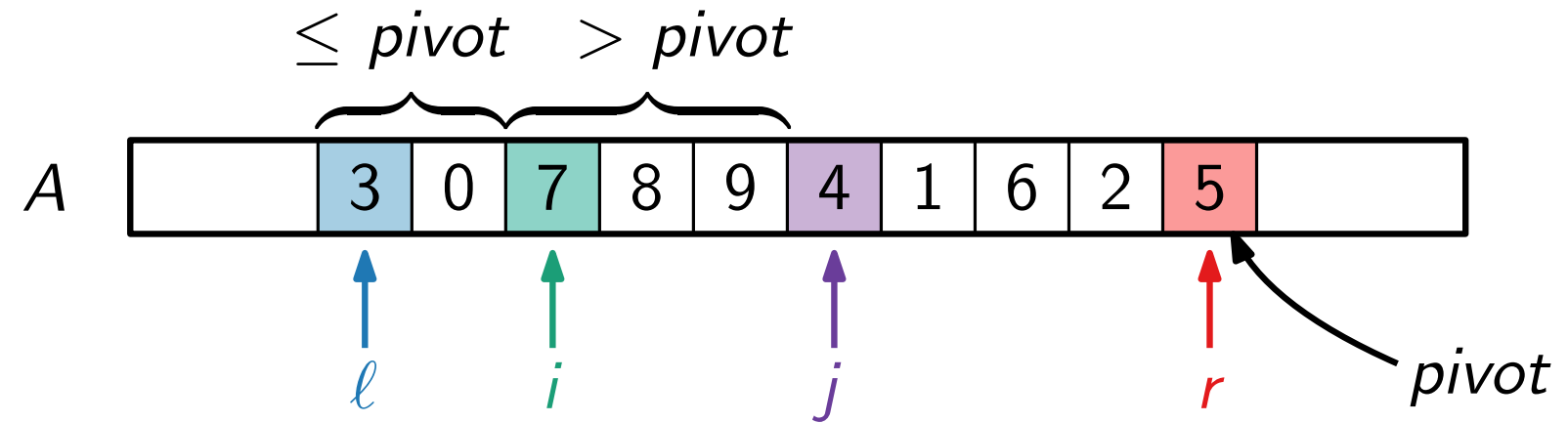
```
     $i = \ell$ 
```

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

```
        if  $A[j] \leq pivot$  then  
             $A[i] \leftrightarrow A[j]$   
             $i = i + 1$ 
```

```
     $A[i] \leftrightarrow A[r]$ 
```

```
    return  $i$ 
```



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

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  $\ell$ , int  $r$ )
```

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

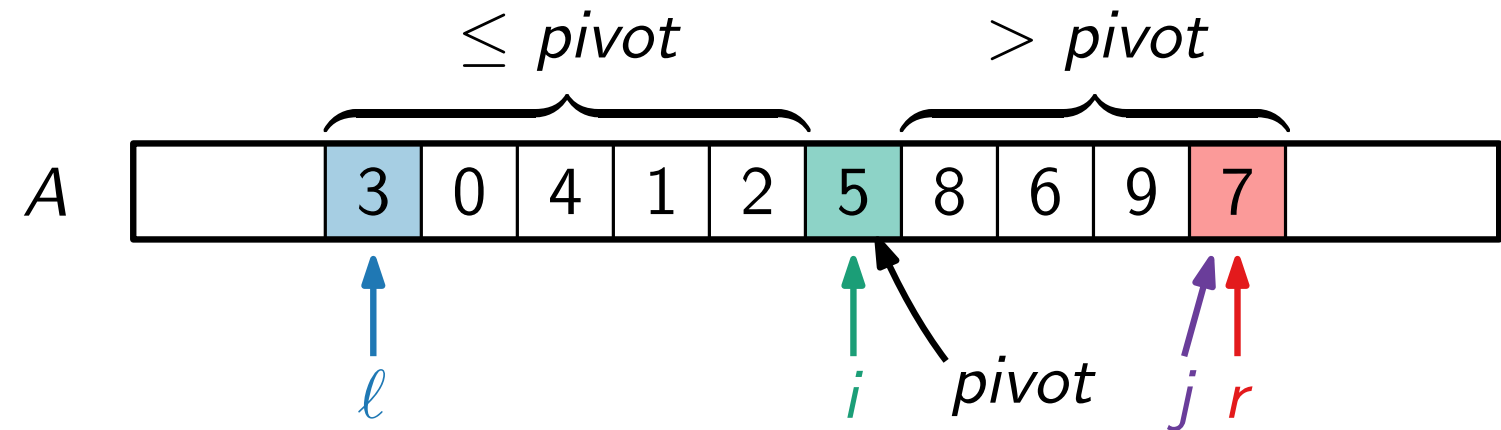
```
     $i = \ell$ 
```

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

```
        if  $A[j] \leq pivot$  then  
             $A[i] \leftrightarrow A[j]$   
             $i = i + 1$ 
```

```
     $A[i] \leftrightarrow A[r]$ 
```

```
    return  $i$ 
```



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

Ein Beispiel

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)

$pivot = A[r]$

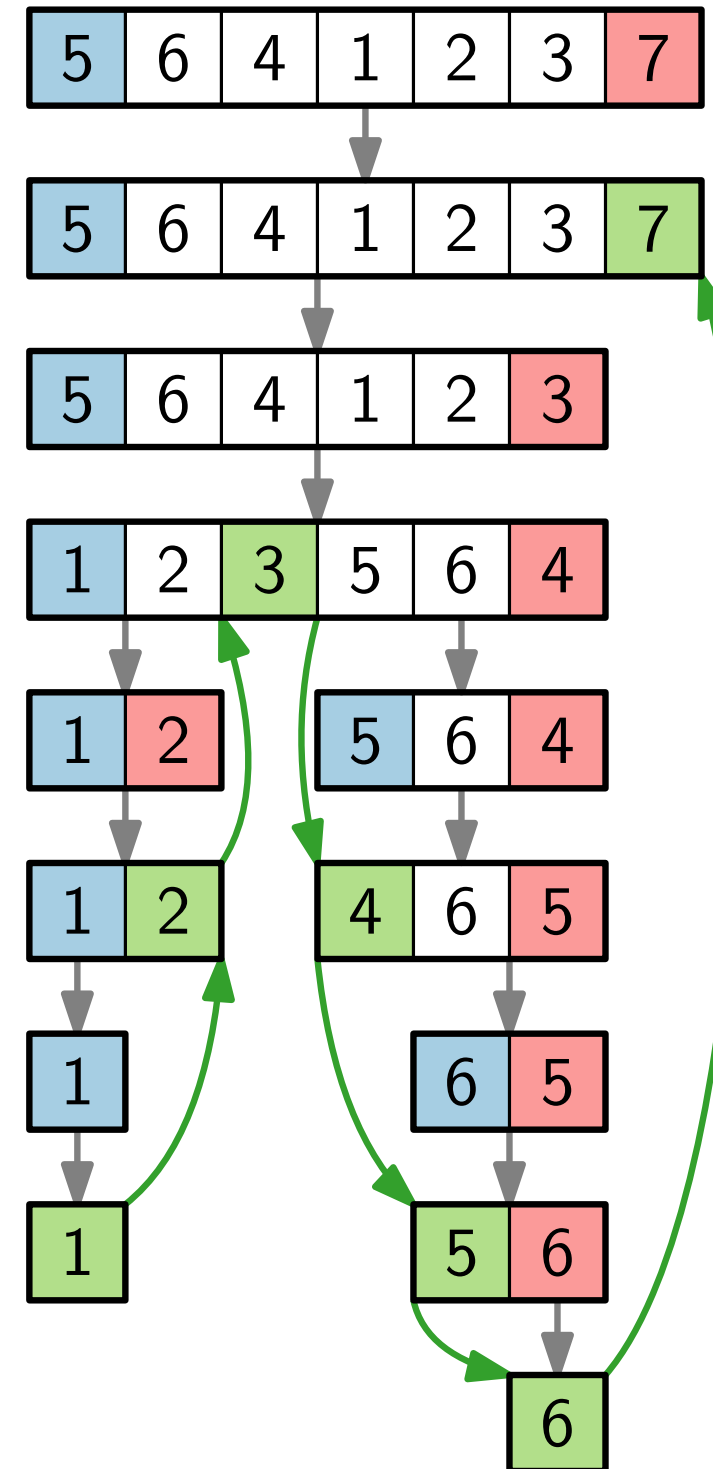
$i = \ell$

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

if $A[j] \leq pivot$ **then**
 $A[i] \leftrightarrow A[j]$
 $i = i + 1$

$A[i] \leftrightarrow A[r]$

return i



Ein Beispiel

```
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  $\ell$ , int  $r$ )
```

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

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

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

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

```
             $A[i] \leftrightarrow A[j]$ 
```

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

```
     $A[i] \leftrightarrow A[r]$ 
```

```
    return  $i$ 
```



Demo.

<https://algo.uni-trier.de/demos/sort.html>

Laufzeit

```
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  $\ell$ , int  $r$ )
```

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

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

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

```
    [ if  $A[j] \leq \text{pivot}$  then
      [  $A[i] \leftrightarrow A[j]$ 
         $i = i + 1$ 
      ]
    ]
```

```
   $A[i] \leftrightarrow A[r]$ 
```

```
  return  $i$ 
```

Zähle Anzahl der Vergleiche!

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

Wovon hängt dann die Laufzeit ab?

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

1. Extremfall: m immer erstes Element

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

2. Extremfall: m immer mittleres Element

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

siehe MergeSort

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 = $\binom{n}{2} / 2$ $\in \Theta(n^2)$ Vergleiche

QUICKSORT: Bester Fall = $n \log_2 n$ $\in \Theta(n \log n)$ Vergleiche
Schlechtester Fall = $\binom{n}{2}$ $\in \Theta(n^2)$ Vergleiche
Durchschnittsfall = ?

Zurück zu QUICKSORT

Idee: Steck Zufall in den Algorithmus!

RANDOMIZEDPARTITION(A, ℓ, r)

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

$A[k] \leftrightarrow A[r]$

return PARTITION(A, ℓ, r)

int PARTITION(int[] A , int ℓ , int r)

$\text{pivot} = A[r]$

$i = \ell$

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

if $A[j] \leq \text{pivot}$ **then**

$A[i] \leftrightarrow A[j]$

$i = i + 1$

$A[i] \leftrightarrow A[r]$

return i



Demo.

<https://algo.uni-trier.de/demos/sort.html>

Zurück zu QUICKSORT

Idee: Steck Zufall in den Algorithmus!

RANDOMIZEDPARTITION(A, ℓ, r)

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

$A[k] \leftrightarrow A[r]$

return PARTITION(A, ℓ, r)

int PARTITION(int[] A , int ℓ , int r)

$pivot = A[r]$

$i = \ell$

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

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

$A[i] \leftrightarrow A[j]$

$i = i + 1$

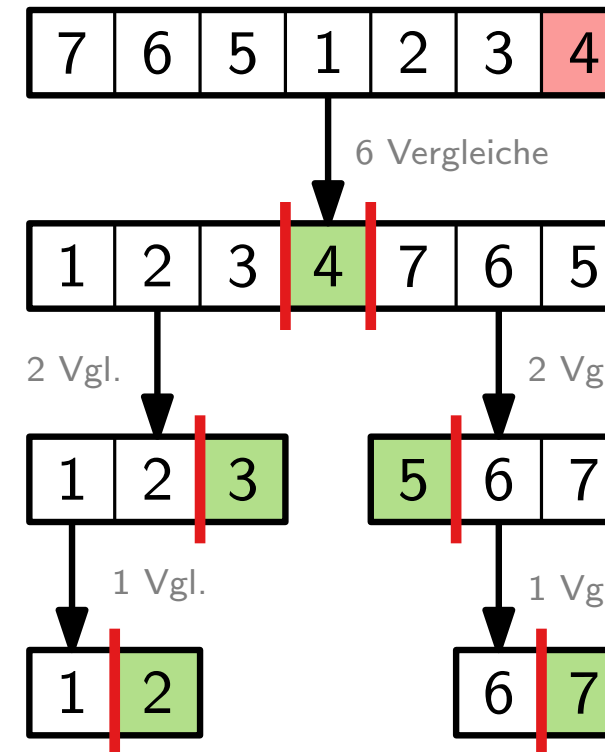
$A[i] \leftrightarrow A[r]$

return i

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.



Wie viele Vergleiche bräuchte INSERTIONSORT für diese Instanz?

$$0 + 1 + 2 + 3 + 4 + 4 + 4 = 18$$

Zurück zu QUICKSORT

Idee: Steck Zufall in den Algorithmus!

RANDOMIZEDPARTITION(A, ℓ, r)

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

$A[k] \leftrightarrow A[r]$

return PARTITION(A, ℓ, r)

int PARTITION(int[] A , int ℓ , int r)

$pivot = A[r]$

$i = \ell$

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

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

$A[i] \leftrightarrow A[j]$

$i = i + 1$

$A[i] \leftrightarrow A[r]$

return i

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 Zufallsvar. für Gesamtanzahl der Vergleiche.

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!

First come, first served

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

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 $z^* = z_j$.

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

Auf zum letzten Gefecht...

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

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

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

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

3) $\sum_{i=1}^n \frac{1}{i} \approx \ln n$ harmonische Reihe

Satz. RANDOMIZEDQUICKSORT sortiert n Zahlen in $\mathcal{O}(n \log n)$ erwarteter Zeit.

Vergleich Laufzeiten

	Bester Fall	Erw. Fall	Schl. Fall	in-situ	stabil
INSERTIONSORT	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✓
SELECTIONSORT	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✗
BUBBLESORT	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✓
MERGESORT	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✗	✓
HEAPSORT	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✓	✗
QUICKSORT	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	(✓)	✗

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.