



Julius-Maximilians-

**UNIVERSITÄT**  
**WÜRZBURG**

Lehrstuhl für

**INFORMATIK I**

Algorithmen & Komplexität



Institut für Informatik

# Algorithmen und Datenstrukturen

Wintersemester 2020/21  
8. Vorlesung

Sortieren – mit dem Würfel!

# Und noch einmal: Sortieren!

Zur Erinnerung: MergeSort...

+

-

# Und noch einmal: Sortieren!

Zur Erinnerung: MergeSort...

+ gute Worst-Case-Laufzeit

—

# Und noch einmal: Sortieren!

Zur Erinnerung: MergeSort...

- + gute Worst-Case-Laufzeit (durch Teile-und-Herrsche)
-

# Und noch einmal: Sortieren!

Zur Erinnerung: MergeSort...

- + gute Worst-Case-Laufzeit (durch Teile-und-Herrsche)
- kein in-situ-Verfahren

# Und noch einmal: Sortieren!

Zur Erinnerung: MergeSort...

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

# 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:**

# 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!

# 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:

*Teile:*

*Herrsche:*

*Kombiniere:*

# 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 Teilstück  $A[\ell..r]$  wie folgt:

*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 Teilstück kleiner gleich  $A[m]$  sind und alle im zweiten größer als  $A[m]$ .

*Herrsche:*

*Kombiniere:*

# 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 Teilstück  $A[\ell..r]$  wie folgt:

*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 Teilstück kleiner gleich  $A[m]$  sind und alle im zweiten größer als  $A[m]$ .

*Herrsche:* durch rekursives Sortieren der beiden Teilstücke.

*Kombiniere:*

# 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 Teilstück  $A[\ell..r]$  wie folgt:

**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 Teilstück kleiner gleich  $A[m]$  sind und alle im zweiten größer als  $A[m]$ .

**Herrsche:** durch rekursives Sortieren der beiden Teilstücke.

**Kombiniere:** —

# 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 Teilstück  $A[\ell..r]$  wie folgt:

*Teile:*

int

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

[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 Teilstück kleiner gleich  $A[m]$  sind und alle im zweiten größer als  $A[m]$ .

*Herrsche:*

durch rekursives Sortieren der beiden Teilstücke.

*Kombiniere:* —

# 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 Teilstück  $A[\ell..r]$  wie folgt: **QuickSort(int[] A, int  $\ell, r$ )**

*Teile:*

int

**Partition(A,  $\ell, r$ )**

[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 Teilstück kleiner gleich  $A[m]$  sind und alle im zweiten größer als  $A[m]$ .

*Herrsche:*

durch rekursives Sortieren der beiden Teilstücke.

*Kombiniere:* —

# 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 Teilstück  $A[\ell..r]$  wie folgt: **QuickSort(int[] A, int  $\ell, r$ )**

*Teile:*

int

**Partition(A,  $\ell, r$ )**

[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 Teilstück kleiner gleich  $A[m]$  sind und alle im zweiten größer als  $A[m]$ .

*Herrsche:*

durch rekursives Sortieren der beiden Teilstücke.

*Kombiniere:* —

*Schreiben Sie QuickSort in Pseudocode unter Verwendung von Partition(A,  $\ell, r$ )!*

# QuickSort

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

# QuickSort

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

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

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

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

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

# 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[]  $A, \ell, r$ )
```

# 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[]  $A, \ell, r$ )
```

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

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

```
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  $i$ 
```

# QuickSort

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

**if**  $\ell < r$  **then**

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

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

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

Was passiert hier?

Finden Sie ~~mit Ihrer Nachbarin~~  
eine textuelle Beschreibung des  
Algorithmus!

`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**  $i$

# QuickSort

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

**if**  $\ell < r$  **then**

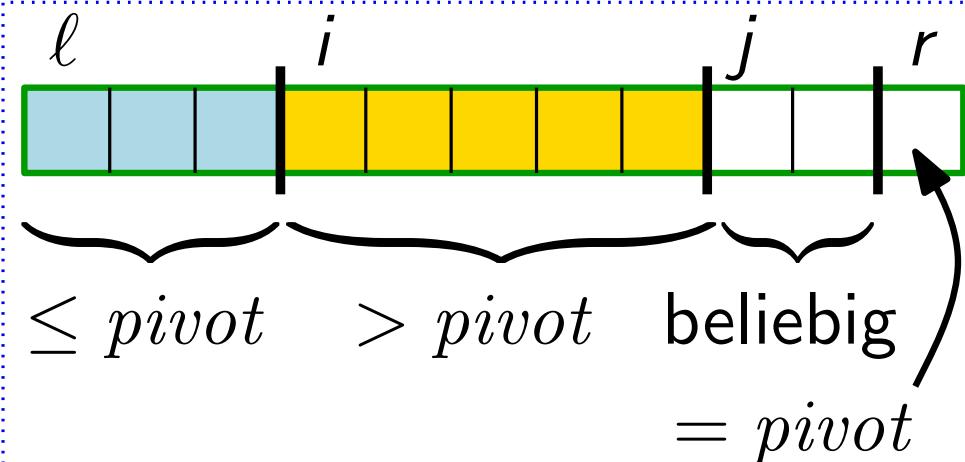
$m = \text{Partition}(A, \ell, r)$

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

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

Was passiert hier?

Finden Sie ~~mit Ihrer Nachbarin~~ eine textuelle Beschreibung des Algorithmus!



`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**  $i$

# QuickSort

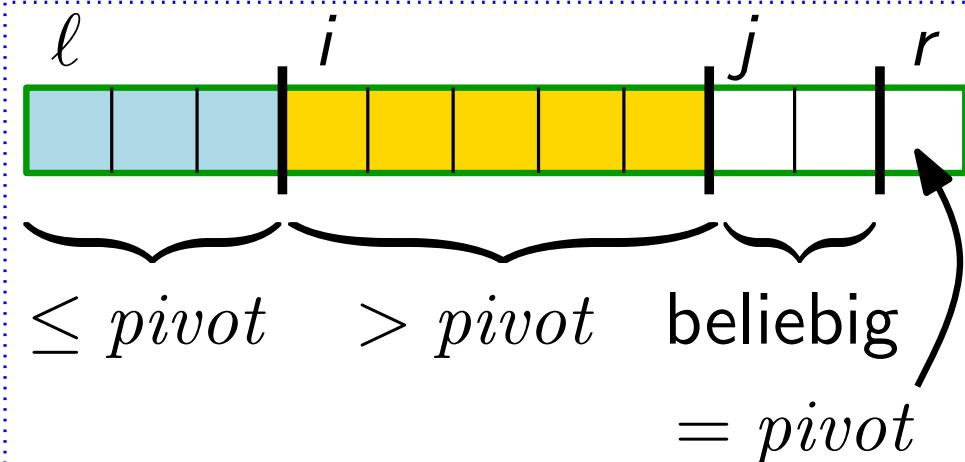
`QuickSort( $A, \ell = 1, r = A.length$ )`

**if**  $\ell < r$  **then**

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

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

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



**Schleifeninvariante:** —————

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

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

$i = \ell$

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

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

`Swap(A, i, j)`

$i = i + 1$

`Swap(A, i, r)`

**return**  $i$

# QuickSort

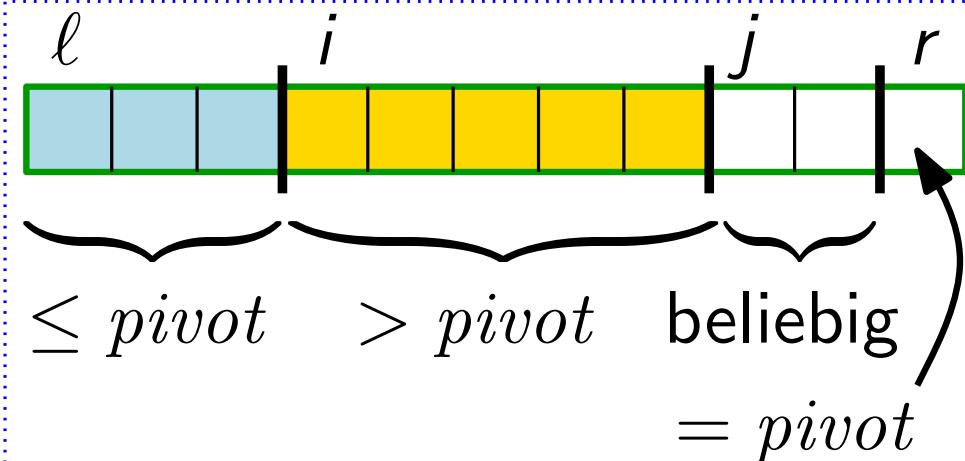
`QuickSort( $A, \ell = 1, r = A.length$ )`

**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  
 $A[k] \leq \text{pivot}$ .

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

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

$i = \ell$

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

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

`Swap(A, i, j)`

$i = i + 1$

`Swap(A, i, r)`

**return**  $i$

# QuickSort

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

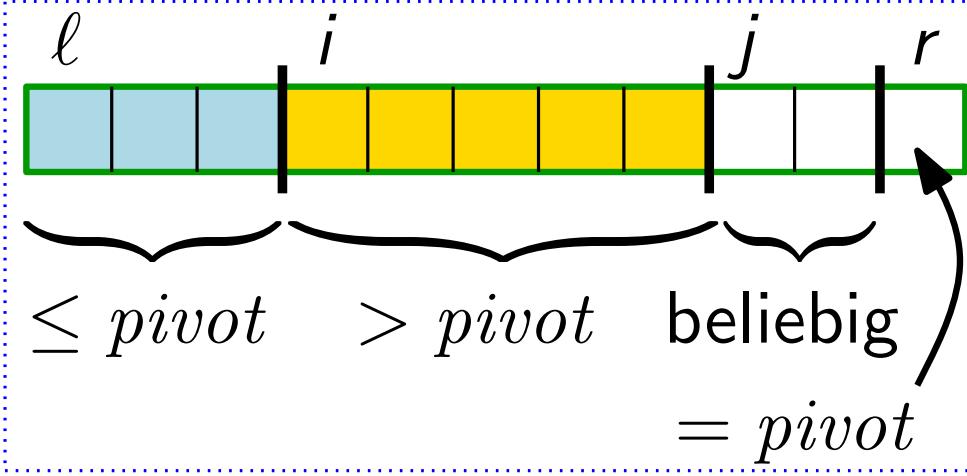
```

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  
 $A[k] > pivot$ .



`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

```

# QuickSort

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

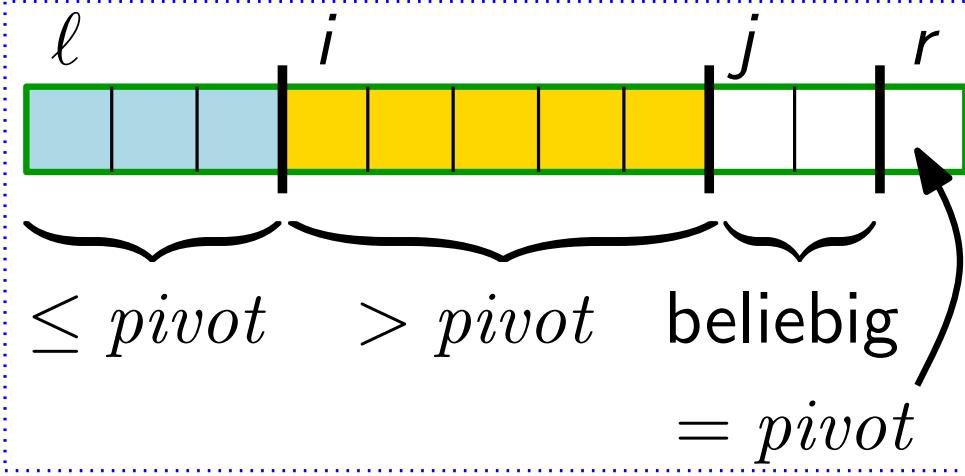
```

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)  $A[r] = pivot$ .



`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

```

# QuickSort

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

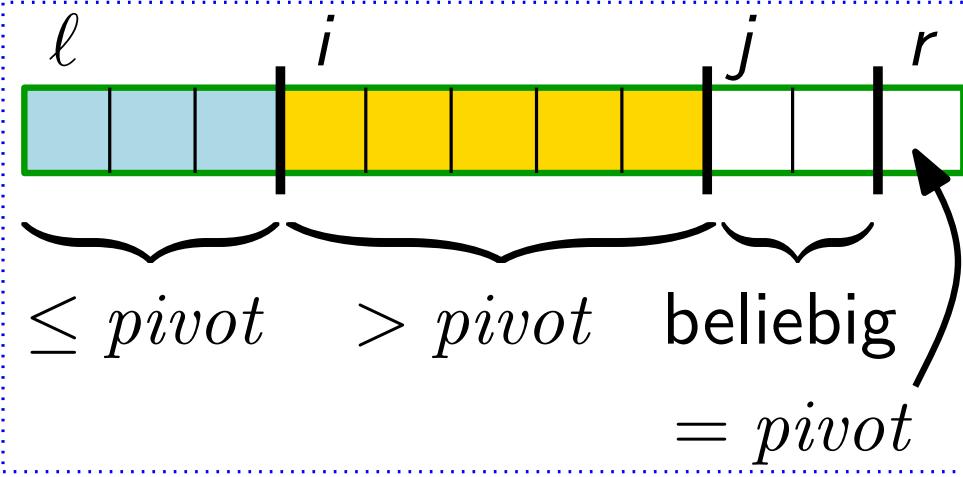
```

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

87 90 72 53 61 62 99

$\text{QuickSort}(A, \ell = 1, r = \dots)$

```
if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m-1)$ 
     $\text{QuickSort}(A, m+1, r)$ 
```

$\text{int Partition}(A, \ell, r)$

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

# Ein Beispiel

87 90 72 53 61 62 99  
 5 6 4 1 2 3 7

$\text{QuickSort}(A, \ell = 1, r = \dots)$

```
if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m-1)$ 
     $\text{QuickSort}(A, m+1, r)$ 
```

$\text{int Partition}(A, \ell, r)$

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

# Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | 7 | -

$\text{QuickSort}(A, \ell = 1, r = \dots)$

```
if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m-1)$ 
     $\text{QuickSort}(A, m+1, r)$ 
```

$\text{int Partition}(A, \ell, r)$

```
 $pivot = A[r]$ 
 $i = \ell$ 
for  $j = \ell$  to  $r - 1$  do
    if  $A[j] \leq pivot$  then
         $\text{Swap}(A, i, j)$ 
         $i = i + 1$ 
```

```
 $\text{Swap}(A, i, r)$ 
return  $i$ 
```

# Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | 7 | -



$\text{QuickSort}(A, \ell = 1, r = \dots)$

```

if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m-1)$ 
     $\text{QuickSort}(A, m+1, r)$ 

```

$\text{int Partition}(A, \ell, r)$

```

    pivot = A[r]
    i = ℓ
    for  $j = \ell$  to  $r - 1$  do
        if  $A[j] \leq \text{pivot}$  then
            Swap(A, i, j)
            i = i + 1

```

```

    Swap(A, i, r)
    return i

```

# Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | 7 | -



1 2 | 3 | 5 6 4

$\text{QuickSort}(A, \ell = 1, r = \dots)$

```
if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m-1)$ 
     $\text{QuickSort}(A, m+1, r)$ 
```

$\text{int Partition}(A, \ell, r)$

$pivot = A[r]$

$i = \ell$

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

```
if  $A[j] \leq pivot$  then
     $\text{Swap}(A, i, j)$ 
     $i = i + 1$ 
```

$\text{Swap}(A, i, r)$

**return**  $i$

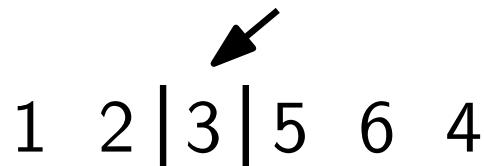
# Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | 7 | -

1 2 | 3 | 5 6 4



$\text{QuickSort}(A, \ell = 1, r = \dots)$

```

if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m-1)$ 
     $\text{QuickSort}(A, m+1, r)$ 

```

$\text{int Partition}(A, \ell, r)$

```

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

```

```

    Swap(A, i, r)
    return i

```

# Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | 7 | -



1 2 | 3 | 5 6 4



1 | 2 | -

$\text{QuickSort}(A, \ell = 1, r = \dots)$

```

if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m-1)$ 
     $\text{QuickSort}(A, m+1, r)$ 

```

$\text{int Partition}(A, \ell, r)$

$pivot = A[r]$

$i = \ell$

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

```

        if  $A[j] \leq pivot$  then
             $\text{Swap}(A, i, j)$ 
             $i = i + 1$ 

```

$\text{Swap}(A, i, r)$

**return**  $i$

# Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | 7 | -

1 2 | 3 | 5 6 4  
 ↙      ↘  
 1 | 2 | -

$\text{QuickSort}(A, \ell = 1, r = \dots)$

```
if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m-1)$ 
     $\text{QuickSort}(A, m+1, r)$ 
```

$\text{int Partition}(A, \ell, r)$

```
 $pivot = A[r]$ 
 $i = \ell$ 
for  $j = \ell$  to  $r - 1$  do
    if  $A[j] \leq pivot$  then
         $\text{Swap}(A, i, j)$ 
         $i = i + 1$ 
```

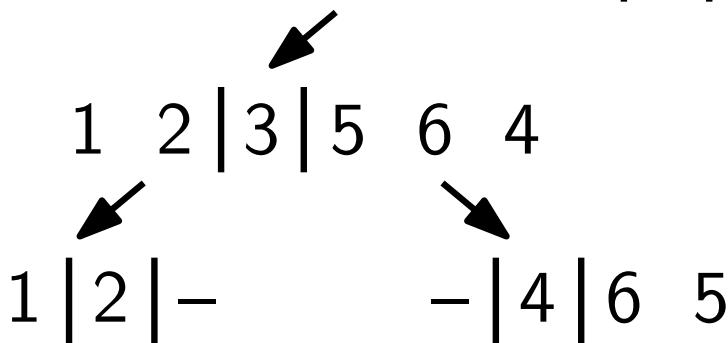
```
 $\text{Swap}(A, i, r)$ 
return  $i$ 
```

# Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | 7 | -



$\text{QuickSort}(A, \ell = 1, r = \dots)$

```

if  $\ell < r$  then
   $m = \text{Partition}(A, \ell, r)$ 
   $\text{QuickSort}(A, \ell, m-1)$ 
   $\text{QuickSort}(A, m+1, r)$ 
  
```

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

```

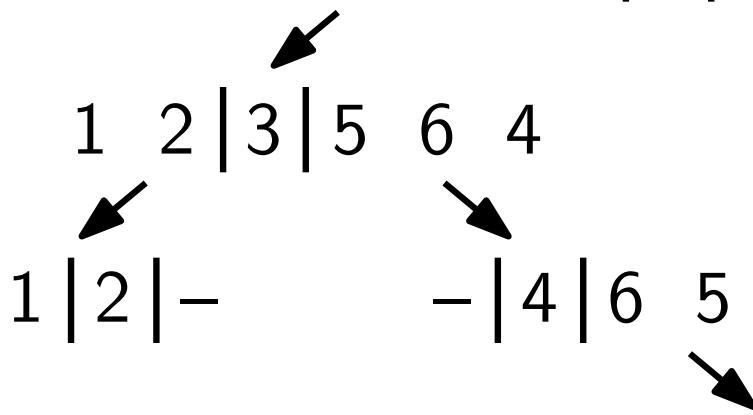
  pivot =  $A[r]$ 
  i =  $\ell$ 
  for  $j = \ell$  to  $r - 1$  do
    if  $A[j] \leq \text{pivot}$  then
      Swap( $A, i, j$ )
       $i = i + 1$ 
  Swap( $A, i, r$ )
  return  $i$ 
  
```

# Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | 7 | -



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( $A, i, j$ )
     $i = i + 1$ 
```

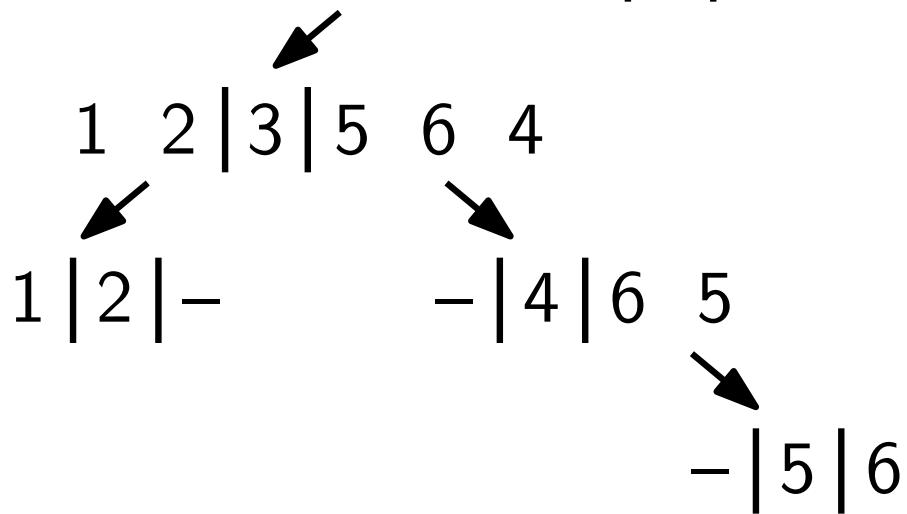
```
Swap( $A, i, r$ )
return  $i$ 
```

# Ein Beispiel

87 90 72 53 61 62 99

5 6 4 1 2 3 7

5 6 4 1 2 3 | 7 | -



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

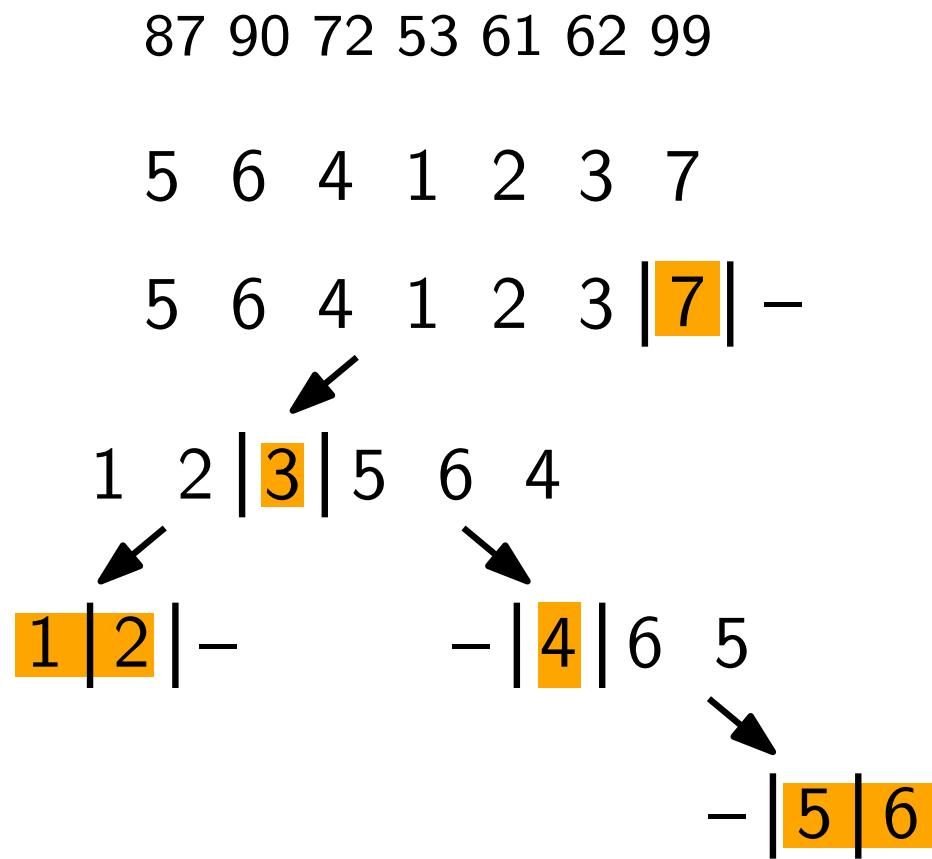
```
if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
    QuickSort( $A, \ell, m-1$ )
    QuickSort( $A, m+1, r$ )
```

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  $i$ 
```

# Ein Beispiel



$\text{QuickSort}(A, \ell = 1, r = \dots)$

```

if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m-1)$ 
     $\text{QuickSort}(A, m+1, r)$ 

```

$\text{int Partition}(A, \ell, r)$

```

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

```

```

     $\text{Swap}(A, i, r)$ 
    return  $i$ 

```

# Laufzeit

```
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  $A[j] \leq \text{pivot}$  then
      Swap( $A, i, j$ )
      i =  $i + 1$ 
```

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

# Laufzeit

Zähle Anzahl der Vergleiche!

$\text{QuickSort}(A, \ell = 1, r = \dots)$

```

if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m - 1)$ 
     $\text{QuickSort}(A, m + 1, r)$ 

```

$\text{int Partition}(A, \ell, r)$

```

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

```

```

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

```

# Laufzeit

Zähle Anzahl der Vergleiche!

$\text{QuickSort}(A, \ell = 1, r = \dots)$

```
if  $\ell < r$  then
     $m = \text{Partition}(A, \ell, r)$ 
     $\text{QuickSort}(A, \ell, m - 1)$ 
     $\text{QuickSort}(A, m + 1, r)$ 
```

$\text{int Partition}(A, \ell, r)$

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

# Laufzeit

Zähle Anzahl der Vergleiche!

**Beob.** Partition benötigt *immer* Vergleiche.

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  $A[j] \leq \text{pivot}$  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.

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  $A[j] \leq \text{pivot}$  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?

$$T_{QS}(n) =$$

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  $A[j] \leq \text{pivot}$  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?

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

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  $A[j] \leq \text{pivot}$  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

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

$$T_{QS}(n) = T_{QS}(0) + T_{QS}(n-1) + n-1$$

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

$$T_{QS}(n) = T_{QS}(0) + T_{QS}(n-1) + n-1$$

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

$$T_{QS}(n) = T_{QS}(0) + T_{QS}(n-1) + n-1$$

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

$$T_{QS}(n) = T_{QS}(0) + T_{QS}(n-1) + n-1$$

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

$$\begin{aligned} T_{QS}(n) &= T_{QS}(0) + T_{QS}(n-1) + n-1 \\ &= (T_{QS}(n-2) + n-2) + n-1 \end{aligned}$$

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

$$\begin{aligned} T_{QS}(n) &= T_{QS}(0) + T_{QS}(n-1) + n-1 \\ &= (T_{QS}(n-2) + n-2) + n-1 \end{aligned}$$

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

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

2. Extremfall:  $m$  immer mittleres Element

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

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

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

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

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

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

2. Extremfall:  $m$  immer mittleres Element

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  $A[j] \leq \text{pivot}$  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

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

2. Extremfall:  $m$  immer mittleres Element

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

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

2. Extremfall:  $m$  immer mittleres Element

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

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

2. Extremfall:  $m$  immer mittleres Element

$$T_{QS}(n) \approx$$

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

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

2. Extremfall:  $m$  immer mittleres Element

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

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

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

2. Extremfall:  $m$  immer mittleres Element

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

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

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

2. Extremfall:  $m$  immer mittleres Element

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

siehe MergeSort

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

```
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*

# Wo ist die Wahrheit?

# Wo ist die Wahrheit?

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

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

# 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 = ?	

# 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 = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

# 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 = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!  
*Schwierig...*

# 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 = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

# 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 = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

Berechne *erwartete* Laufzeit  $E[T_{IS}]$  einer *zufälligen* Permutation

# 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 = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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

# 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 = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

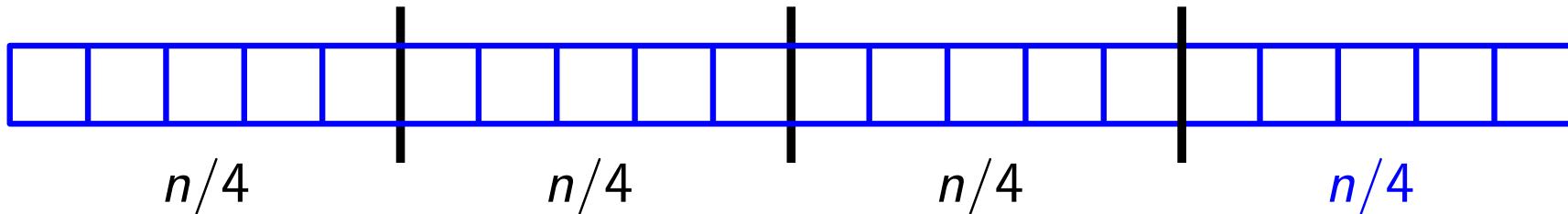
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechtester Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

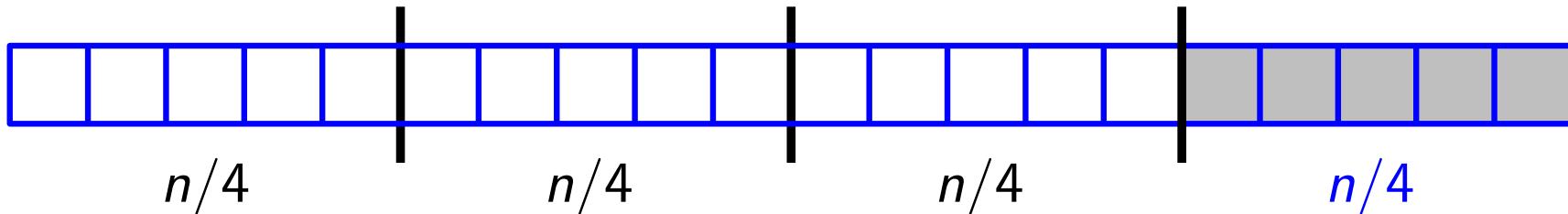
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechtester Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

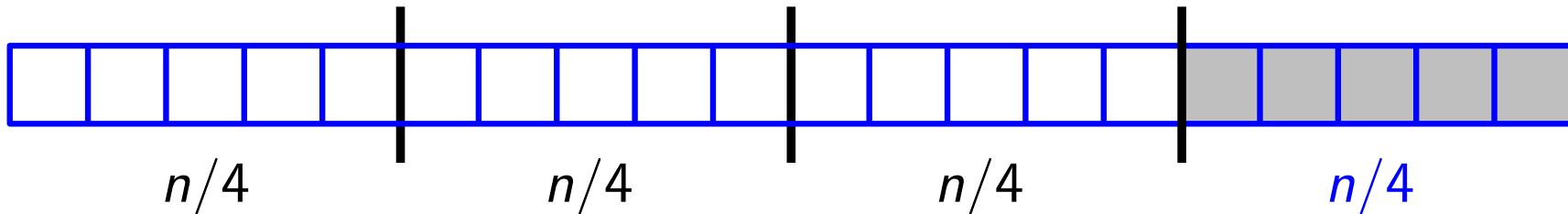
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechtester Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq E[\text{Aufwand für letzte } \frac{n}{4} \text{ Elem.}] \geq$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

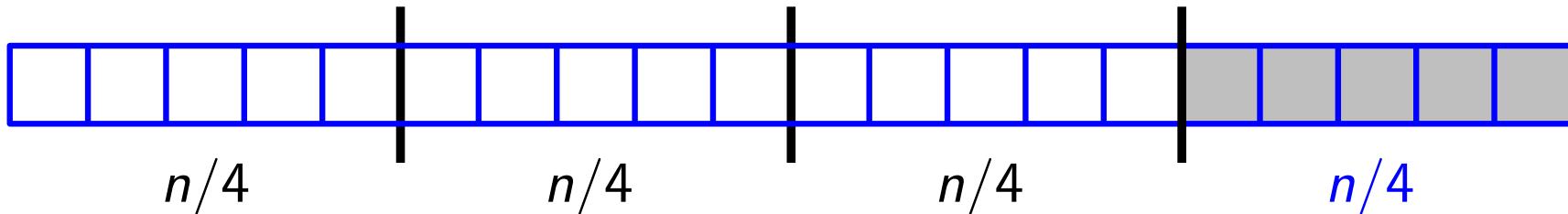
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechtester Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq E[\text{Aufwand für letzte } \frac{n}{4} \text{ Elem.}] \geq \frac{n}{4}.$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

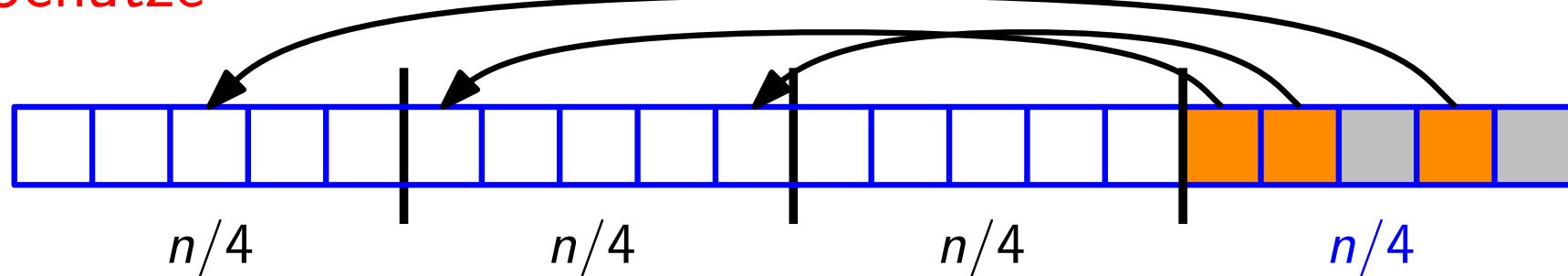
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechterer Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq E[\text{Aufwand für letzte } \frac{n}{4} \text{ Elém.}] \geq \frac{n}{4}.$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

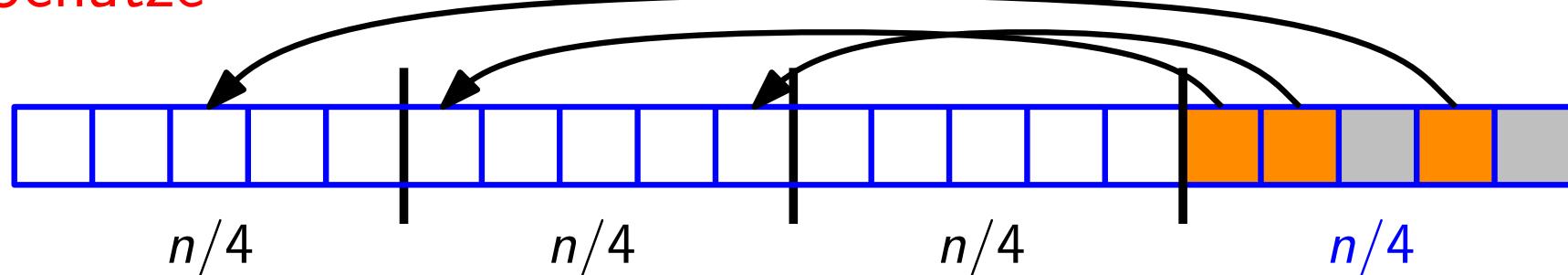
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechterer Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq E[\text{Aufwand für letzte } \frac{n}{4} \text{ Elém.}] \geq \frac{n}{4} \cdot \frac{1}{2}$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

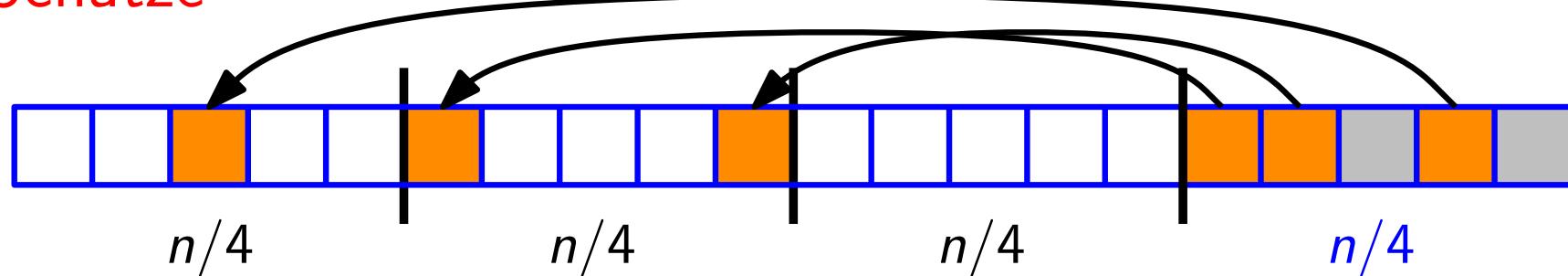
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechterer Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq E[\text{Aufwand für letzte } \frac{n}{4} \text{ Elém.}] \geq \frac{n}{4} \cdot \frac{1}{2}$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

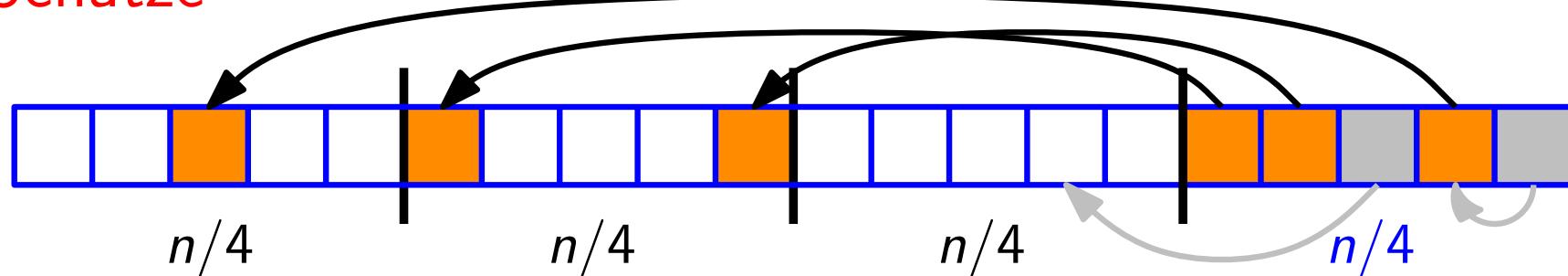
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechterer Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq E[\text{Aufwand für letzte } \frac{n}{4} \text{ Elém.}] \geq \frac{n}{4} \cdot \frac{1}{2}$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

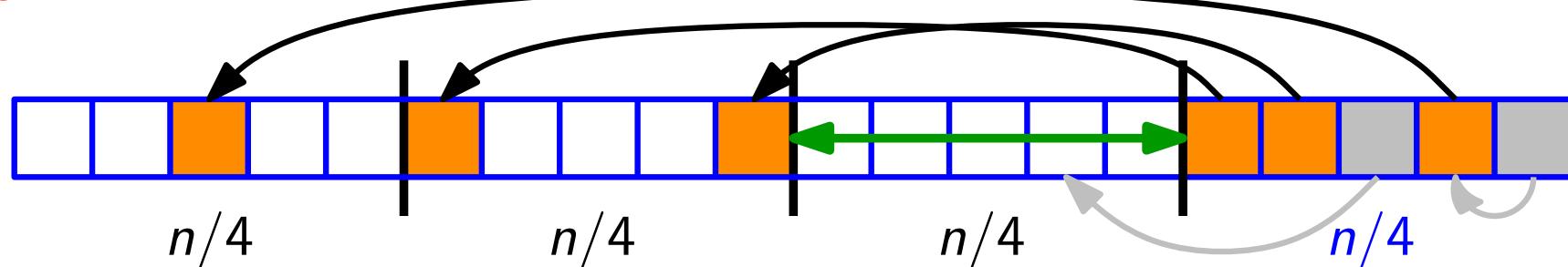
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechtester Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq E[\text{Aufwand für letzte } \frac{n}{4} \text{ Elém.}] \geq \frac{n}{4} \cdot \frac{1}{2}$$

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

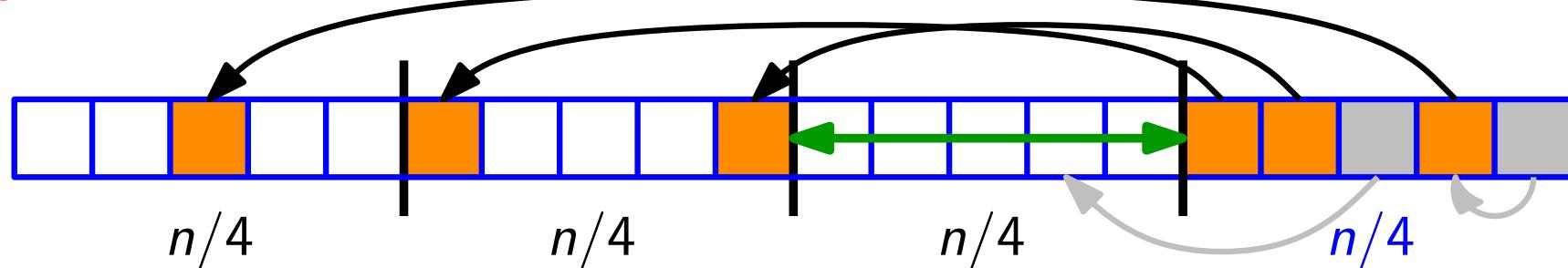
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechtester Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



$$E[T_{IS}] \geq E[\text{Aufwand für letzte } \frac{n}{4} \text{ Elém.}] \geq \frac{n}{4} \cdot \frac{1}{2} \cdot \frac{n}{4}$$

# Wo ist die Wahrheit?

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

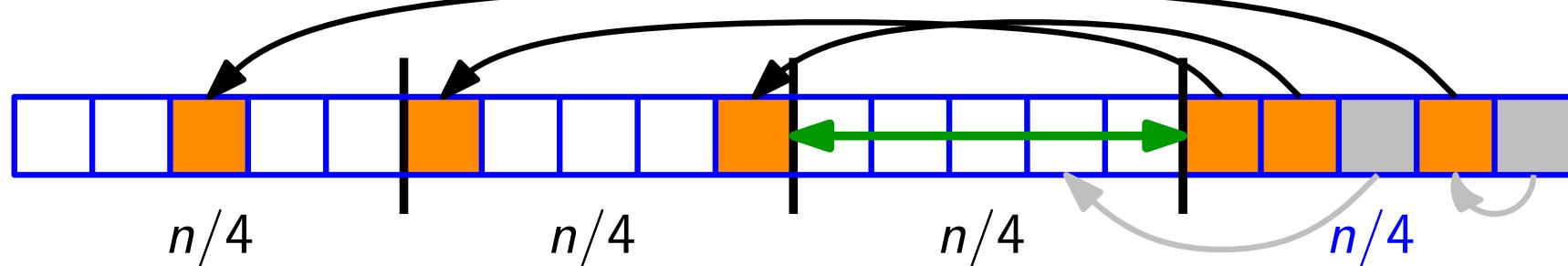
Vgl. InsertionSort:

Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechterer Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = ?	

Mittle die Laufzeit über alle Permutationen der Eingabe!  
*Schwierig...*

*Statt dessen:*

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



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

# Wo ist die Wahrheit?

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

Vgl. InsertionSort:

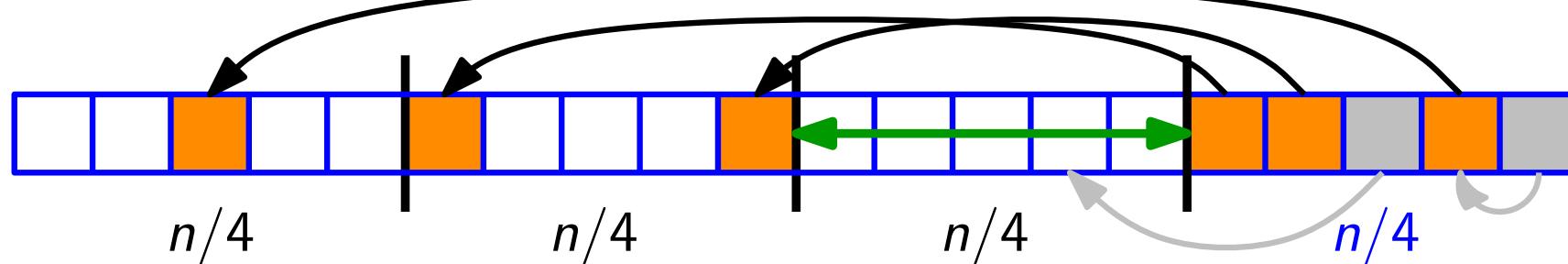
Bester Fall = $n - 1 \in \Theta(n)$	Vergleiche
Schlechterer Fall = $\binom{n}{2} \in \Theta(n^2)$	Vergleiche
Durchschnittsfall = $\Theta(n^2)$	

Mittle die Laufzeit über alle Permutationen der Eingabe!

*Schwierig...*

*Statt dessen:*

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



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

# Zurück zu QuickSort

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**  $i$

# Zurück zu QuickSort

Idee: Steck Zufall in den Algorithmus!

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**  $i$

# Zurück zu QuickSort

Idee: Steck Zufall in den Algorithmus!

RandomizedPartition( $A, \ell, r$ )

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*

# Zurück zu QuickSort

Idee: Steck Zufall in den Algorithmus!

**RandomizedPartition( $A, \ell, r$ )**

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

$\text{Swap}(A, r, k)$

**return** Partition( $A, \ell, r$ )

**Partition( $A, \ell, r$ )**

$pivot = A[r]$

$i = \ell$

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

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

$\text{Swap}(A, i, j)$

$i = i + 1$

$\text{Swap}(A, i, r)$

**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.

**RandomizedPartition( $A, \ell, r$ )**

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

**Swap( $A, r, k$ )**

**return** Partition( $A, \ell, r$ )

**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*

# 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  $z_j$ ?

**RandomizedPartition( $A, \ell, r$ )**

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

**Swap( $A, r, k$ )**

**return** Partition( $A, \ell, r$ )

**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**  $i$

# 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:

**RandomizedPartition( $A, \ell, r$ )**

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

**Swap( $A, r, k$ )**

**return Partition( $A, \ell, r$ )**

**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**  $i$

# 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( $A, \ell, r$ )**

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

**Swap( $A, r, k$ )**

**return** Partition( $A, \ell, r$ )

**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**  $i$

# 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.

5	6	4	1	2	3	7
---	---	---	---	---	---	---

**RandomizedPartition( $A, \ell, r$ )**

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

**Swap( $A, r, k$ )**

**return** Partition( $A, \ell, r$ )

**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**  $i$

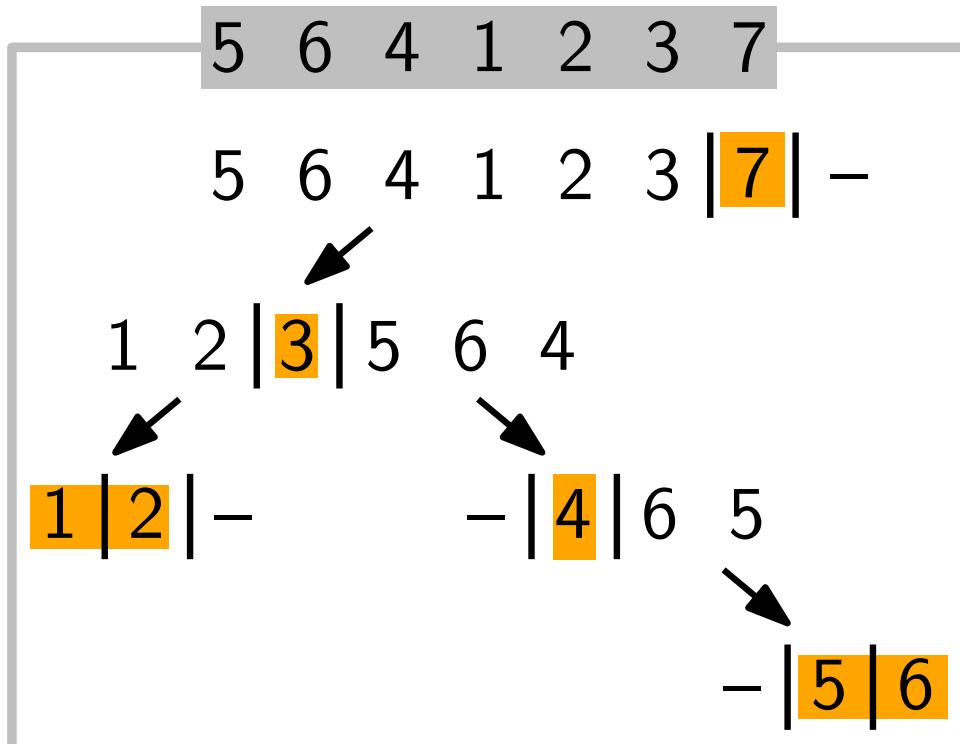
# Zurück zu QuickSort

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( $A, \ell, r$ )

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

Swap( $A, r, k$ )

**return** Partition( $A, \ell, r$ )

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**  $i$

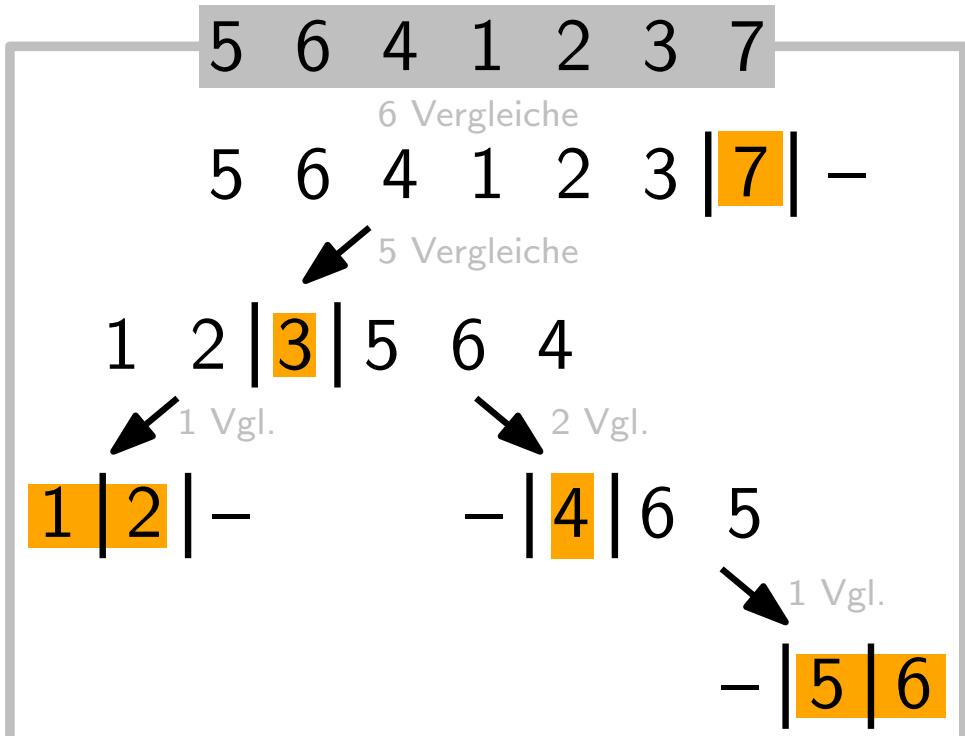
# Zurück zu QuickSort

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( $A, \ell, r$ )

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

Swap( $A, r, k$ )

**return** Partition( $A, \ell, r$ )

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**  $i$

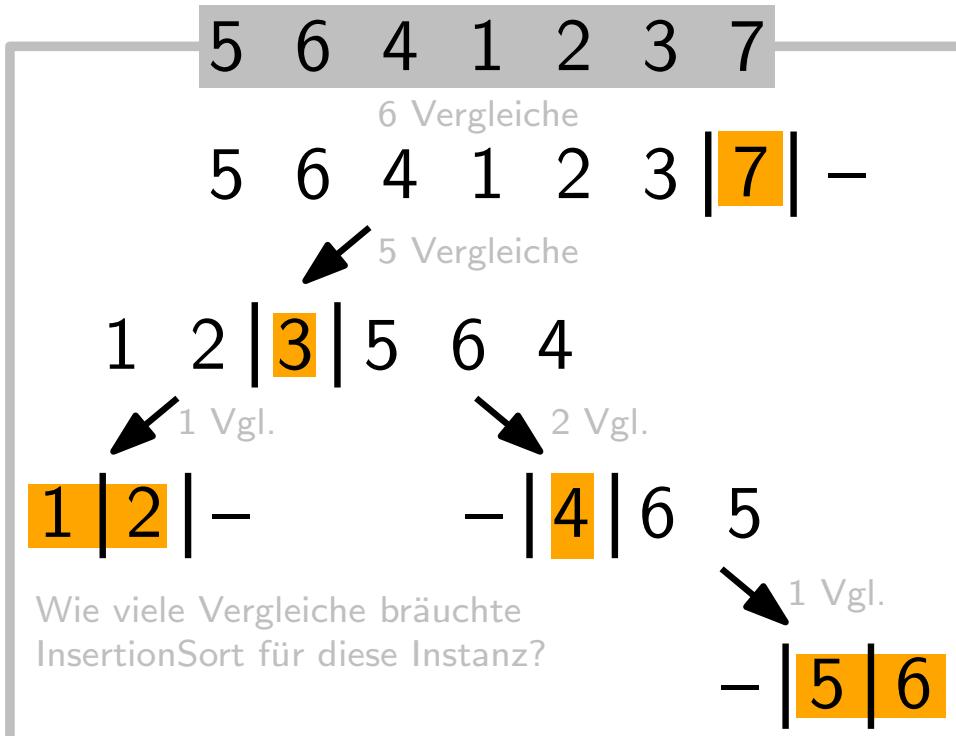
# Zurück zu QuickSort

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( $A, \ell, r$ )

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

Swap( $A, r, k$ )

**return** Partition( $A, \ell, r$ )

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**  $i$

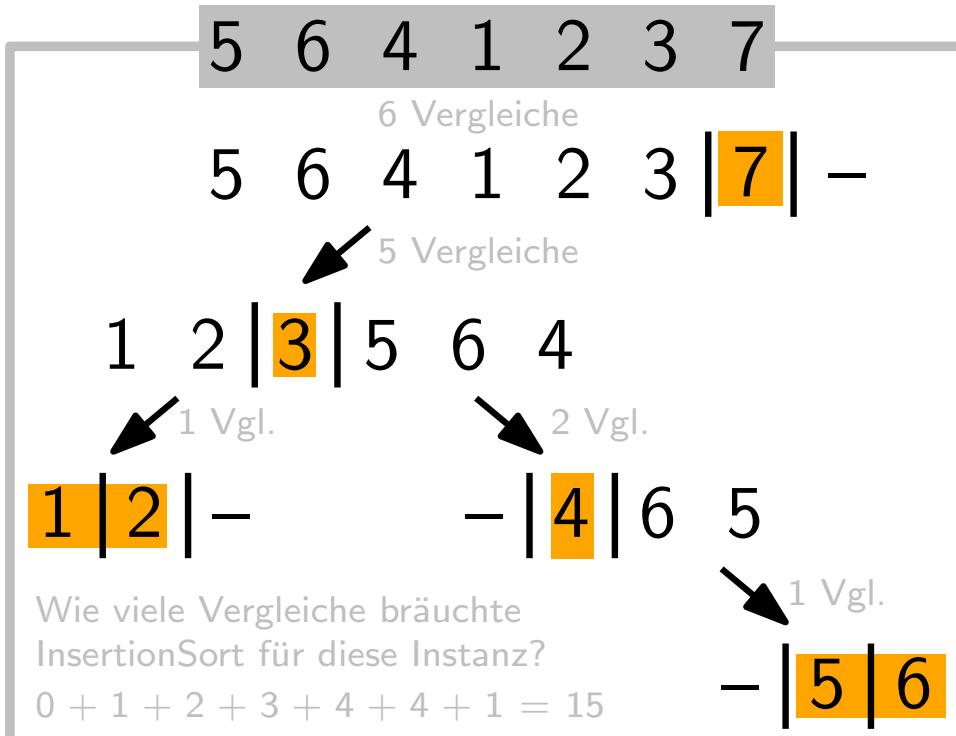
# Zurück zu QuickSort

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( $A, \ell, r$ )

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

Swap( $A, r, k$ )

**return** Partition( $A, \ell, r$ )

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**  $i$

# 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:

**RandomizedPartition( $A, \ell, r$ )**

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

**Swap( $A, r, k$ )**

**return Partition( $A, \ell, r$ )**

**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**  $i$

# 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:

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

RandomizedPartition( $A, \ell, r$ )

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

Swap( $A, r, k$ )

**return** Partition( $A, \ell, r$ )

Partition( $A, \ell, r$ )

*pivot* =  $A[r]$

$i = \ell$

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

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

RandomizedPartition( $A, \ell, r$ )

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

Swap( $A, r, k$ )

**return** Partition( $A, \ell, r$ )

Partition( $A, \ell, r$ )

*pivot* =  $A[r]$

$i = \ell$

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

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

Dann gilt  $V =$

RandomizedPartition( $A, \ell, r$ )

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

Swap( $A, r, k$ )

**return** Partition( $A, \ell, r$ )

Partition( $A, \ell, r$ )

*pivot* =  $A[r]$

$i = \ell$

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

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

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

RandomizedPartition( $A, \ell, r$ )

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

Swap( $A, r, k$ )

**return** Partition( $A, \ell, r$ )

Partition( $A, \ell, r$ )

*pivot* =  $A[r]$

$i = \ell$

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

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

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

$$\Rightarrow E[V] =$$

RandomizedPartition( $A, \ell, r$ )

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

Swap( $A, r, k$ )

**return** Partition( $A, \ell, r$ )

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**  $i$

# 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.

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

$$\Rightarrow E[V] =$$



Linearität des Erwartungswerts!

**RandomizedPartition( $A, \ell, r$ )**

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

**Swap( $A, r, k$ )**

**return** Partition( $A, \ell, r$ )

**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**  $i$

# 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.

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( $A, \ell, r$ )**

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

**Swap( $A, r, k$ )**

**return** Partition( $A, \ell, r$ )

**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**  $i$

# First come, first serve

$$E[V_{ij}] =$$

# First come, first serve

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

# First come, first serve

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

# First come, first serve

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

# First come, first serve

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

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

# First come, first serve

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

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

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

# First come, first serve

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

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  $z_j \Leftrightarrow$

# First come, first serve

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

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  $z_j \Leftrightarrow z^* = z_i$  oder  $z^* = z_j$ .

# First come, first serve

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

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  $z_j \Leftrightarrow z^* = z_i$  oder  $z^* = z_j$ .

$$\Rightarrow \Pr[\text{Alg. vergleicht } z_i \text{ und } z_j] =$$

# First come, first serve

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

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  $z_j \Leftrightarrow z^* = z_i$  oder  $z^* = z_j$ .

$$\Rightarrow \Pr[\text{Alg. vergleicht } z_i \text{ und } z_j] = \Pr[z^* = z_i \text{ oder } z^* = z_j]$$

# First come, first serve

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

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  $z_j \Leftrightarrow z^* = z_i$  oder  $z^* = z_j$ .

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

# First come, first serve

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

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  $z_j \Leftrightarrow z^* = z_i$  oder  $z^* = z_j$ .

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

# First come, first serve

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

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  $z_j \Leftrightarrow z^* = z_i$  oder  $z^* = z_j$ .

$$\Rightarrow \Pr[\text{Alg. vergleicht } z_i \text{ und } z_j] = \Pr[z^* = z_i \text{ oder } z^* = z_j]$$

$$= \Pr[z^* = z_i] + \Pr[z^* = z_j]$$

$$= \frac{1}{|Z_{ij}|} + \frac{1}{|Z_{ij}|}$$

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

# First come, first serve

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

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  $z_j \Leftrightarrow z^* = z_i$  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 . . .

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

$$E[V] =$$

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

$$\begin{aligned} E[V] &= \sum_{1 \leq i < j \leq n} E[V_{ij}] = \sum_{1 \leq i < j \leq n} \frac{2}{j - i + 1} \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k! \end{aligned}$$

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k!$$

$$= \sum_{i=1}^{n-1}$$

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k!$$

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

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

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

**Trick:** ersetze  $j - i$  durch  $k$ !

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

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

**Trick:** ersetze  $j - i$  durch  $k$ !

$-i$

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$\begin{aligned} &= \sum_{i=1}^{n-1} \left( \sum_{j=i+1}^n \frac{2}{j - i + 1} \right) \\ &= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k + 1} \end{aligned}$$

**Trick:** ersetze  $j - i$  durch  $k$ !

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k!$$

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k!$$

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k!$$

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k!$$

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k!$$

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

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k!$$

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

*harmonische Reihe!*

# Auf zum letzten Gefecht . . .

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

Wir wissen:

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

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j - i + 1} \quad \text{Trick: ersetze } j - i \text{ durch } k!$$

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

*harmonische Reihe!*

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

# Zusammenfassung Sortierverfahren

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

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

<sup>2</sup>) Sortieralg. *stabil*, wenn er gleiche Schlüssel in Ursprungsreihenf. belässt.

# Zusammenfassung Sortierverfahren

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

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

<sup>2</sup>) Sortieralg. *stabil*, wenn er gleiche Schlüssel in Ursprungsreihenf. belässt.

# 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	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$	
Best-Case-Laufzeit	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	
in situ <sup>1</sup> ( <i>in place</i> )	✓	✗	✓	
stabil <sup>2</sup>	✓	✓	✗	

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

<sup>2</sup>) Sortieralg. *stabil*, wenn er gleiche Schlüssel in Ursprungsreihenf. belässt.

# 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	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	
in situ <sup>1</sup> ( <i>in place</i> )	✓	✗	✓	
stabil <sup>2</sup>	✓	✓	✗	

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

<sup>2</sup>) Sortieralg. *stabil*, wenn er gleiche Schlüssel in Ursprungsreihenf. belässt.

# 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 <sup>1</sup> ( <i>in place</i> )	✓	✗	✓	
stabil <sup>2</sup>	✓	✓	✗	

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

<sup>2</sup>) Sortieralg. *stabil*, wenn er gleiche Schlüssel in Ursprungsreihenf. belässt.

# 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 <sup>1</sup> ( <i>in place</i> )	✓	✗	✓	( ✓ )*
stabil <sup>2</sup>	✓	✓	✗	

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

<sup>2</sup>) 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.

# 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 <sup>1</sup> ( <i>in place</i> )	✓	✗	✓	( ✓ )*
stabil <sup>2</sup>	✓	✓	✗	✗

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

<sup>2</sup>) 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.