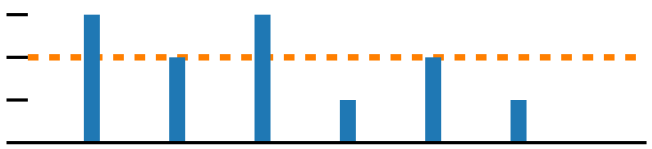
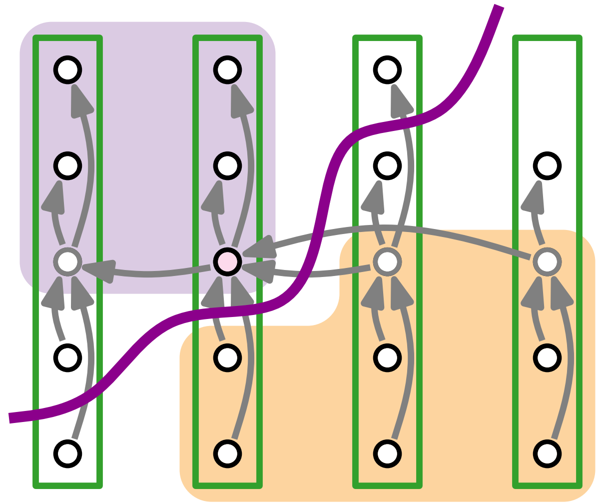
# Algorithmen und Datenstrukturen

## Vorlesung 10:
## Das Auswahlproblem





*Alexander Wolff*

*Wintersemester 2024*

# Analyse von Messreihen

# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$

# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \ldots n]$

**Gesucht:** Ein „guter“ Mittelwert

# Analyse von Messreihen

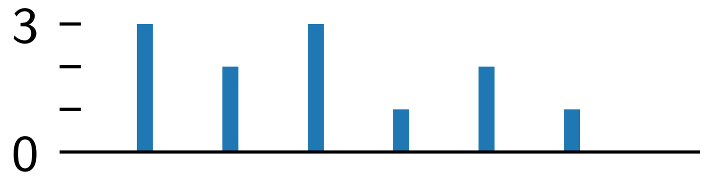
**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Ein „guter“ Mittelwert

**Beispiel.**

# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Ein „guter" Mittelwert

**Beispiel.**

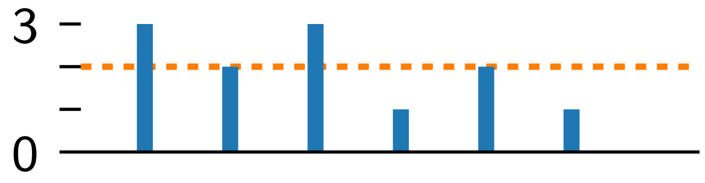


# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$

**Gesucht:** Ein „guter“ Mittelwert

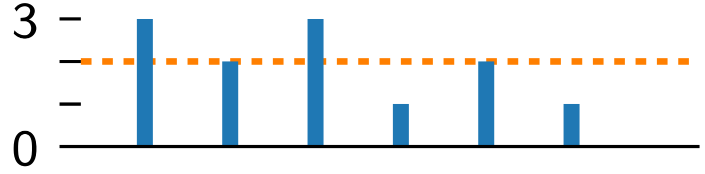
**Beispiel.**



# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \ldots n]$
**Gesucht:** Ein „guter“ Mittelwert

**Beispiel.**

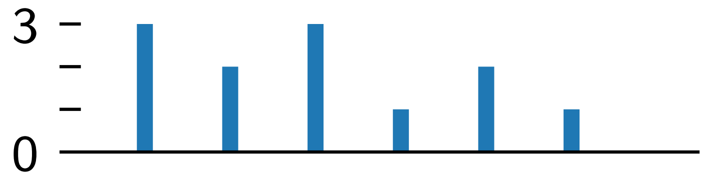



**arithmetisches Mittel**
$\frac{1}{n} \sum_{i=1}^{n} A[i]$

# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \ldots n]$
**Gesucht:** Ein „guter“ Mittelwert

**Beispiel.**

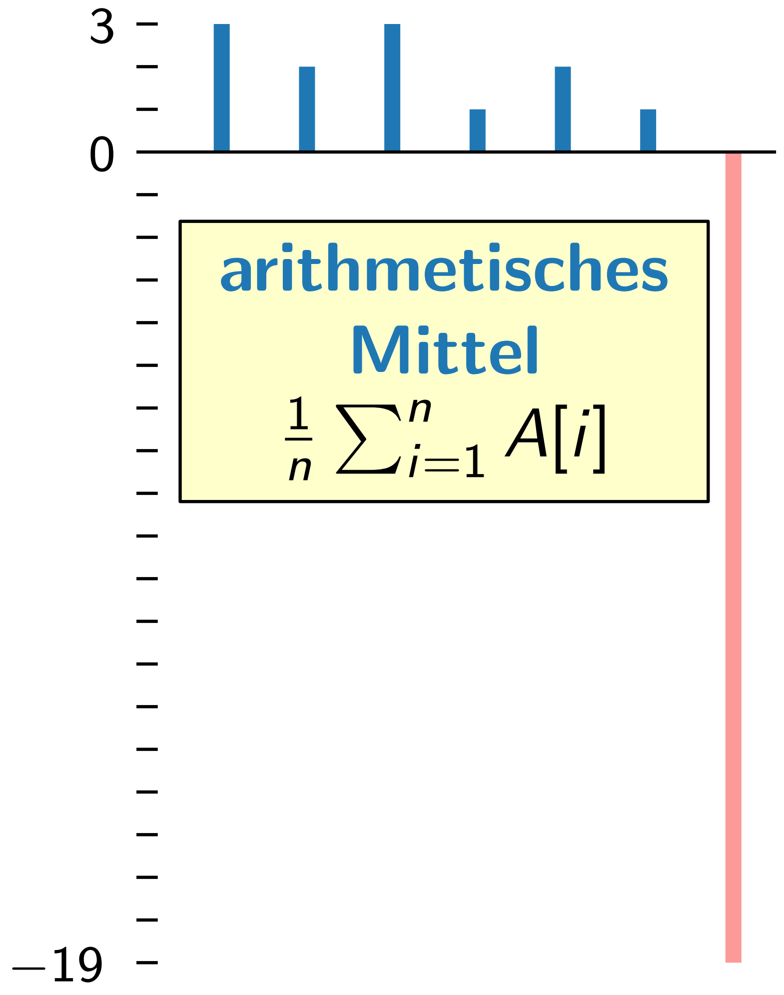



**arithmetisches Mittel**
$\frac{1}{n}\sum_{i=1}^{n} A[i]$

# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Ein „guter“ Mittelwert

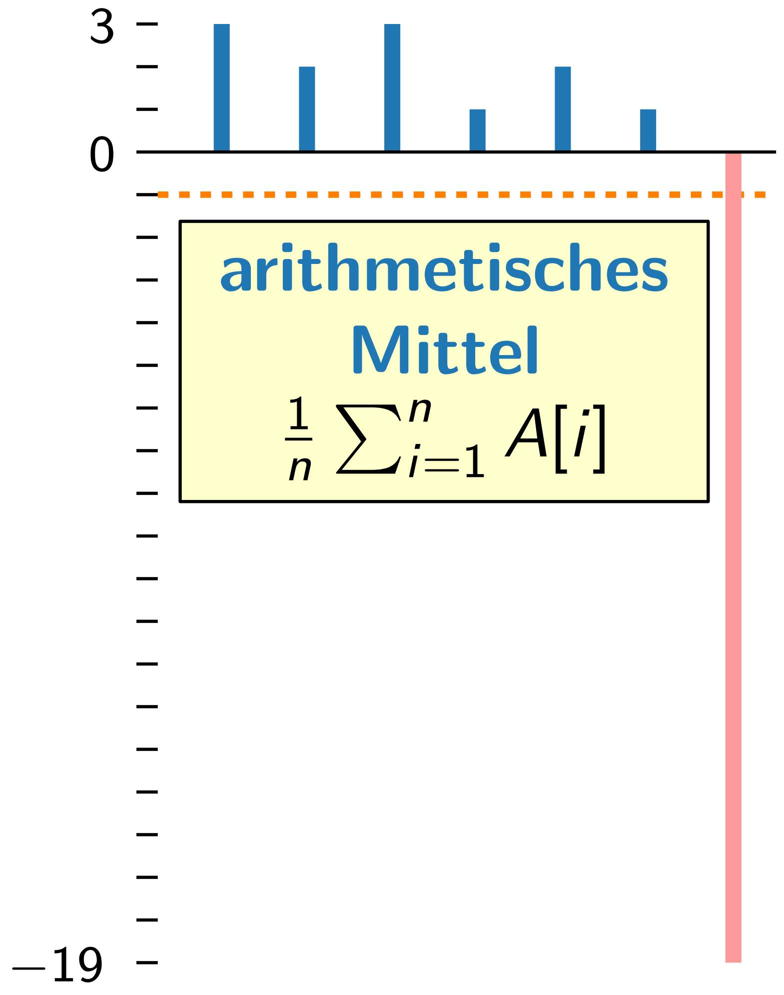
**Beispiel.**



# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \ldots n]$
**Gesucht:** Ein „guter“ Mittelwert

**Beispiel.**

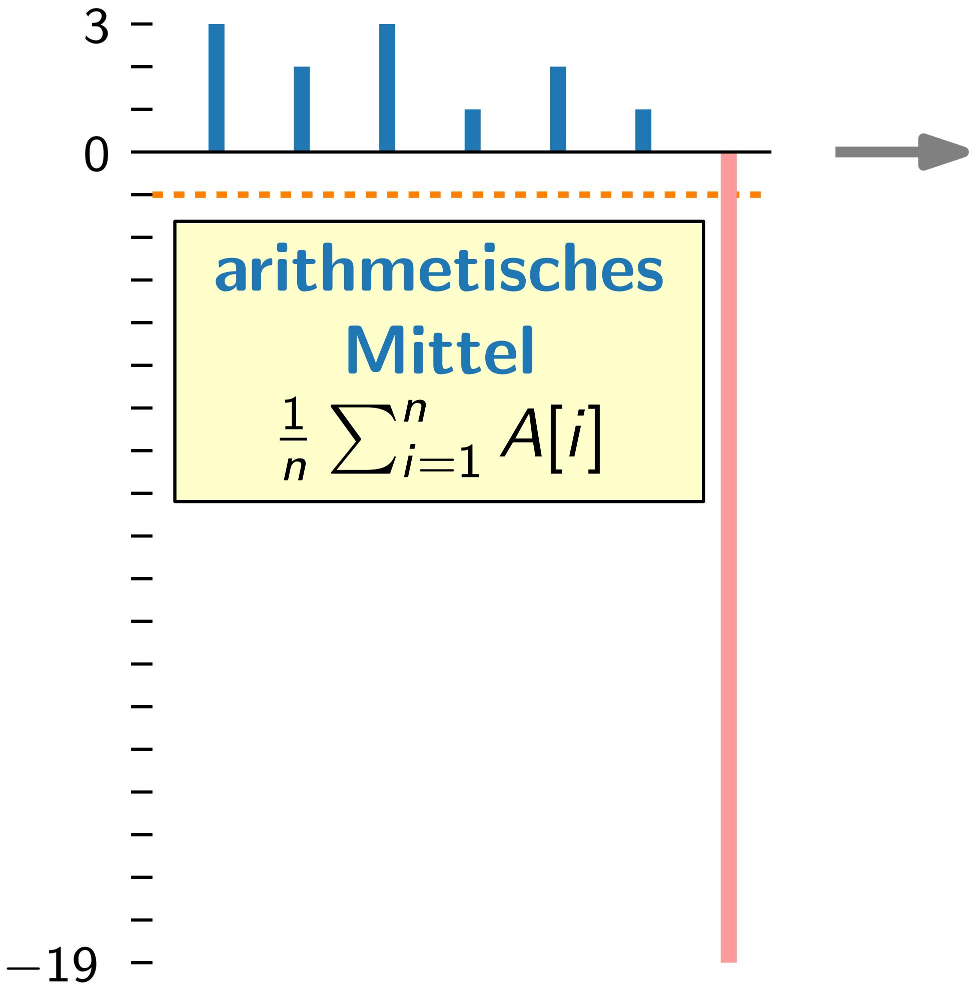


# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \ldots n]$

**Gesucht:** Ein „guter“ Mittelwert

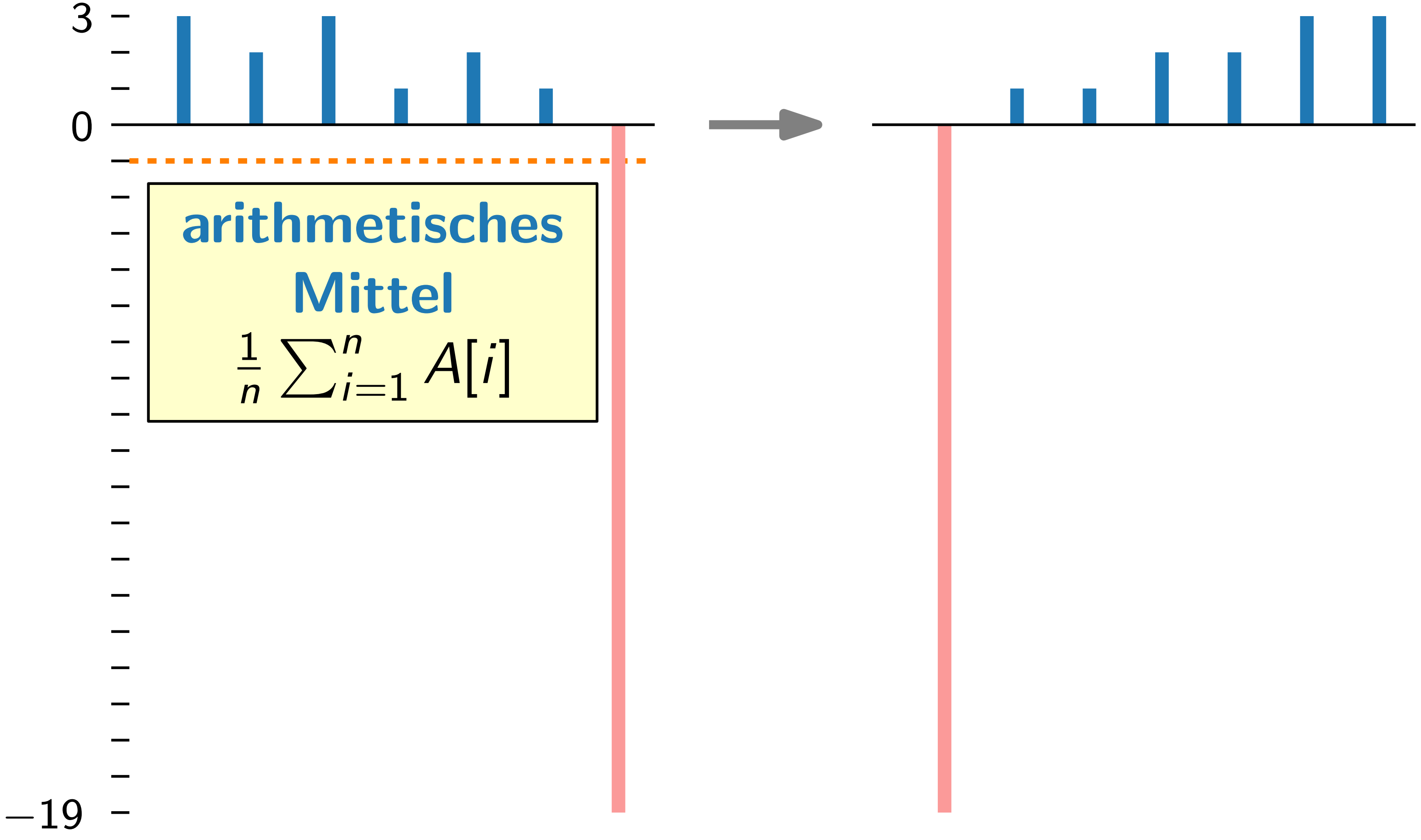
**Beispiel.**



# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \ldots n]$
**Gesucht:** Ein „guter“ Mittelwert

**Beispiel.**

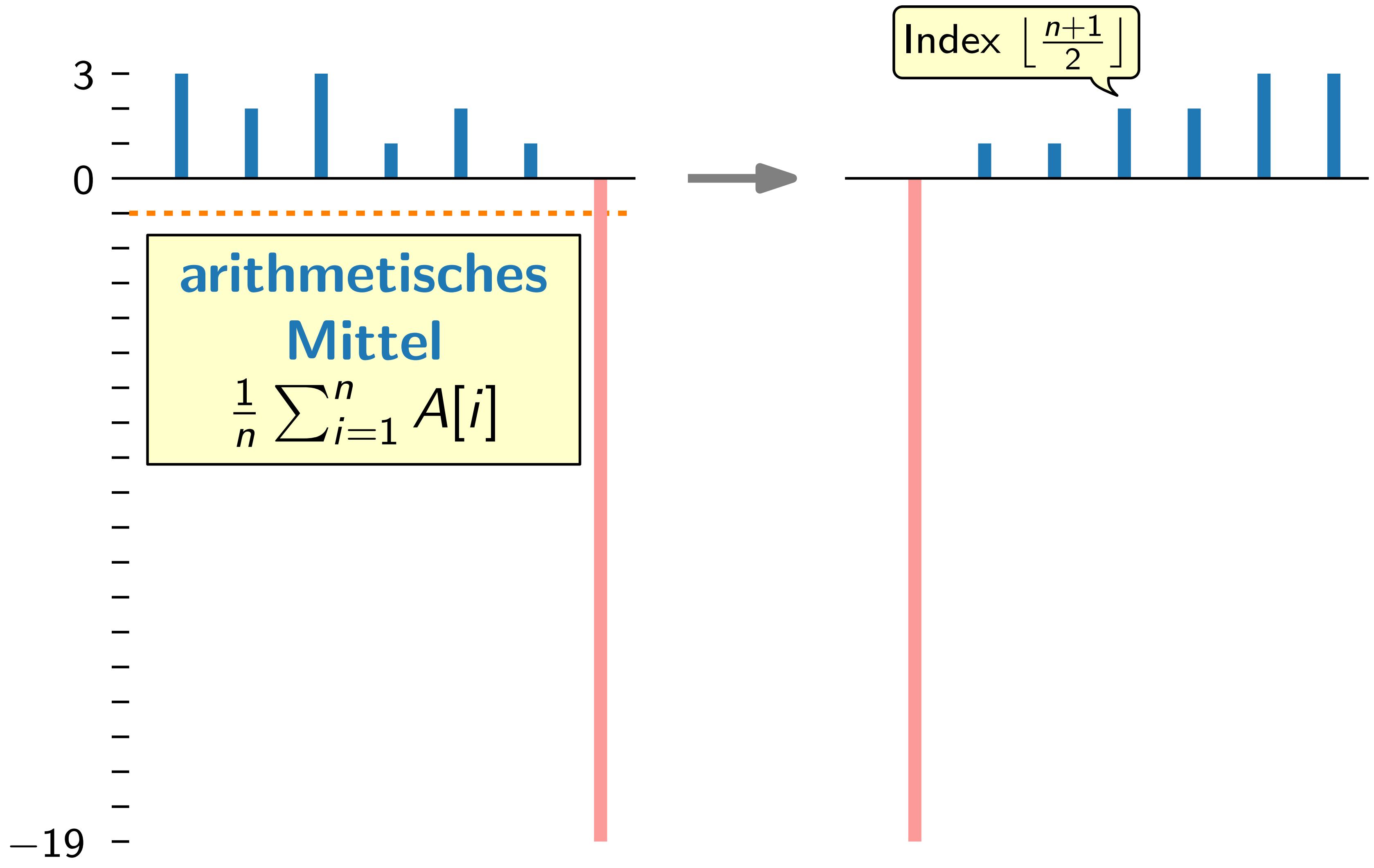


# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$

**Gesucht:** Ein „guter“ Mittelwert

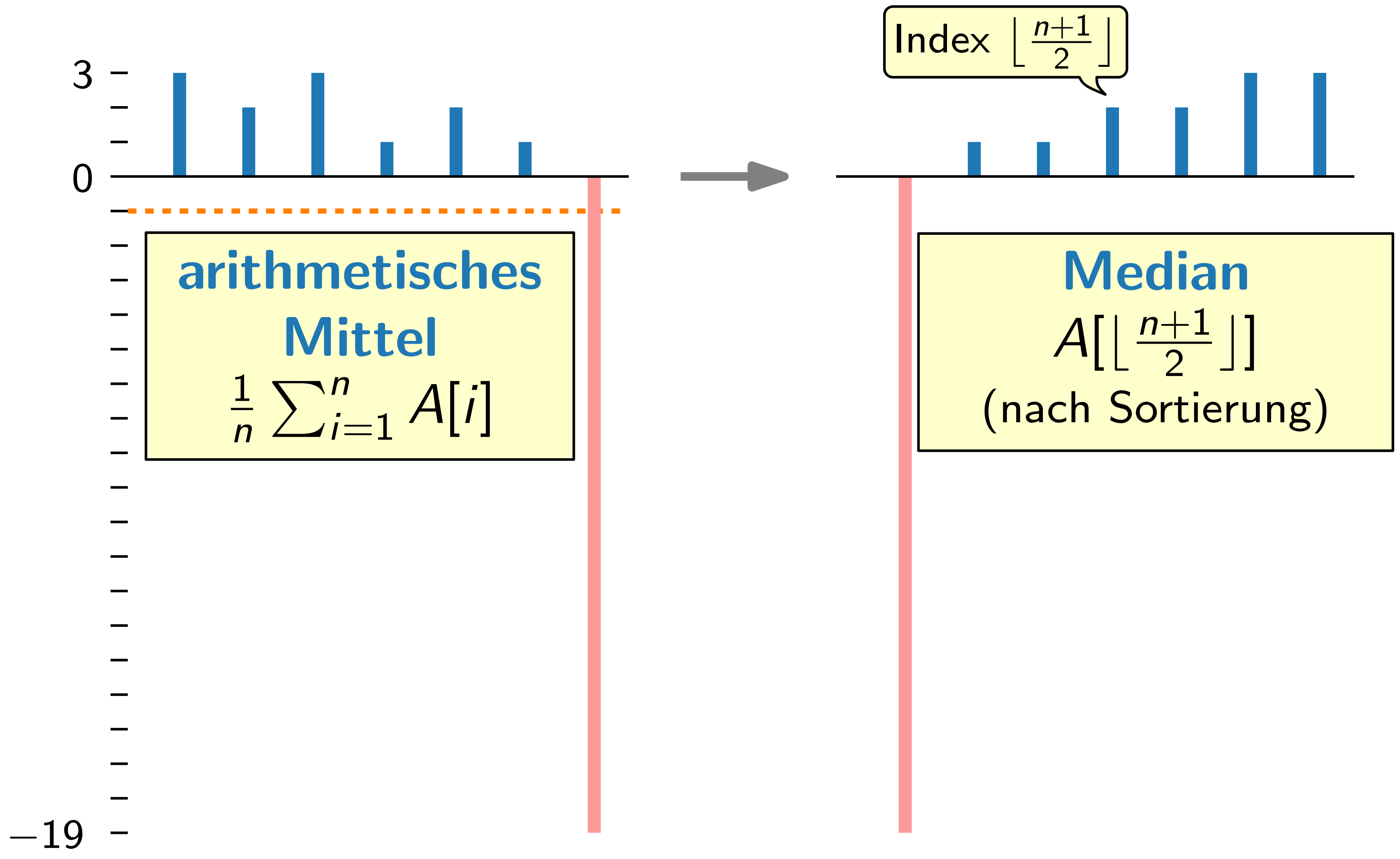
**Beispiel.**



# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Ein „guter“ Mittelwert

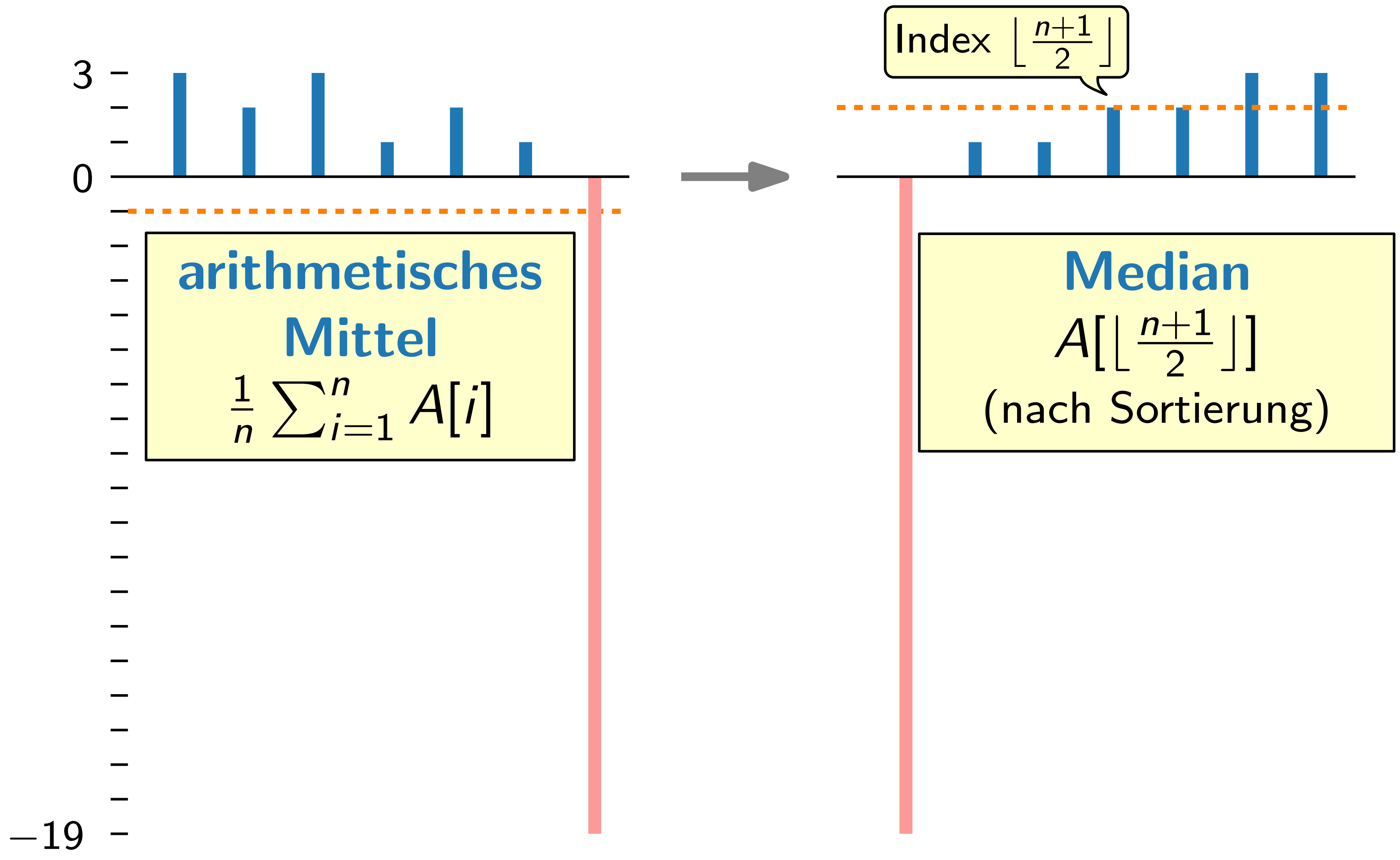
**Beispiel.**



# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Ein „guter“ Mittelwert

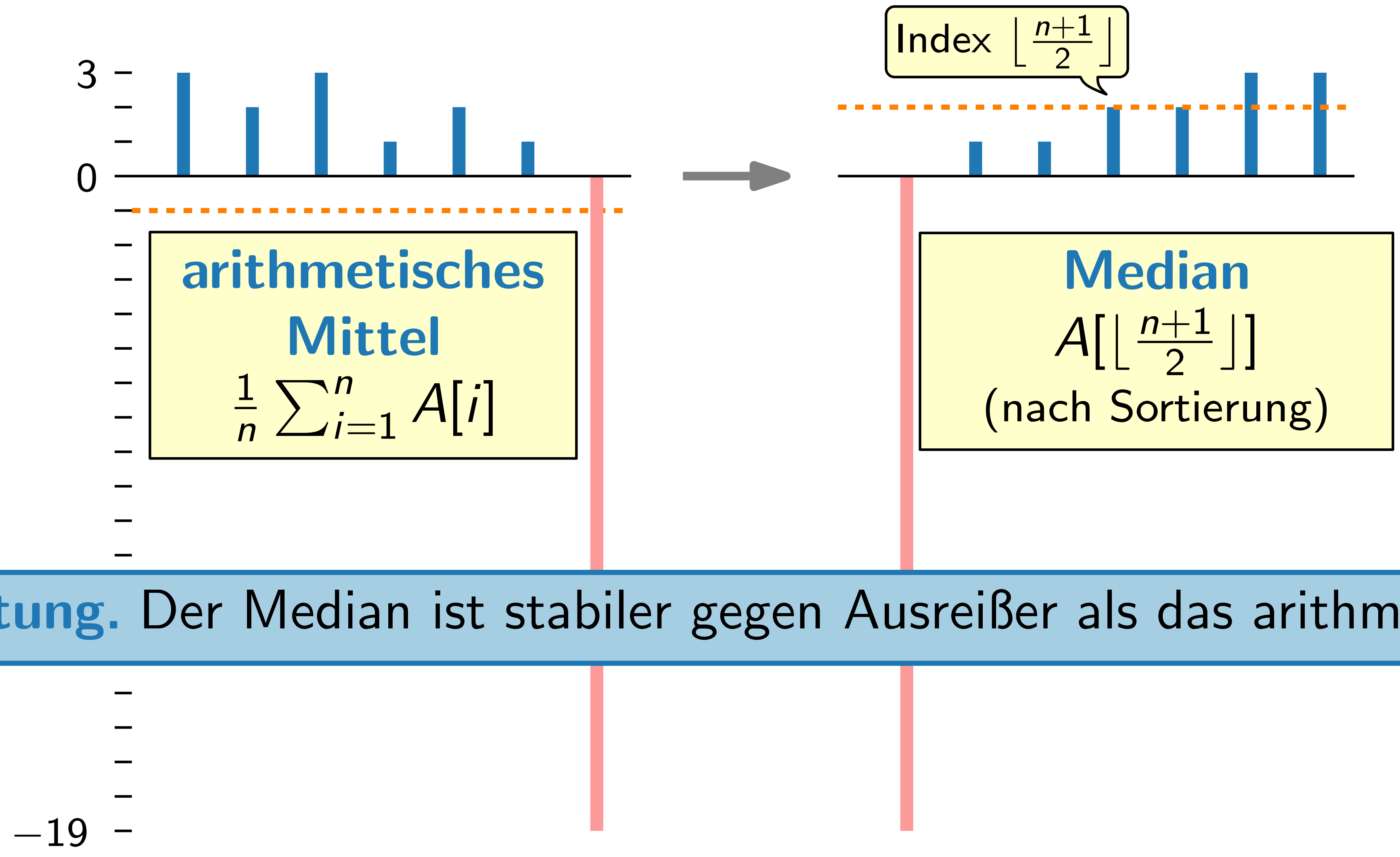
**Beispiel.**



# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Ein „guter“ Mittelwert

**Beispiel.**



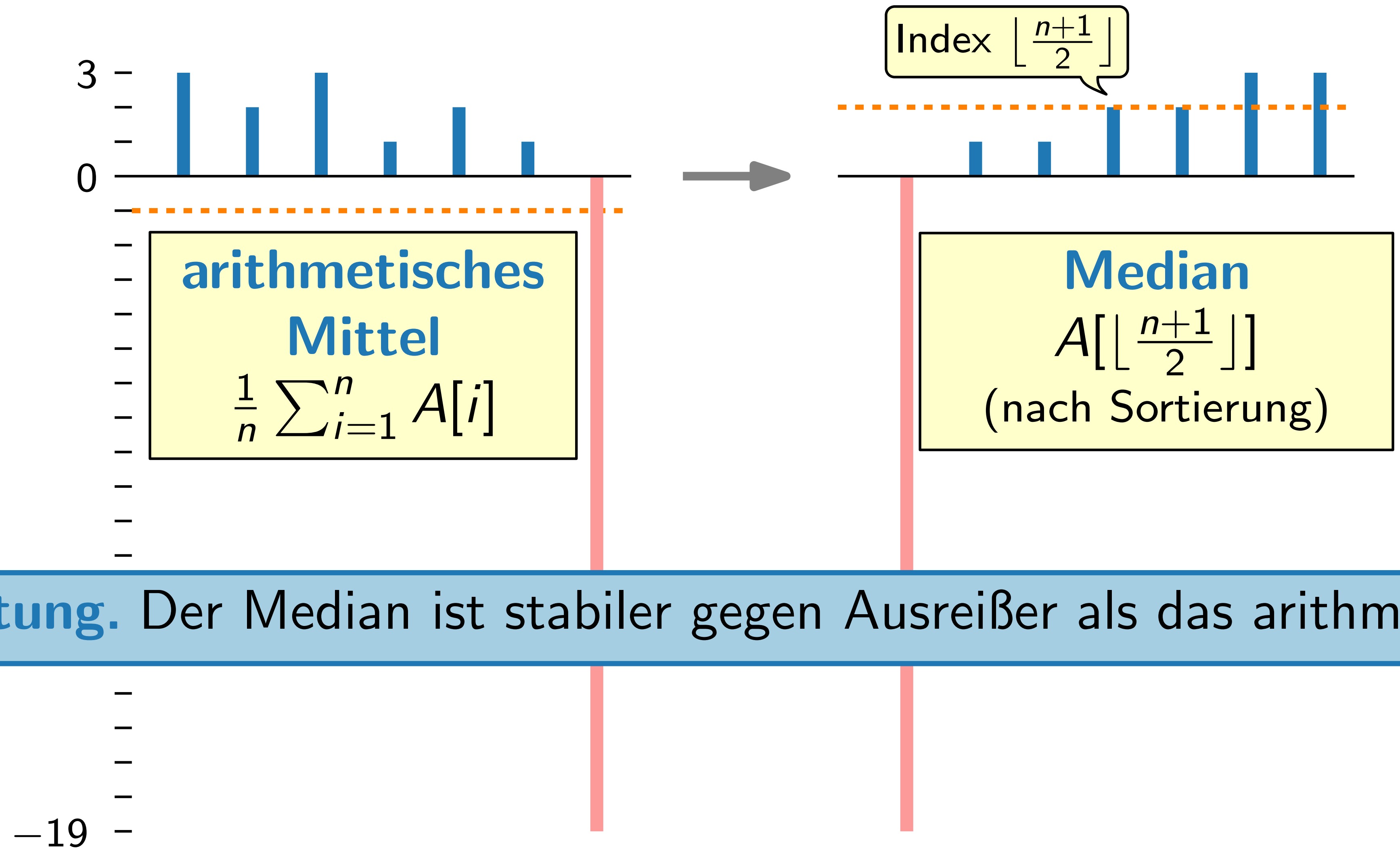

**Beobachtung.** Der Median ist stabiler gegen Ausreißer als das arithmetische Mittel.

# Analyse von Messreihen

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$

**Gesucht:** Ein „guter“ Mittelwert

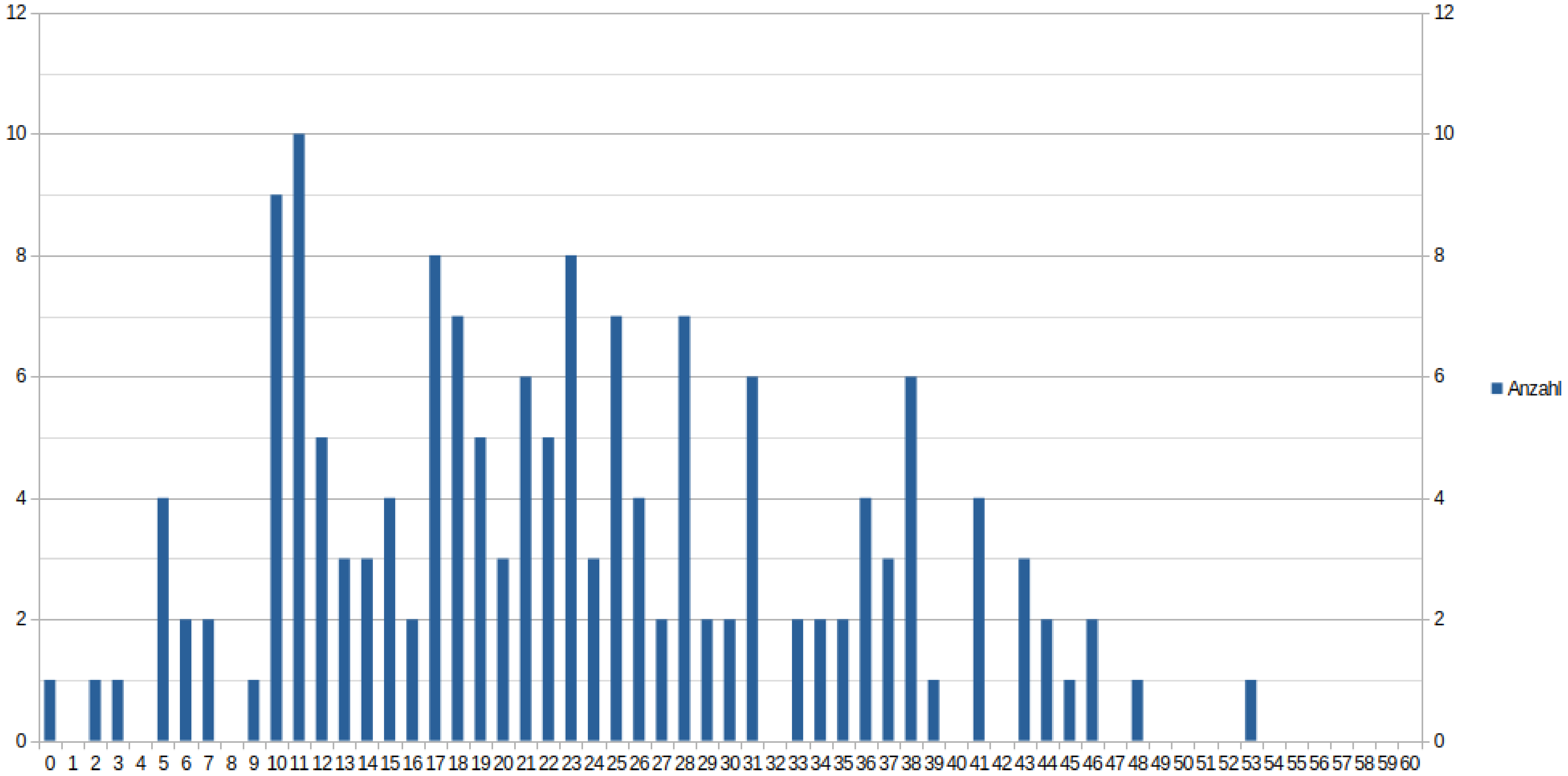
**Beispiel.**




**Berechnung?**

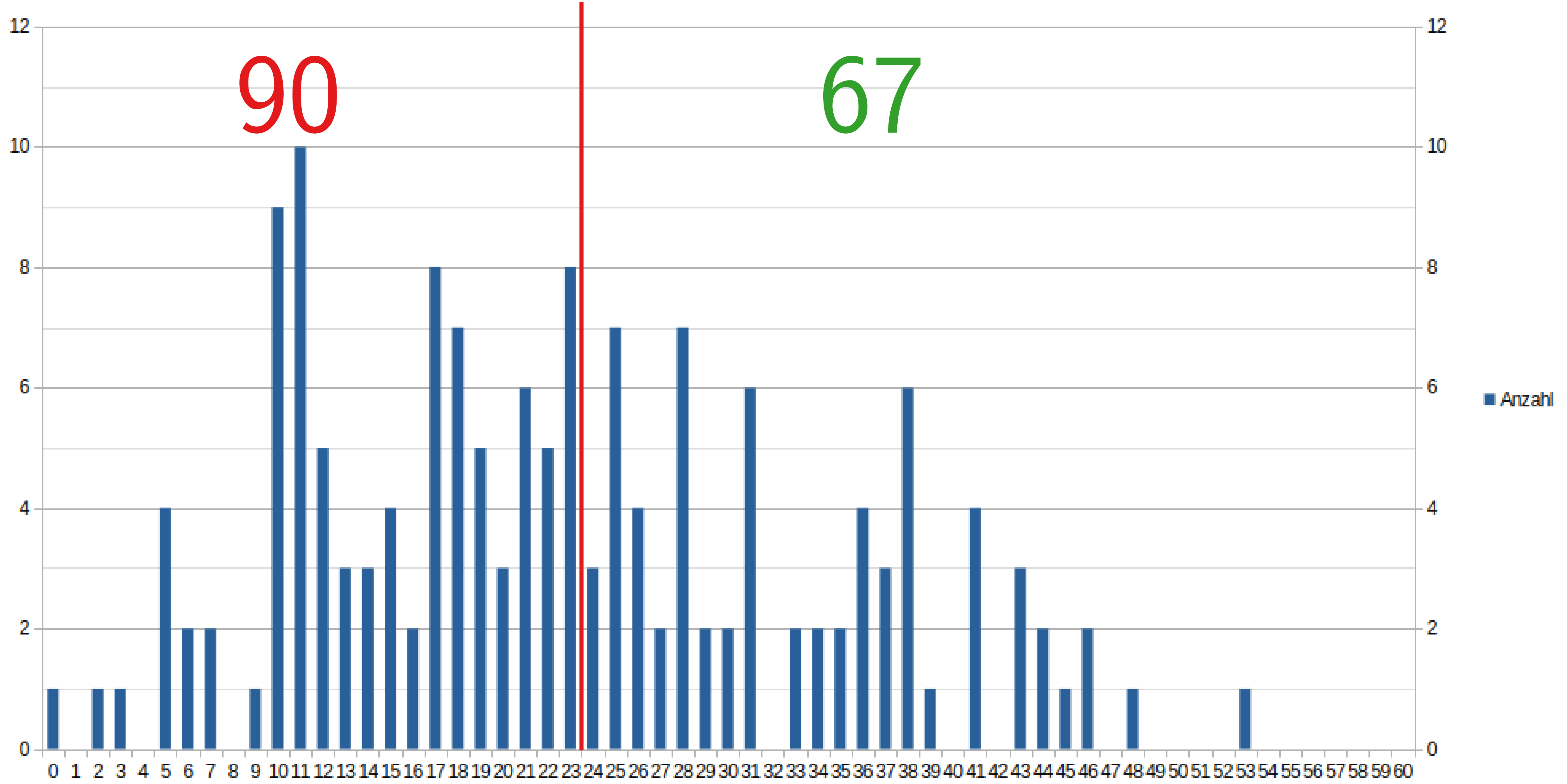
**Beobachtung.** Der Median ist stabiler gegen Ausreißer als das arithmetische Mittel.

# 1. Zwischentest



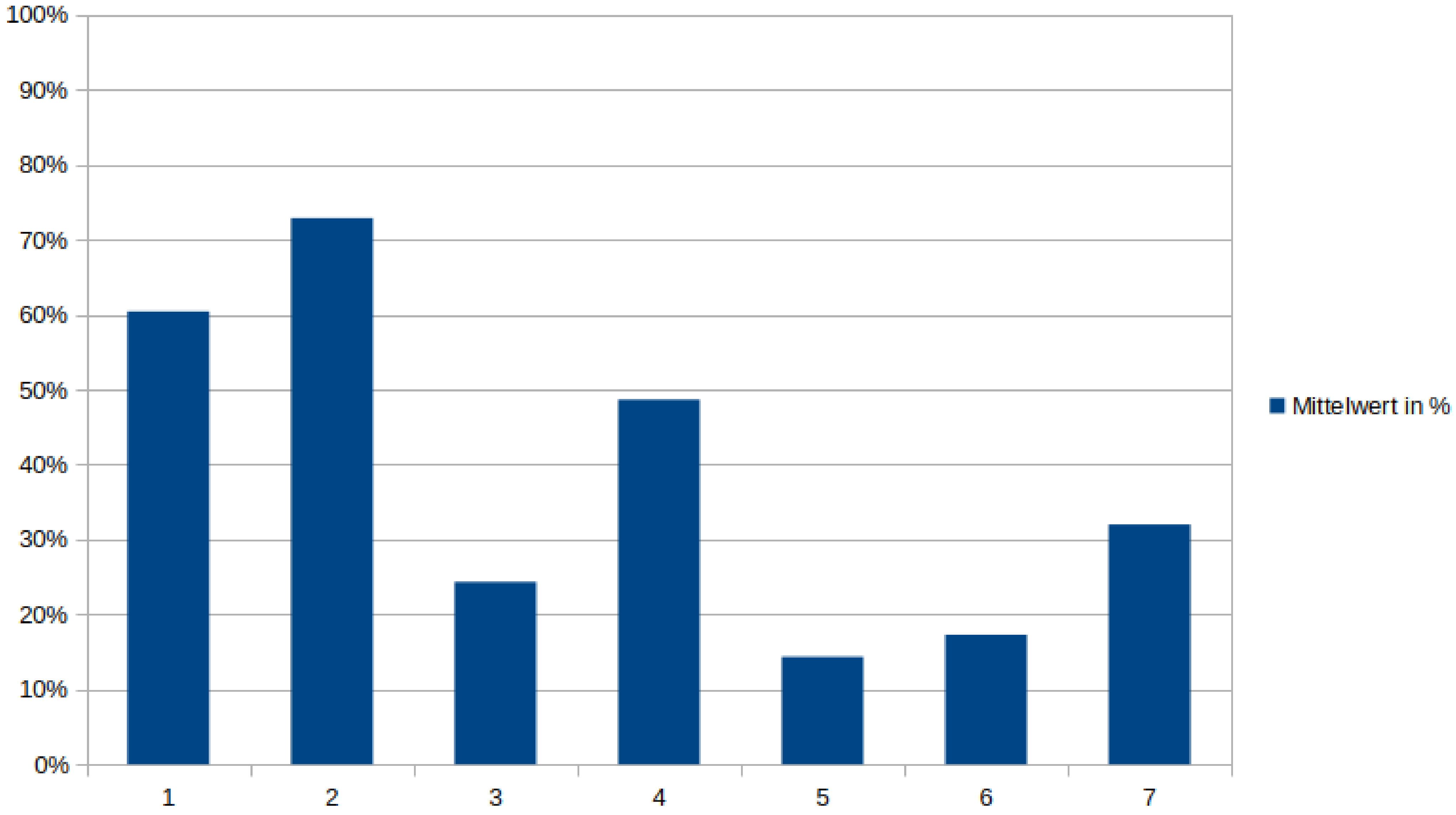

$n = 157$

# 1. Zwischentest

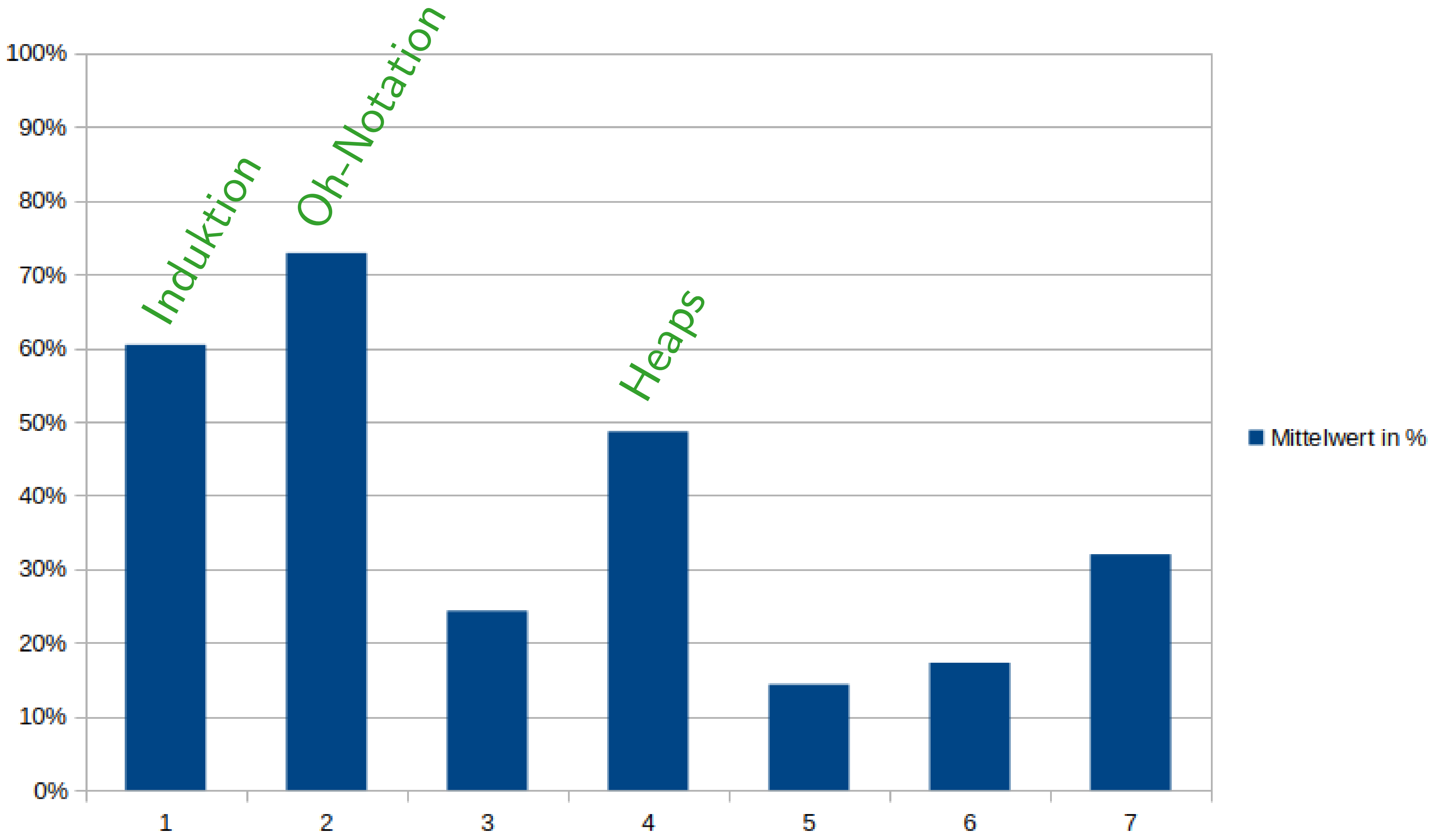



$n = 157$ arithmetisches Mittel $= 22{,}9$ Median $= 19{,}5$

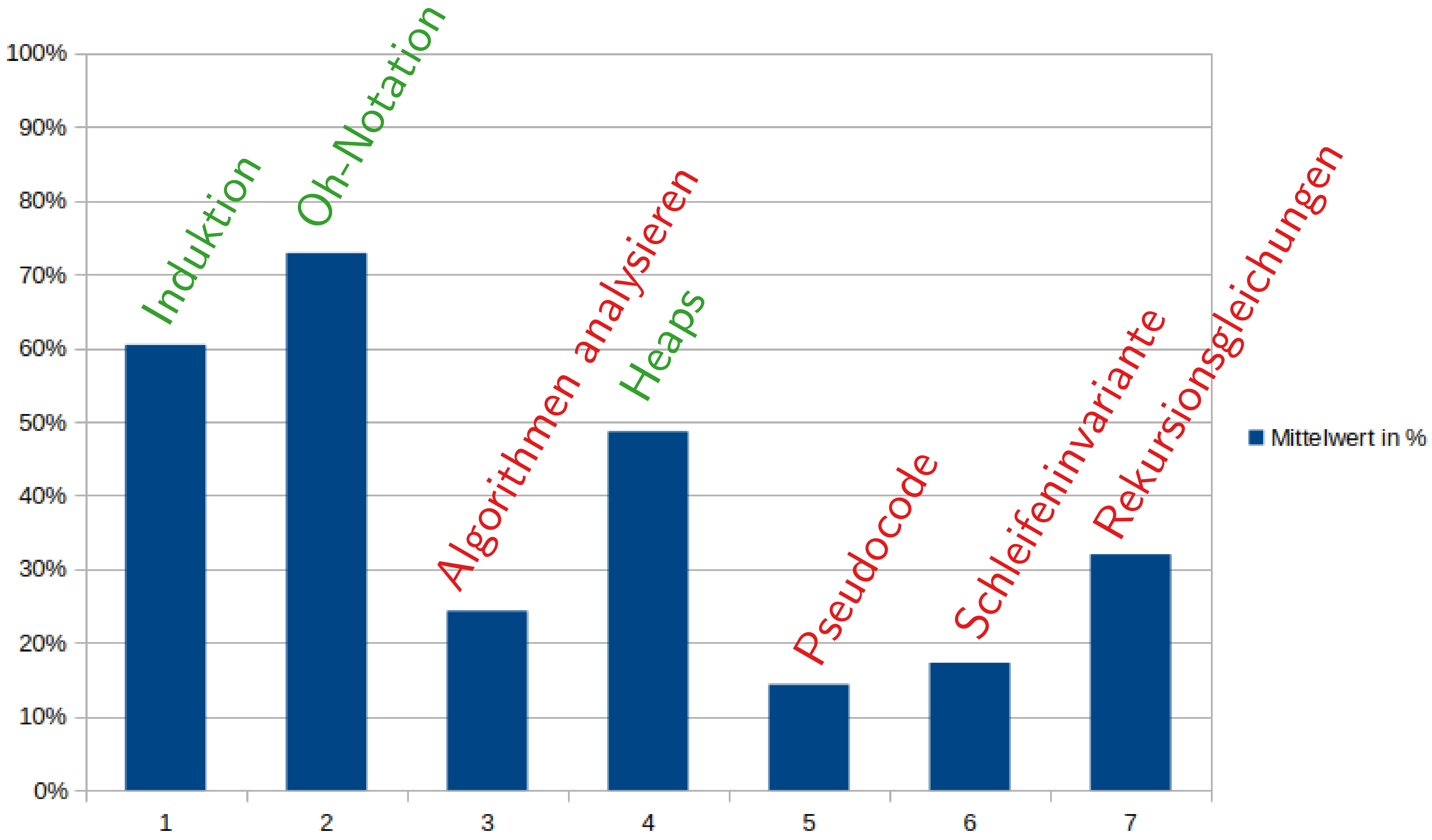
# Ergebnisse nach Aufgabe



# Ergebnisse nach Aufgabe



# Ergebnisse nach Aufgabe



# Das Auswahlproblem

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$

# Das Auswahlproblem

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Das $i$-kleinste Element

# Das Auswahlproblem

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$

**Gesucht:** Das $i$-kleinste Element

**Lösung.**

# Das Auswahlproblem

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Das $i$-kleinste Element

**Lösung.** Sortiere und gib $A[i]$ zurück!

# Das Auswahlproblem

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Das $i$-kleinste Element

**Lösung.** Sortiere und gib $A[i]$ zurück!

Worst-Case-Laufzeit:

# Das Auswahlproblem

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Das $i$-kleinste Element

**Lösung.** Sortiere und gib $A[i]$ zurück!

Worst-Case-Laufzeit: $\Theta(n \log n)$

# Das Auswahlproblem

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$
**Gesucht:** Das $i$-kleinste Element

**Lösung.** Sortiere und gib $A[i]$ zurück!

Worst-Case-Laufzeit: $\Theta(n \log n)$

wenn man nichts über die Verteilung der Zahlen weiß

# Das Auswahlproblem

**Gegeben:** Eine Reihe von $n$ Messwerten $A[1 \dots n]$

**Gesucht:** Das $i$-kleinste Element

**Lösung.** Sortiere und gib $A[i]$ zurück!

Worst-Case-Laufzeit: $\Theta(n \log n)$

wenn man nichts über die Verteilung der Zahlen weiß

Geht das besser?

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$:

$i = 1$:

$i = n$:

# Spezialfälle

| | |
|---|---|
| $i = \lfloor \frac{n+1}{2} \rfloor$: | Median |
| $i = 1$: | Minimum |
| $i = n$: | Maximum |

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

$i = n$: Maximum

$\textsc{Minimum}(\text{int}[]\ A)$

**Aufgabe.**
Schreiben Sie die Funktion $\textsc{Minimum}$ in Pseudocode.

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

$i = n$: Maximum

```
MINIMUM(int[] A)
   min = A[1]



   return min
```

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

$i = n$: Maximum

$\textsc{Minimum}(\text{int}[\,]\ A)$
  $min = A[1]$
  **for** $i = 2$ **to** $A.length$ **do**

  **return** $min$

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

$i = n$: Maximum

```
MINIMUM(int[] A)
  min = A[1]
  for i = 2 to A.length do
  |  if min > A[i] then min = A[i]
  return min
```

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

$i = n$: Maximum

```
Minimum(int[] A)
  min = A[1]
  for i = 2 to A.length do
    if min > A[i] then min = A[i]
  return min
```

Anzahl Vergleiche =

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

$i = n$: Maximum

```
MINIMUM(int[] A)
   min = A[1]
   for i = 2 to A.length do
      if min > A[i] then min = A[i]
   return min
```

Anzahl Vergleiche $= \quad n - 1$

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

$i = n$: Maximum

$\left.\begin{array}{l} \text{Minimum} \\ \text{Maximum} \end{array}\right\}$ Laufzeit $\Theta(n)$

```
Minimum(int[] A)
  min = A[1]
  for i = 2 to A.length do
    if min > A[i] then min = A[i]
  return min
```

Anzahl Vergleiche $= \quad n - 1$

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

$i = n$: Maximum

$\left.\begin{array}{l} \text{Minimum} \\ \text{Maximum} \end{array}\right\}$ Laufzeit $\Theta(n)$

```
Minimum(int[] A)
  min = A[1]
  for i = 2 to A.length do
    if min > A[i] then min = A[i]
  return min
```

Anzahl Vergleiche $= n - 1$

Ist das optimal?

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

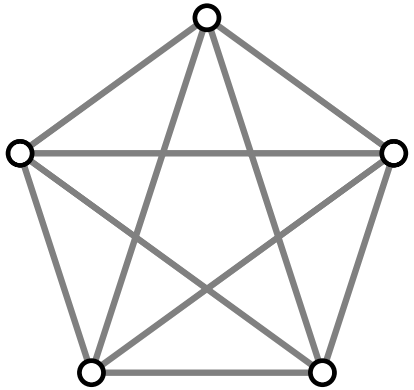
$i = n$: Maximum

$\left.\begin{array}{l} \text{Minimum} \\ \text{Maximum} \end{array}\right\}$ Laufzeit $\Theta(n)$

MINIMUM(int[] $A$)
  $min = A[1]$
  **for** $i = 2$ **to** $A.length$ **do**
    **if** $min > A[i]$ **then** $min = A[i]$
  **return** $min$

Anzahl Vergleiche $=$ $n - 1$

Ist das optimal? Betrachte ein K.O.-Turnier.



# Spezialfälle

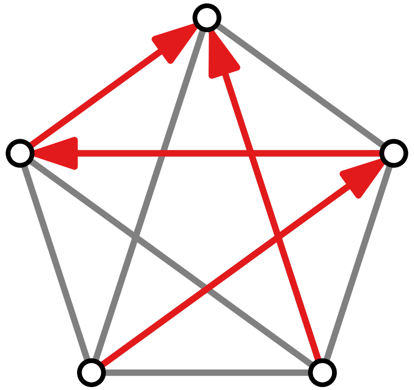
$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$\left.\begin{array}{ll} i = 1\text{:} & \text{Minimum} \\ i = n\text{:} & \text{Maximum} \end{array}\right\}$ Laufzeit $\Theta(n)$

```
MINIMUM(int[] A)
  min = A[1]
  for i = 2 to A.length do
    if min > A[i] then min = A[i]
  return min
```

Anzahl Vergleiche $= n - 1$

Ist das optimal?



Betrachte ein K.O.-Turnier.

Bis ein Gewinner feststeht, muss *jeder* – außer dem Gewinner – mindestens einmal verlieren.

# Spezialfälle

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

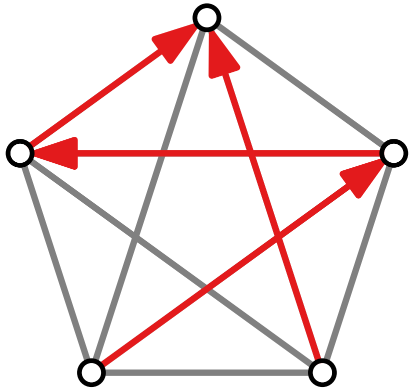
$i = n$: Maximum

$\Big\}$ Laufzeit $\Theta(n)$

```
MINIMUM(int[] A)
  min = A[1]
  for i = 2 to A.length do
    if min > A[i] then min = A[i]
  return min
```

Anzahl Vergleiche $= n - 1$

Ist das optimal?



Betrachte ein K.O.-Turnier.

Bis ein Gewinner feststeht, muss *jeder* – außer dem Gewinner – mindestens einmal verlieren.

Also sind $n - 1$ Vergleiche optimal.

# Spezialfälle

Geht das auch in linearer Zeit?

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

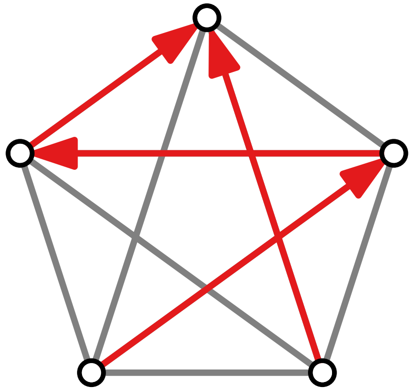
$i = n$: Maximum

$\left.\begin{array}{l} \text{Minimum} \\ \text{Maximum} \end{array}\right\}$ Laufzeit $\Theta(n)$

**Minimum**(int[] $A$)
  $min = A[1]$
  **for** $i = 2$ **to** $A.length$ **do**
    **if** $min > A[i]$ **then** $min = A[i]$
  **return** $min$

Anzahl Vergleiche $= n - 1$

Ist das optimal?



Betrachte ein K.O.-Turnier.

Bis ein Gewinner feststeht, muss *jeder* – außer dem Gewinner – mindestens einmal verlieren.

Also sind $n - 1$ Vergleiche optimal.

# Spezialfälle

Geht das auch in linearer Zeit?

$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: Minimum

$i = n$: Maximum

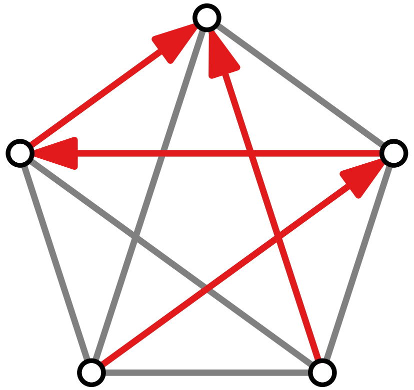
$\left.\begin{array}{l} \\ \end{array}\right\}$ Laufzeit $\Theta(n)$

Geht beides zusammen mit weniger als $2(n-1)$ Vergleichen?

```
MINIMUM(int[] A)
  min = A[1]
  for i = 2 to A.length do
    if min > A[i] then min = A[i]
  return min
```

Anzahl Vergleiche $= n - 1$

Ist das optimal?



Betrachte ein K.O.-Turnier.

Bis ein Gewinner feststeht, muss *jeder* – außer dem Gewinner – mindestens einmal verlieren.

Also sind $n - 1$ Vergleiche optimal.

# Zuerst zur zweiten Frage

Sei $V_{\mathrm{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq$

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \ 2 \cdot V_{\text{min}}(n) =$

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq 2 \cdot V_{\text{min}}(n) = 2 \cdot (n-1)$

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq 2 \cdot V_{\text{min}}(n) = 2 \cdot (n-1)$

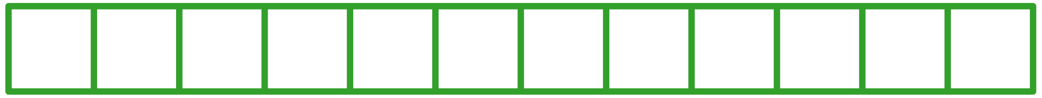
**Frage.** Geht es auch mit weniger Vergleichen?

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n - 1)$

**Frage.** Geht es auch mit weniger Vergleichen?



min

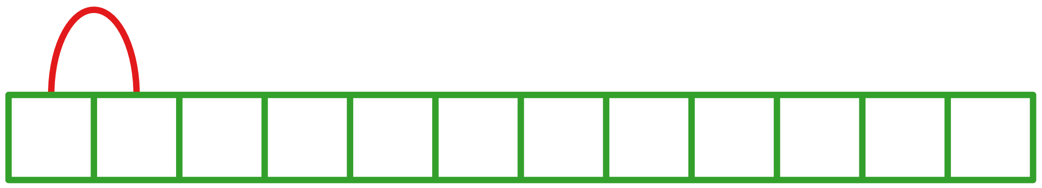
max

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \ 2 \cdot V_{\text{min}}(n) = \ 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?



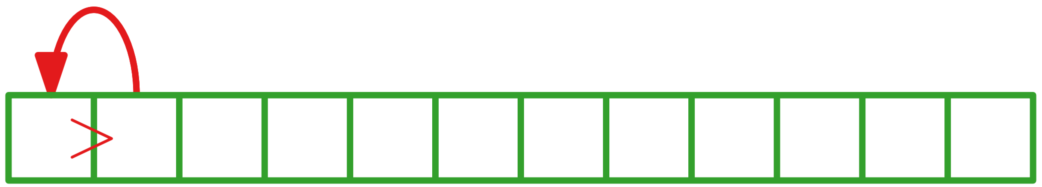
min
max

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?




min

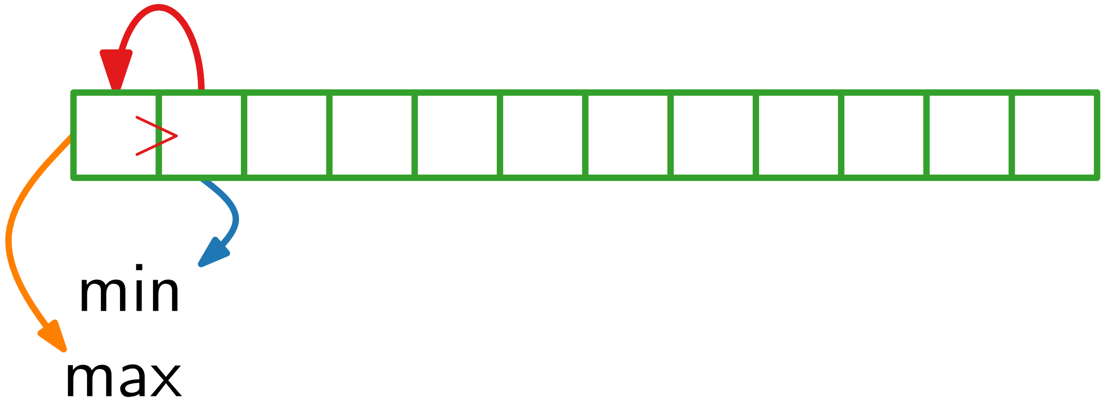
max

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?

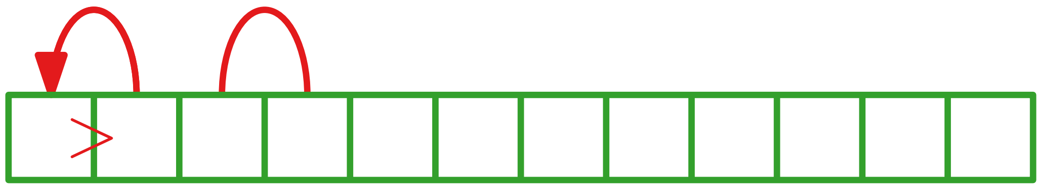


# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?



min

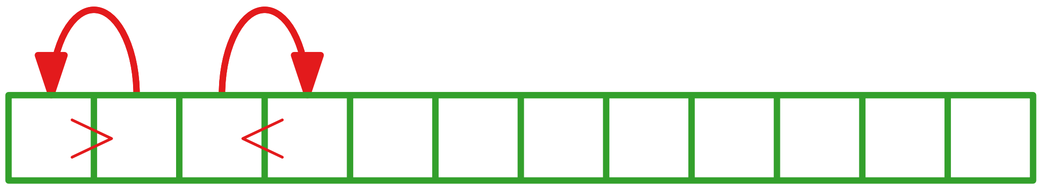
max

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?




min

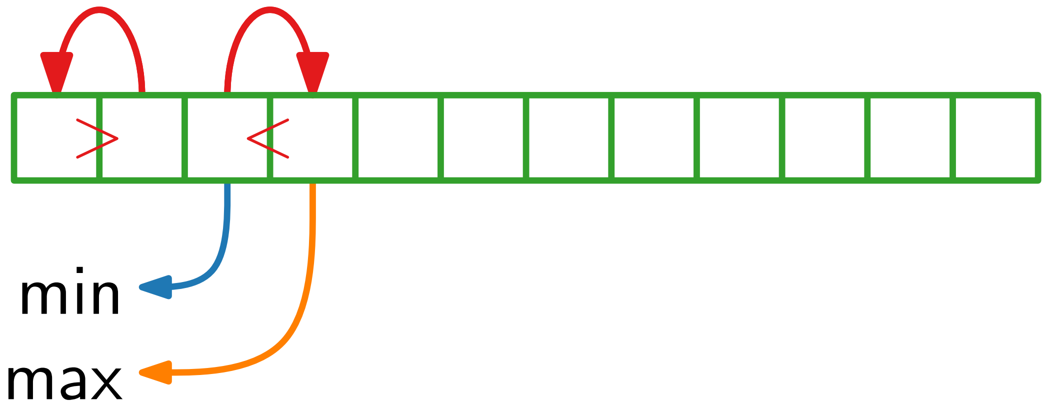
max

# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?

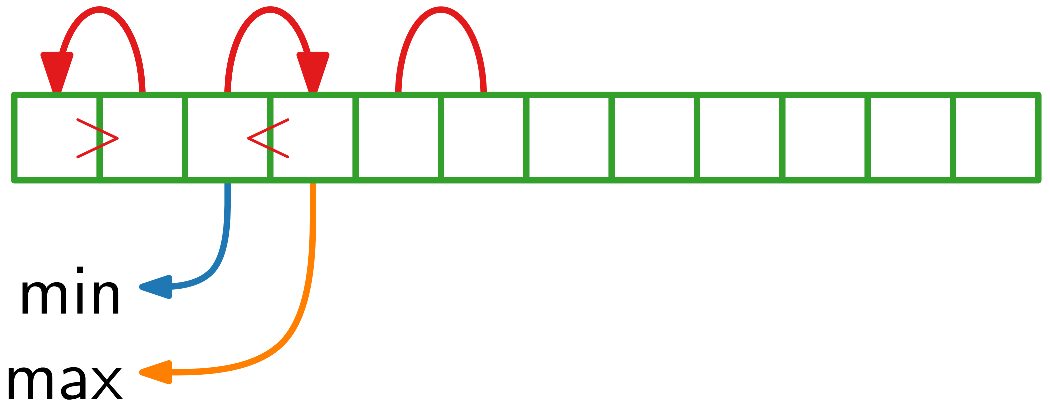


# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq 2 \cdot V_{\text{min}}(n) = 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?

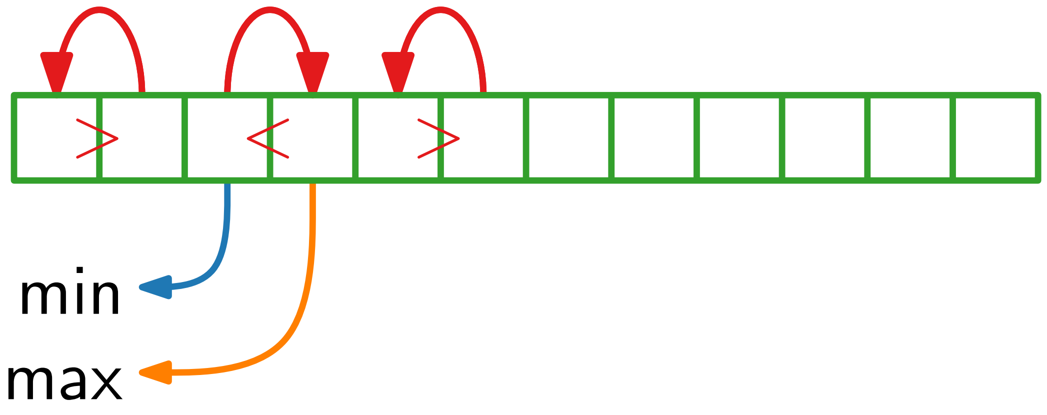


# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?

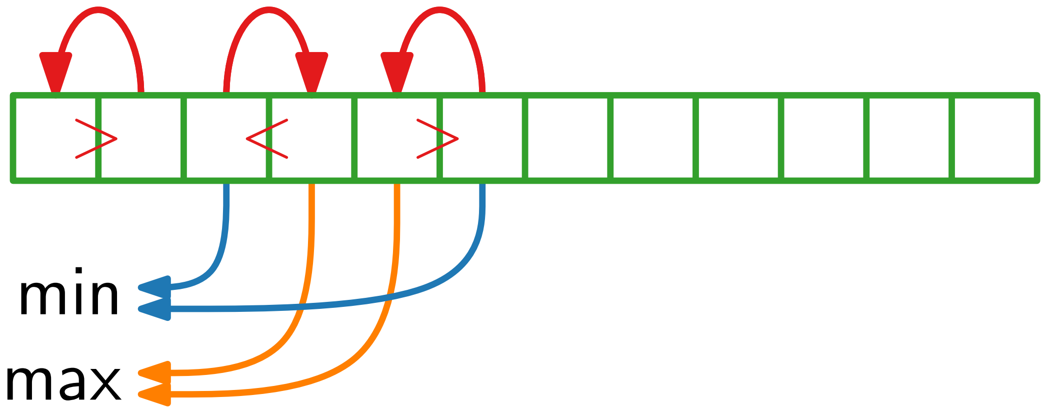


# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq\ 2 \cdot V_{\text{min}}(n) =\ 2 \cdot (n - 1)$

**Frage.** Geht es auch mit weniger Vergleichen?

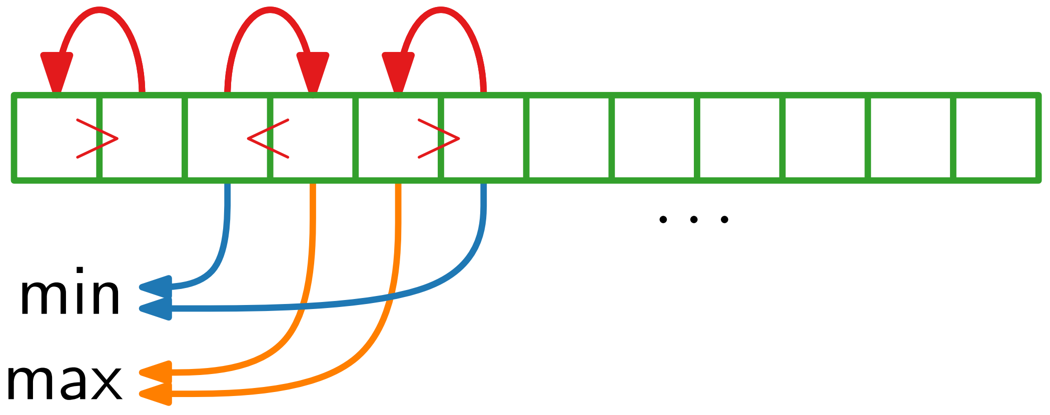


# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?

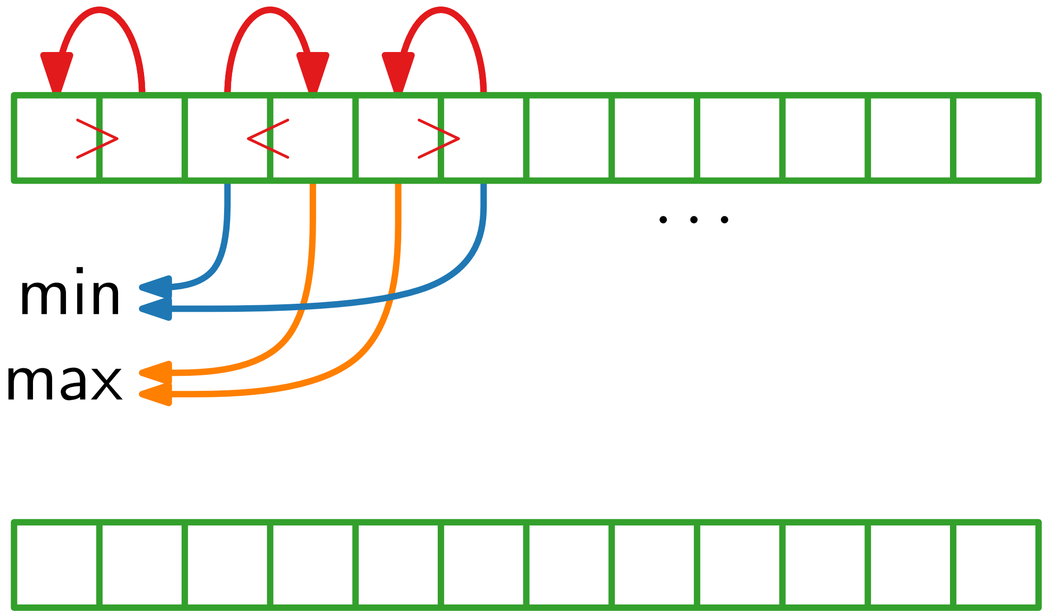


# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?



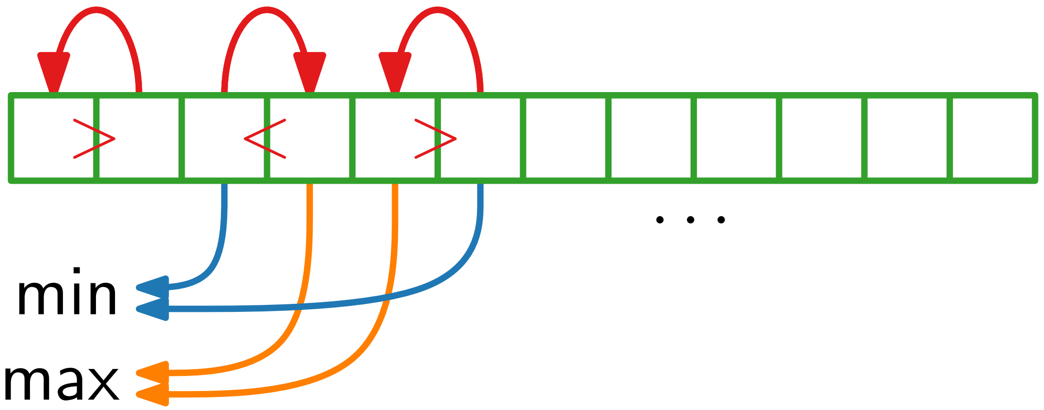

Anzahl der Vergleiche

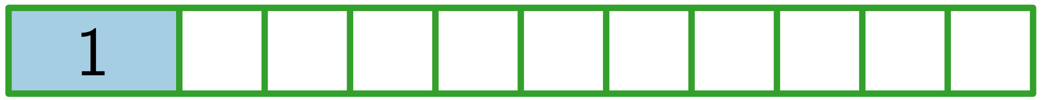
# Zuerst zur zweiten Frage

Sei $V_{\mathrm{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\mathrm{minmax}}(n) \leq 2 \cdot V_{\mathrm{min}}(n) = 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?



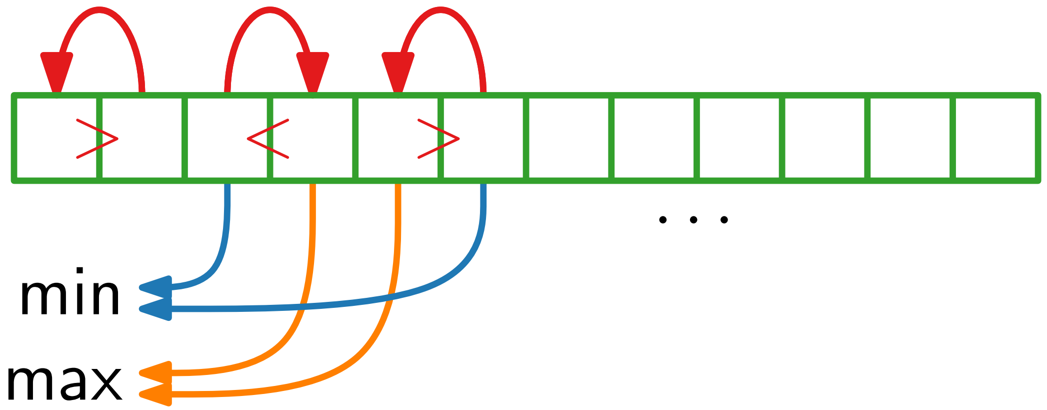

Anzahl der Vergleiche

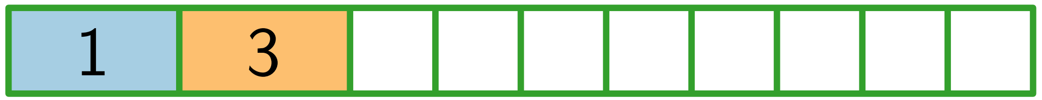
# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq\ 2 \cdot V_{\min}(n) =\ 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?



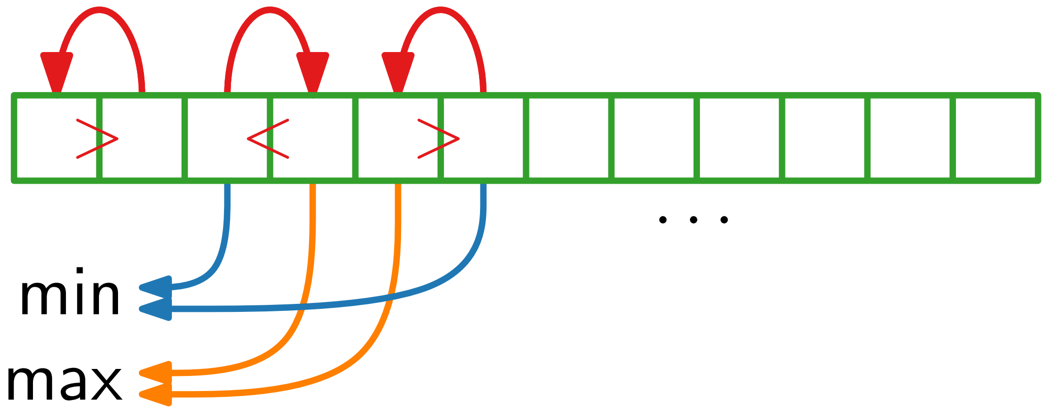

Anzahl der Vergleiche

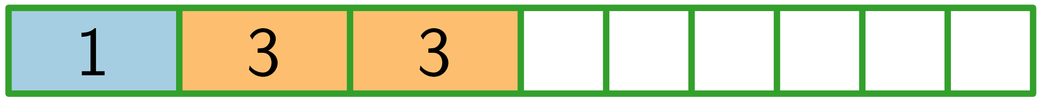
# Zuerst zur zweiten Frage

Sei $V_{\mathrm{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\mathrm{minmax}}(n) \leq \; 2 \cdot V_{\mathrm{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?



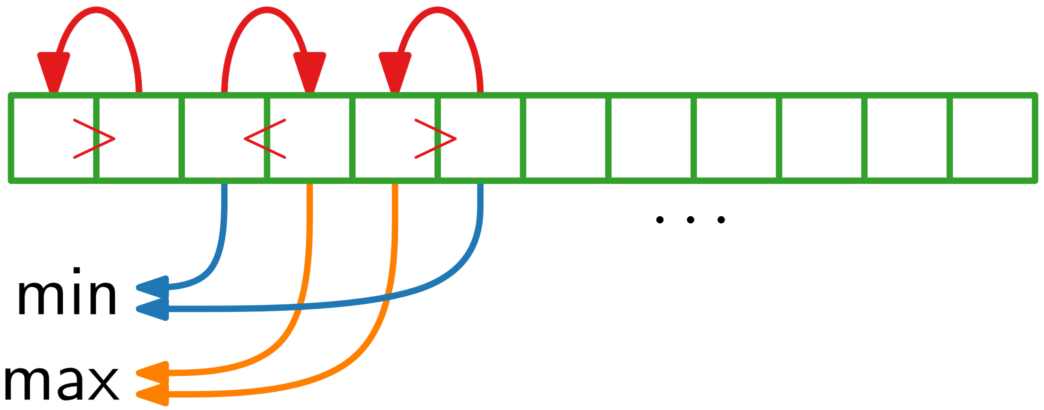

Anzahl der Vergleiche

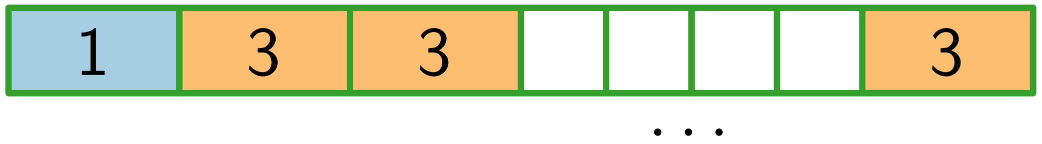
# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?



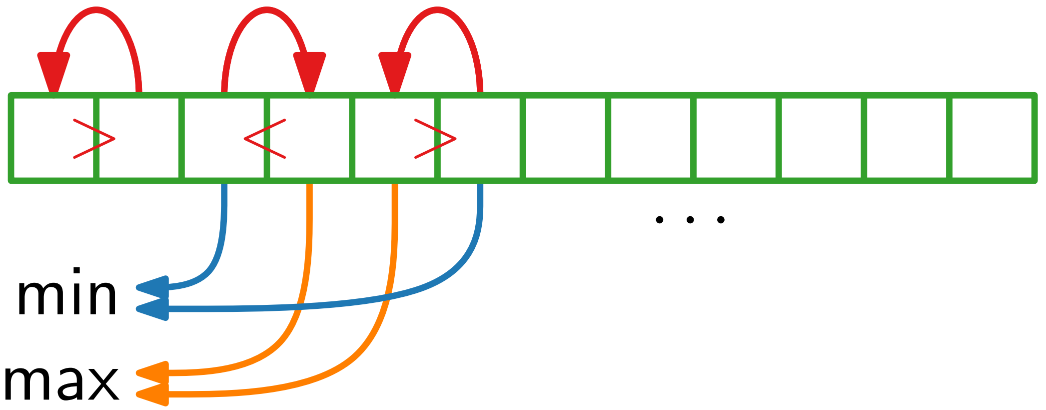

Anzahl der Vergleiche

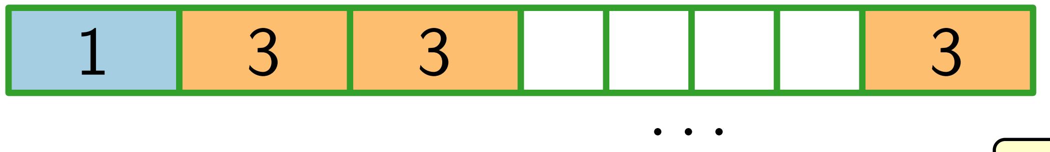
# Zuerst zur zweiten Frage

Sei $V_{\mathrm{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\mathrm{minmax}}(n) \leq 2 \cdot V_{\mathrm{min}}(n) = 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?




Anzahl der Vergleiche

$=$

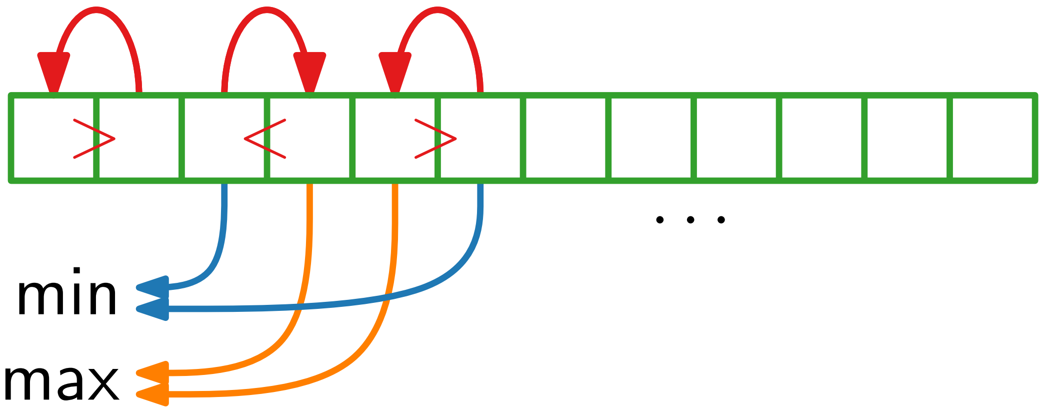
$n$ gerade

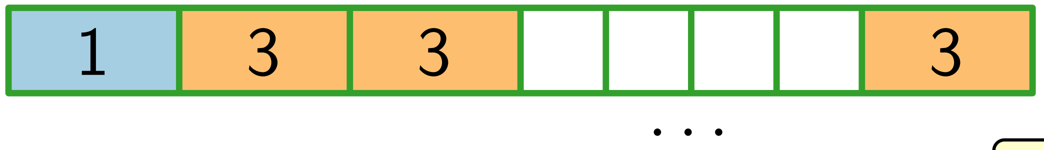
# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq 2 \cdot V_{\text{min}}(n) = 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?




Anzahl der Vergleiche

$= 1 \cdot 1 + (n/2 - 1) \cdot 3$

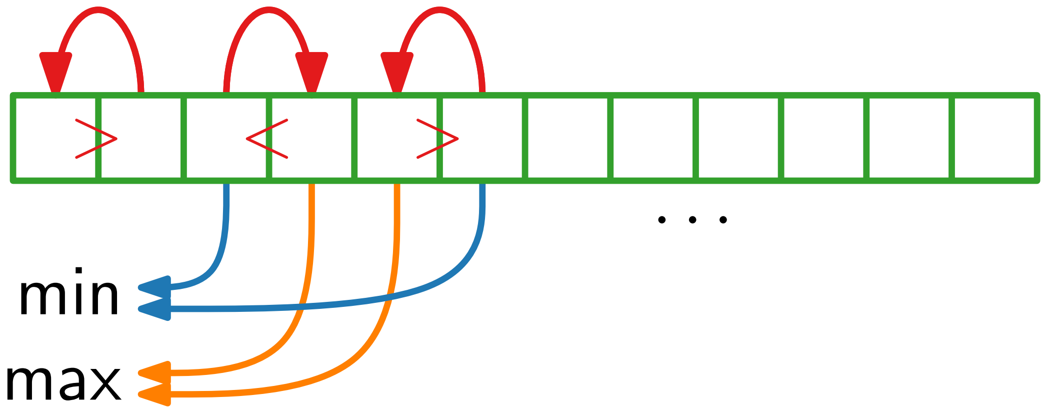
$n$ gerade

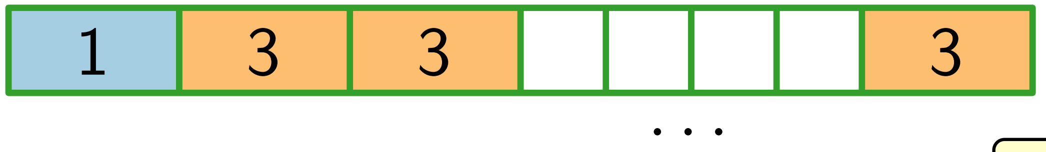
# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \ 2 \cdot V_{\text{min}}(n) = \ 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?




Anzahl der Vergleiche

$= 1 \cdot 1 + (n/2 - 1) \cdot 3 \ = 3n/2 - 2$

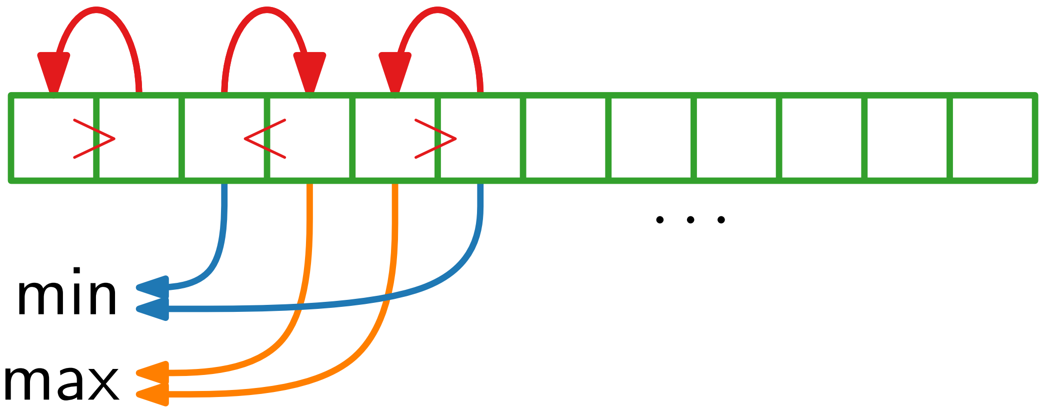
*n* gerade

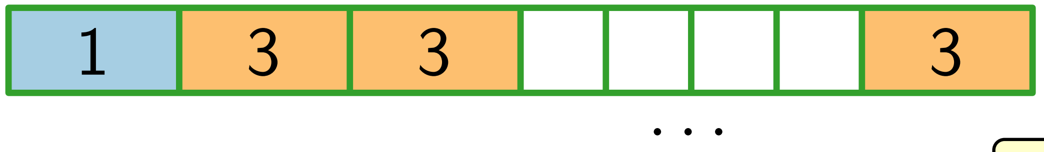
# Zuerst zur zweiten Frage

Sei $V_{\text{minmax}}(n)$ die Anzahl der Vergleiche, die man braucht, um Minimum **und** Maximum von $n$ Zahlen zu bestimmen.

**Klar.** $V_{\text{minmax}}(n) \leq \; 2 \cdot V_{\text{min}}(n) = \; 2 \cdot (n-1)$

**Frage.** Geht es auch mit weniger Vergleichen?




Anzahl der Vergleiche

$= 1 \cdot 1 + (n/2 - 1) \cdot 3 \; = 3n/2 - 2$

*n* gerade

Ist das optimal?

# Auswahl per Teile & Herrsche

Zur Erinnerung...

# Auswahl per Teile & Herrsche

Zur Erinnerung...

$\textsc{QuickSort}(A, \ell = 1, r = A.length)$

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

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

  $\textsc{QuickSort}(A, \ell, m - 1)$

  $\textsc{QuickSort}(A, m + 1, r)$

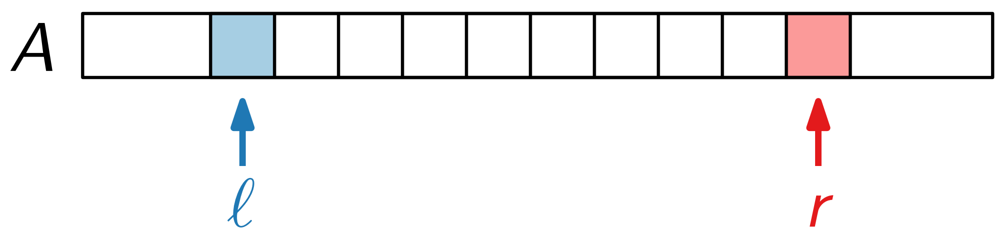
# Auswahl per Teile & Herrsche

Zur Erinnerung...

$\textsc{QuickSort}(A, \ell = 1, r = A.length)$

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

  $m = \textsc{Partition}(A, \ell, r)$
  $\textsc{QuickSort}(A, \ell, m-1)$
  $\textsc{QuickSort}(A, m+1, r)$



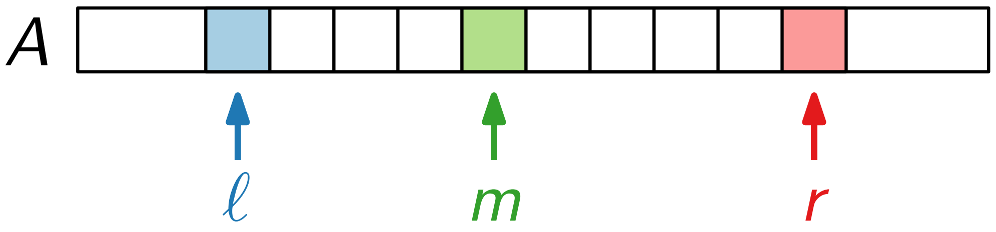
# Auswahl per Teile & Herrsche

Zur Erinnerung...

$\textsc{QuickSort}(A, \ell = 1, r = A.length)$

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

$m = \textsc{Partition}(A, \ell, r)$
$\textsc{QuickSort}(A, \ell, m - 1)$
$\textsc{QuickSort}(A, m + 1, r)$



# Auswahl per Teile & Herrsche

Zur Erinnerung...

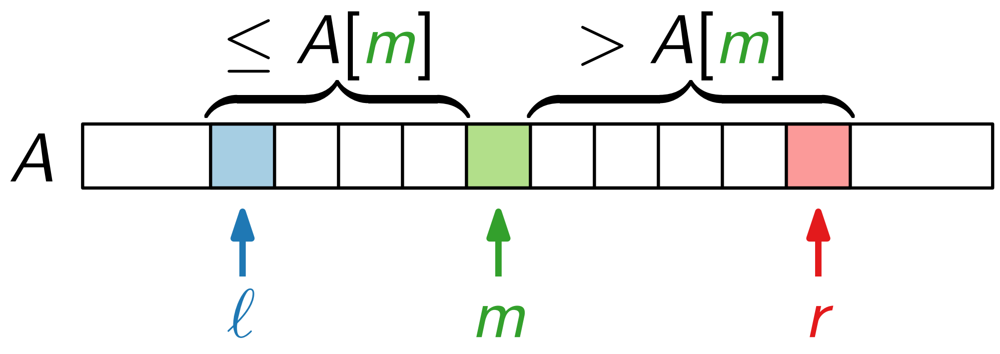
$\textsc{QuickSort}(A, \ell = 1, r = A.length)$

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

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

  $\textsc{QuickSort}(A, \ell, m - 1)$

  $\textsc{QuickSort}(A, m + 1, r)$



# Auswahl per Teile & Herrsche

Zur Erinnerung...

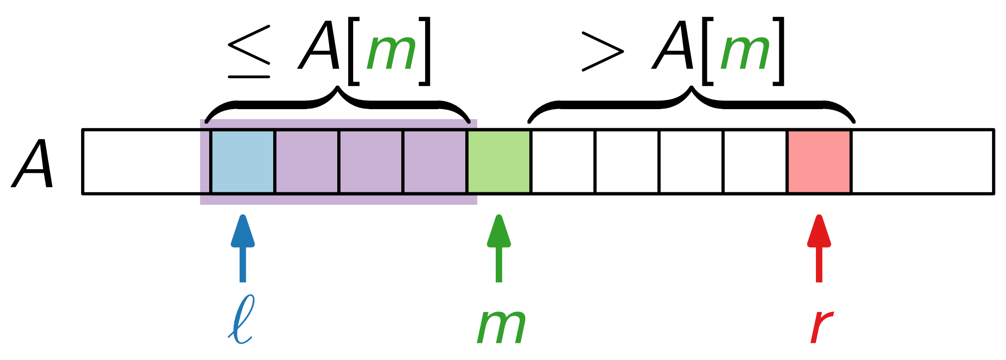
$\textsc{QuickSort}(A, \ell = 1, r = A.length)$

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

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

$\quad \textsc{QuickSort}(A, \ell, m - 1)$

$\quad \textsc{QuickSort}(A, m + 1, r)$



# Auswahl per Teile & Herrsche

Zur Erinnerung...

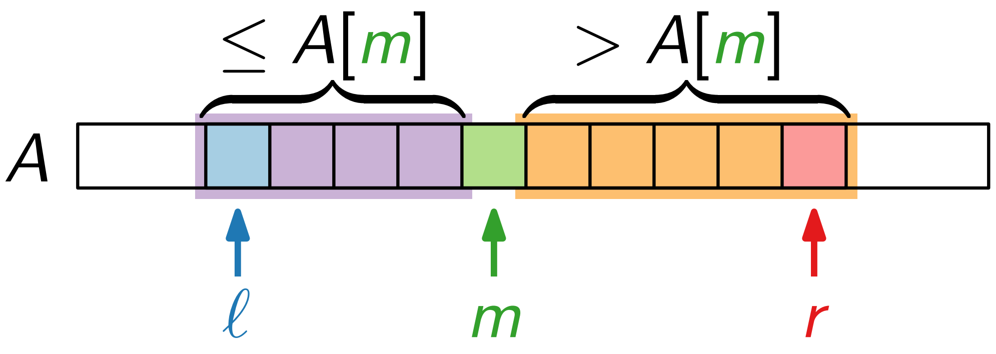
$\textsc{QuickSort}(A, \ell = 1, r = A.length)$

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

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

$\quad \textsc{QuickSort}(A, \ell, m - 1)$

$\quad \textsc{QuickSort}(A, m + 1, r)$



# Auswahl per Teile & Herrsche

Zur Erinnerung...

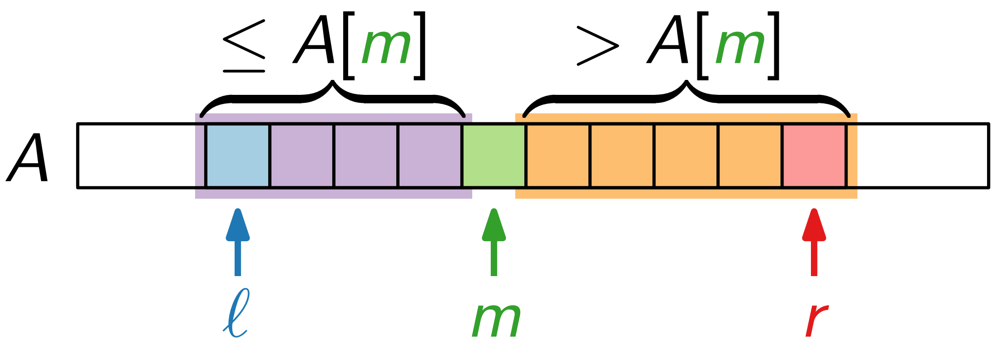
RANDOMIZED
QUICKSORT($A, \ell = 1, r = A.length$)

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

  $m =$ RANDOMIZED PARTITION($A, \ell, r$)

  QUICKSORT($A, \ell, m - 1$)

  QUICKSORT($A, m + 1, r$)



# Auswahl per Teile & Herrsche

Zur Erinnerung...

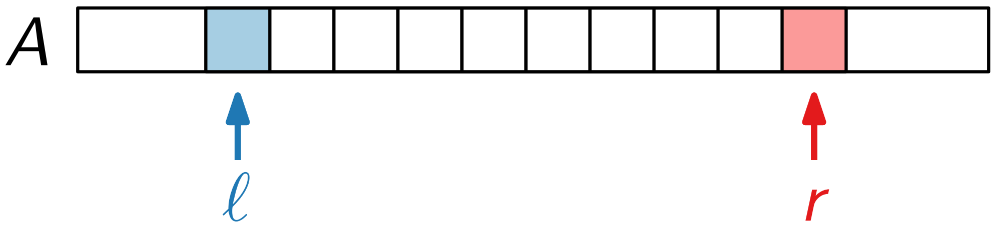
Randomized
$\textsc{QuickSort}(A, \ell = 1, r = A.length)$

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

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

$\textsc{QuickSort}(A, \ell, m - 1)$

$\textsc{QuickSort}(A, m + 1, r)$



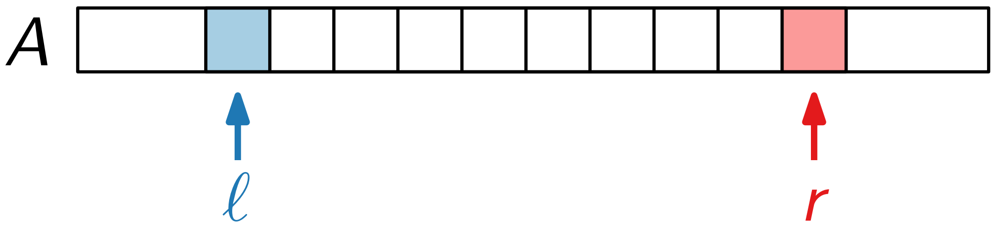

Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

# Auswahl per Teile & Herrsche

Zur Erinnerung...

RANDOMIZED
QUICKSORT($A$, $\ell = 1$, $r = A.length$)

**if** $\ell < r$ **then**
- RANDOMIZED
  $m$ = PARTITION($A$, $\ell$, $r$)
- QUICKSORT($A$, $\ell$, $m - 1$)
- QUICKSORT($A$, $m + 1$, $r$)




Finde $i$.-kleinstes Element in $A[\ell \ldots r]$!

RANDOMIZEDSELECT(int[] $A$, int $\ell$, int $r$, int $i$)

# Auswahl per Teile & Herrsche

Zur Erinnerung...

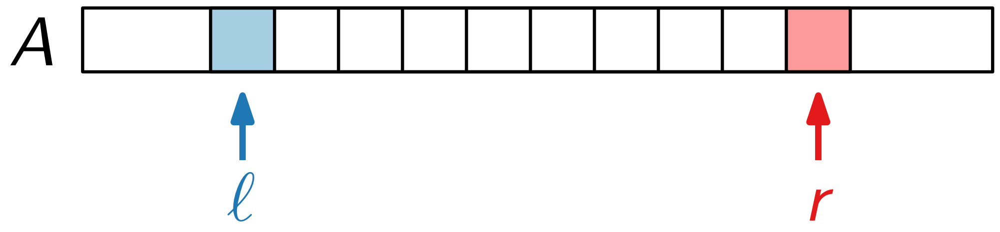
$\overset{\text{Randomized}}{}$QuickSort$(A, \ell = 1, r = A.length)$

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

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

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

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




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

RandomizedSelect(int[] $A$, int $\ell$, int $r$, int $i$)

**if** $\ell == r$ **then return** $A[\ell]$

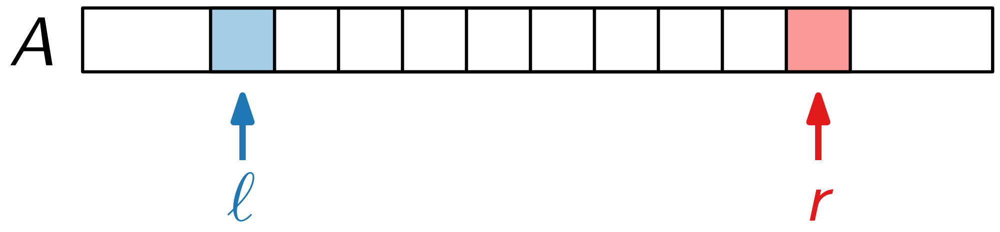
# Auswahl per Teile & Herrsche

Zur Erinnerung...

RANDOMIZED
QUICKSORT($A, \ell = 1, r = A.length$)

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

    $m =$ RANDOMIZED PARTITION($A, \ell, r$)
    QUICKSORT($A, \ell, m-1$)
    QUICKSORT($A, m+1, r$)




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

RANDOMIZEDSELECT(int[] $A$, int $\ell$, int $r$, int $i$)

**if** $\ell == r$ **then return** $A[\ell]$
$m =$ RANDOMIZEDPARTITION($A, \ell, r$)

# Auswahl per Teile & Herrsche

Zur Erinnerung...

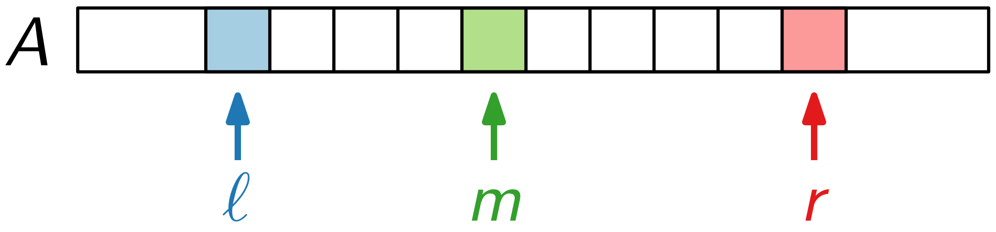
$\text{RANDOMIZED}$
$\text{QUICKSORT}(A, \ell = 1, r = A.length)$

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

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

$\text{QUICKSORT}(A, \ell, m - 1)$

$\text{QUICKSORT}(A, m + 1, r)$




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

$\text{RANDOMIZEDSELECT}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ r, \text{int}\ i)$

**if** $\ell == r$ **then return** $A[\ell]$

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

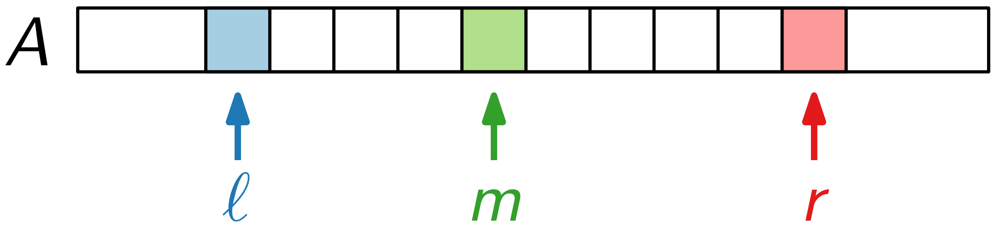
# Auswahl per Teile & Herrsche

Zur Erinnerung...

RANDOMIZED
QUICKSORT($A$, $\ell = 1$, $r = A.length$)

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

  RANDOMIZED
  $m =$ PARTITION($A, \ell, r$)
  QUICKSORT($A, \ell, m - 1$)
  QUICKSORT($A, m + 1, r$)




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

RANDOMIZEDSELECT(int[] $A$, int $\ell$, int $r$, int $i$)

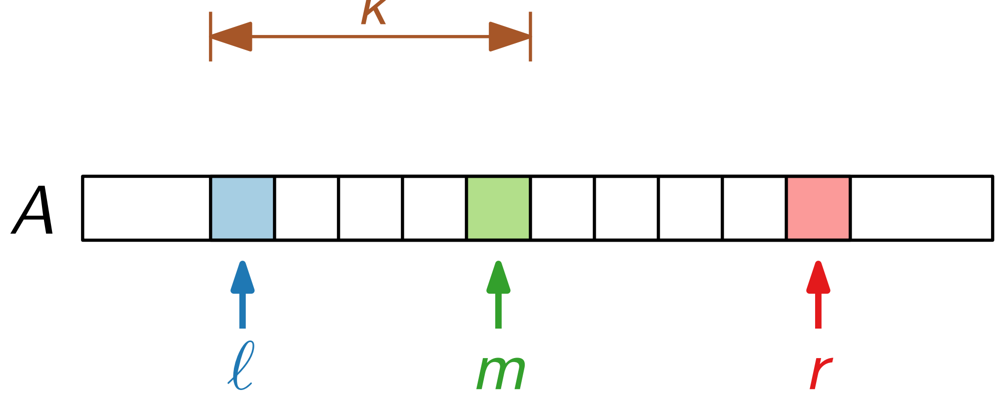
**if** $\ell == r$ **then return** $A[\ell]$
$m =$ RANDOMIZEDPARTITION($A, \ell, r$)
$k = m - \ell + 1$

# Auswahl per Teile & Herrsche

Zur Erinnerung...

$\text{Randomized}$
$\textsc{QuickSort}(A, \ell = 1, r = A.\mathit{length})$

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




Finde $i$.-kleinstes Element in $A[\ell \ldots r]$!

$\textsc{RandomizedSelect}(\text{int}[\,]\ A, \text{int } \ell, \text{int } r, \text{int } i)$

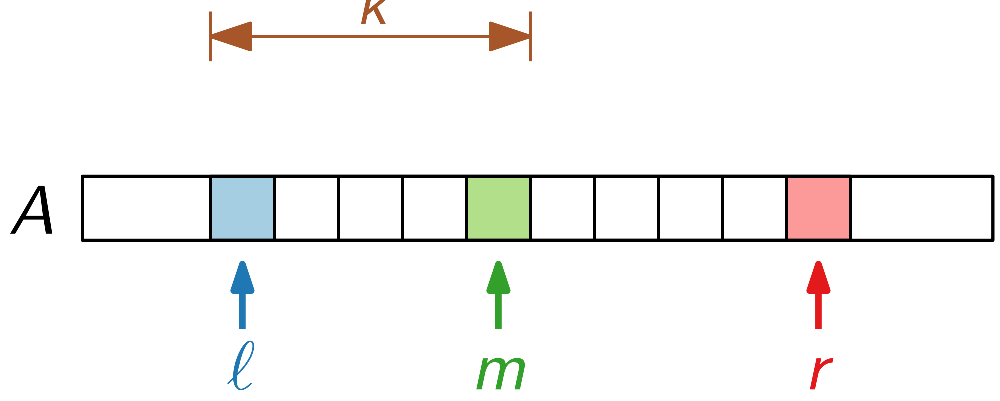
**if** $\ell == r$ **then return** $A[\ell]$
$m = \textsc{RandomizedPartition}(A, \ell, r)$
$k = m - \ell + 1$

# Auswahl per Teile & Herrsche

Zur Erinnerung...

RANDOMIZED
QUICKSORT($A, \ell = 1, r = A.length$)

**if** $\ell < r$ **then**
- $m =$ RANDOMIZED PARTITION($A, \ell, r$)
- QUICKSORT($A, \ell, m-1$)
- QUICKSORT($A, m+1, r$)




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

RANDOMIZEDSELECT(int[] $A$, int $\ell$, int $r$, int $i$)

**if** $\ell == r$ **then return** $A[\ell]$
$m =$ RANDOMIZEDPARTITION($A, \ell, r$)
$k = m - \ell + 1$ Rang von $A[m]$ in $A[\ell \dots r]$

# Auswahl per Teile & Herrsche

Zur Erinnerung...

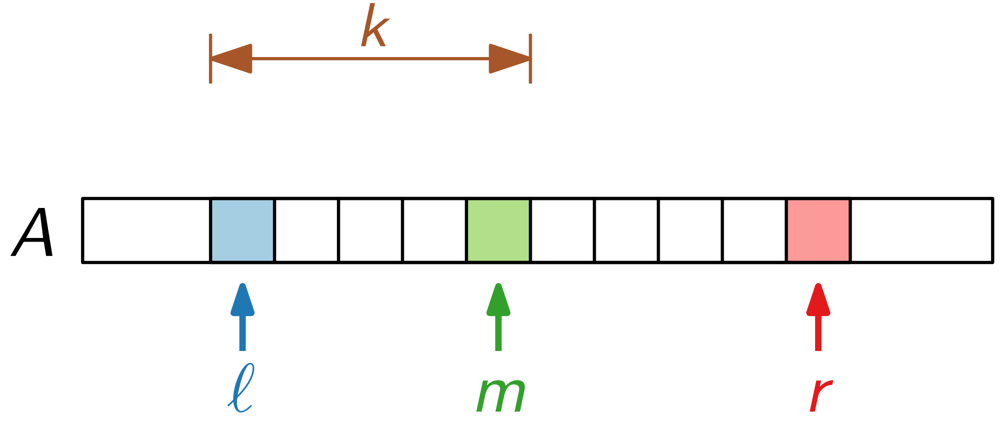
$\textsc{RandomizedQuickSort}(A, \ell = 1, r = A.\mathit{length})$

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

  $m = \textsc{RandomizedPartition}(A, \ell, r)$

  $\textsc{QuickSort}(A, \ell, m-1)$

  $\textsc{QuickSort}(A, m+1, r)$




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

$\textsc{RandomizedSelect}(\text{int}[\,]\,A, \text{int } \ell, \text{int } r, \text{int } i)$

**if** $\ell == r$ **then return** $A[\ell]$

$m = \textsc{RandomizedPartition}(A, \ell, r)$

$k = m - \ell + 1$ *Rang* von $A[m]$ in $A[\ell \dots r]$

**if** $i == k$ **then**

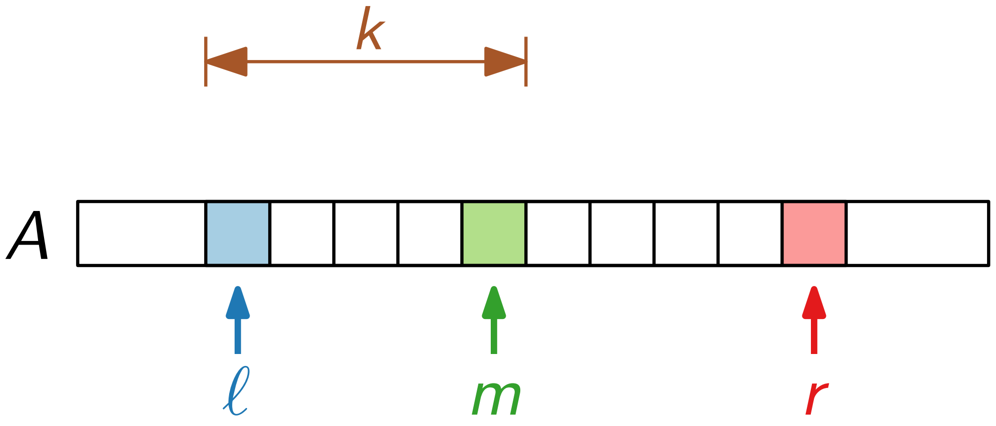
**else**

# Auswahl per Teile & Herrsche

Zur Erinnerung...

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

**if** $\ell < r$ **then**
- $m =$ RandomizedPartition($A, \ell, r$)
- QuickSort($A, \ell, m - 1$)
- QuickSort($A, m + 1, r$)




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

RandomizedSelect(int[] $A$, int $\ell$, int $r$, int $i$)

**if** $\ell == r$ **then return** $A[\ell]$
$m =$ RandomizedPartition($A, \ell, r$)
$k = m - \ell + 1$ — *Rang* von $A[m]$ in $A[\ell \dots r]$
**if** $i == k$ **then**
- **return** $A[m]$

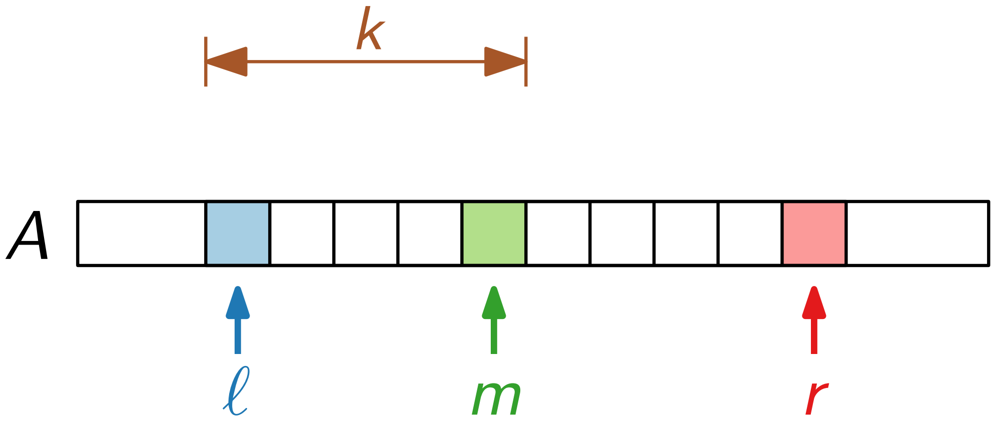
**else**

# Auswahl per Teile & Herrsche

Zur Erinnerung...

$\text{RANDOMIZED}$
QuickSort($A$, $\ell = 1$, $r = A.length$)

**if** $\ell < r$ **then**
  $m =$ RandomizedPartition($A, \ell, r$)
  QuickSort($A, \ell, m - 1$)
  QuickSort($A, m + 1, r$)




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

RandomizedSelect(int[] $A$, int $\ell$, int $r$, int $i$)

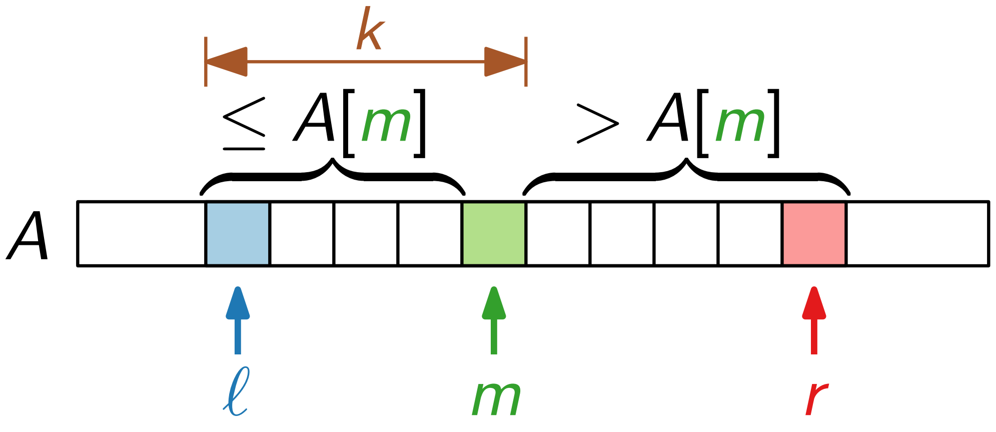
**if** $\ell == r$ **then return** $A[\ell]$
$m =$ RandomizedPartition($A, \ell, r$)
$k = m - \ell + 1$ $\boxed{\textit{Rang von } A[m] \text{ in } A[\ell \dots r]}$
**if** $i == k$ **then**
  **return** $A[m]$
**else**
  **if** $i < k$ **then**
  **else**

# Auswahl per Teile & Herrsche

Zur Erinnerung...

$\text{RANDOMIZED}$
$\text{QUICKSORT}(A, \ell = 1, r = A.length)$

**if** $\ell < r$ **then**
  $m = \text{RANDOMIZEDPARTITION}(A, \ell, r)$
  $\text{QUICKSORT}(A, \ell, m-1)$
  $\text{QUICKSORT}(A, m+1, r)$




Finde $i$.-kleinstes Element in $A[\ell \ldots r]$!

$\text{RANDOMIZEDSELECT}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ r, \text{int}\ i)$

**if** $\ell == r$ **then return** $A[\ell]$
$m = \text{RANDOMIZEDPARTITION}(A, \ell, r)$
$k = m - \ell + 1$ — *Rang* von $A[m]$ in $A[\ell \ldots r]$
**if** $i == k$ **then**
  **return** $A[m]$
**else**
  **if** $i < k$ **then**

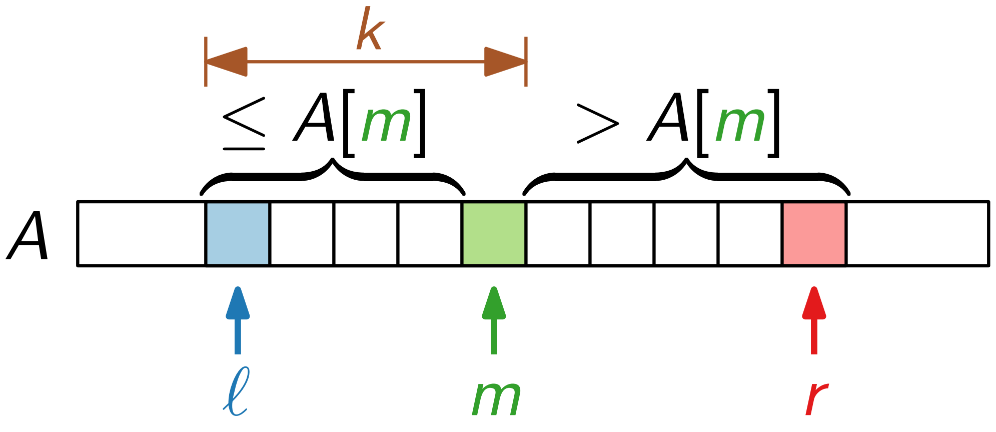
  **else**

# Auswahl per Teile & Herrsche

Zur Erinnerung...

RANDOMIZED QUICKSORT($A$, $\ell = 1$, $r = A.length$)

**if** $\ell < r$ **then**
- $m =$ RANDOMIZED PARTITION($A, \ell, r$)
- QUICKSORT($A, \ell, m - 1$)
- QUICKSORT($A, m + 1, r$)




Finde $i$.-kleinstes Element in $A[\ell \ldots r]$!

RANDOMIZEDSELECT(int[] $A$, int $\ell$, int $r$, int $i$)

**if** $\ell == r$ **then return** $A[\ell]$
$m =$ RANDOMIZEDPARTITION($A, \ell, r$)
$k = m - \ell + 1$ $\boxed{\textit{Rang von } A[m] \text{ in } A[\ell \ldots r]}$
**if** $i == k$ **then**
- **return** $A[m]$

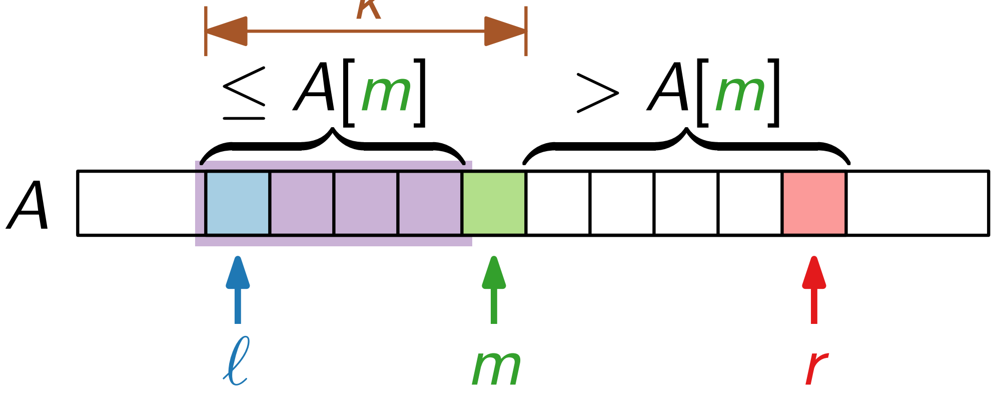
**else**
- **if** $i < k$ **then**
  - **return** RSELECT($A, \ell, m - 1, i$)
- **else**

# Auswahl per Teile & Herrsche

Zur Erinnerung...

RANDOMIZED
QUICKSORT($A$, $\ell = 1$, $r = A.length$)

**if** $\ell < r$ **then**
  $m$ = RANDOMIZED PARTITION($A, \ell, r$)
  QUICKSORT($A, \ell, m-1$)
  QUICKSORT($A, m+1, r$)




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

RANDOMIZEDSELECT(int[] $A$, int $\ell$, int $r$, int $i$)

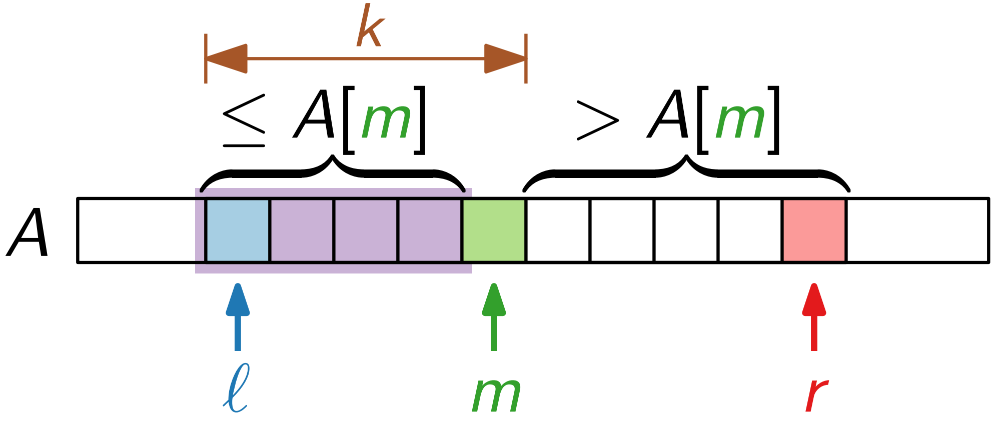
**if** $\ell == r$ **then return** $A[\ell]$
$m$ = RANDOMIZEDPARTITION($A, \ell, r$)
$k = m - \ell + 1$   *Rang* von $A[m]$ in $A[\ell \dots r]$
**if** $i == k$ **then**
  **return** $A[m]$
**else**
  **if** $i < k$ **then**
    **return** RSELECT($A, \ell, m-1, i$)
  **else**

# Auswahl per Teile & Herrsche

Zur Erinnerung...

$\text{RANDOMIZED}$ $\textsc{QuickSort}(A, \ell = 1, r = A.\mathit{length})$

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




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

$\textsc{RandomizedSelect}(\text{int}[]\, A, \text{int } \ell, \text{int } r, \text{int } i)$

**if** $\ell == r$ **then return** $A[\ell]$
$m = \textsc{RandomizedPartition}(A, \ell, r)$
$k = m - \ell + 1$ *Rang* von $A[m]$ in $A[\ell \dots r]$
**if** $i == k$ **then**
- **return** $A[m]$

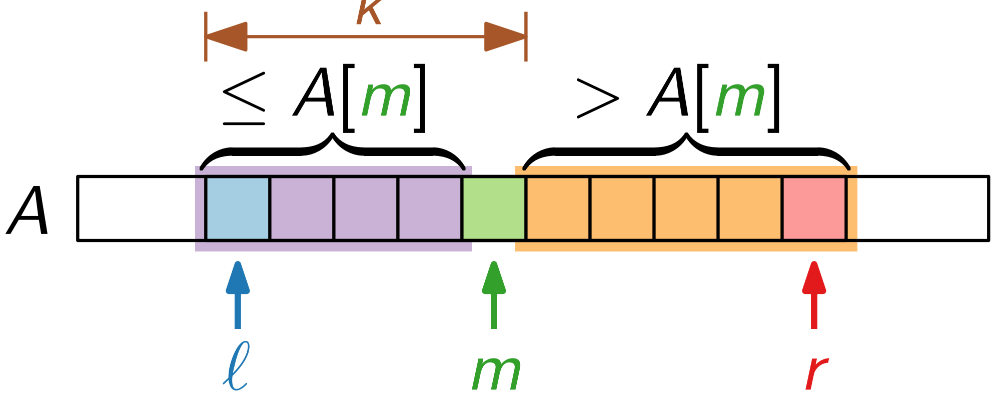
**else**
- **if** $i < k$ **then**
  - **return** $\textsc{RSelect}(A, \ell, m-1, i)$
- **else**
  - **return** $\textsc{RSelect}(A, m+1, r, i-k)$

# Auswahl per Teile & Herrsche

Zur Erinnerung...

$\text{RANDOMIZED}$ QuickSort($A$, $\ell = 1$, $r = A.length$)

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




Finde $i$.-kleinstes Element in $A[\ell \dots r]$!

RandomizedSelect(int[] $A$, int $\ell$, int $r$, int $i$)

**if** $\ell == r$ **then return** $A[\ell]$
$m$ = RandomizedPartition($A$, $\ell$, $r$)
$k = m - \ell + 1$ *Rang* von $A[m]$ in $A[\ell \dots r]$
**if** $i == k$ **then**
- **return** $A[m]$

**else**
- **if** $i < k$ **then**
  - **return** RSelect($A$, $\ell$, $m-1$, $i$)
- **else**
  - **return** RSelect($A$, $m+1$, $r$, $i-k$)

# Laufzeitanalyse

Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

# Laufzeitanalyse

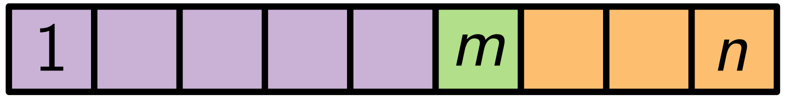
Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.

# Laufzeitanalyse

Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

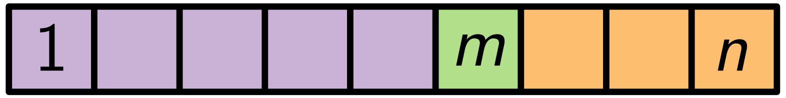
**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.



# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.

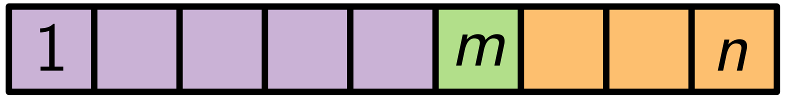



$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.



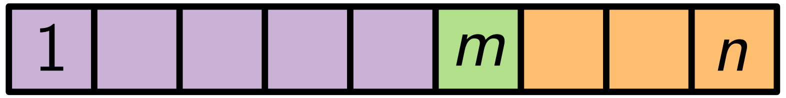

$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.



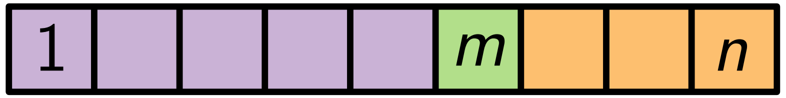

$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

# Laufzeitanalyse

Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

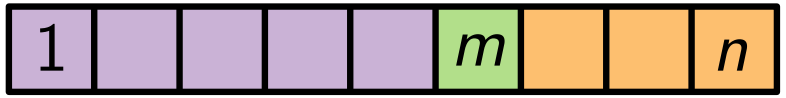
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$V(n) =$

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

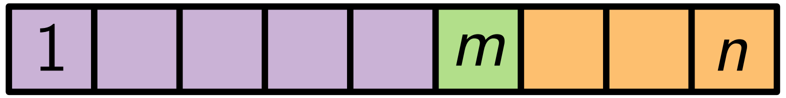
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$V(n) = V_{\text{Part}}(n) +$

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

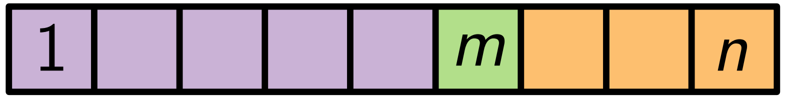
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = V_{\text{Part}}(n) + \begin{cases} V(n-1) & \text{falls } m = 1 \\ & \\ & \end{cases}$$

# Laufzeitanalyse

Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = V_{\text{Part}}(n) + \begin{cases} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ & \\ & \\ & \end{cases}$$

# Laufzeitanalyse

Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | $m$ | | | $n$ |

$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

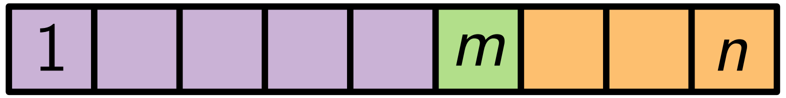
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = V_{\text{Part}}(n) + \begin{cases} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ & \\ & \\ & \end{cases}$$

# Laufzeitanalyse

Anzahl Vergleiche von $\textsc{RandomizedSelect}$ ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

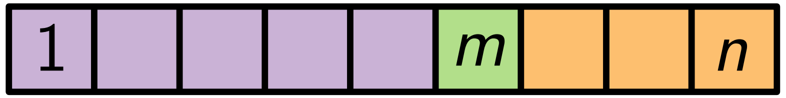
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = V_{\text{Part}}(n) + \begin{cases} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ & \end{cases}$$

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

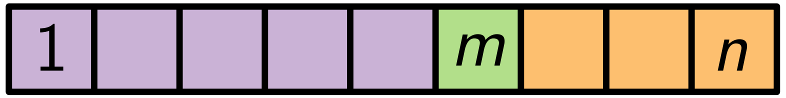
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = V_{\text{Part}}(n) + \begin{cases} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{cases}$$

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

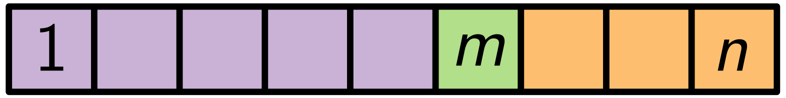
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\text{Part}}(n)}_{} + \begin{cases} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{cases}$$

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n-1} + \begin{cases} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{cases}$$

# Laufzeitanalyse

Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | $m$ | | | $n$ |

$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

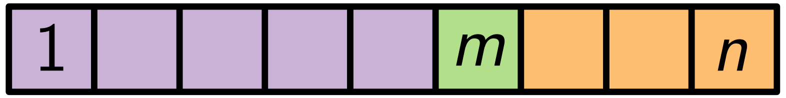
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\mathrm{Part}}(n)}_{= n-1} + \left\{\begin{array}{ll} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{array}\right\} \text{Alle Fälle gleich wahrscheinlich!}$$

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n-1} + \left\{ \begin{array}{ll} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{array} \right\} \text{Alle Fälle gleich wahrscheinlich!}$$

vorausgesetzt alle Elemente sind verschieden!

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.

| 1 | | | | | $m$ | | | $n$ |
|---|---|---|---|---|---|---|---|---|

$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

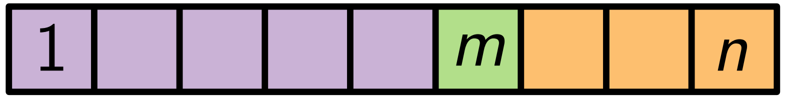
$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n-1} + \left. \begin{cases} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{cases} \right\} \text{Alle Fälle gleich wahrscheinlich!}$$

vorausgesetzt alle Elemente sind verschieden!

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n-1} + \left\{ \begin{array}{ll} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \ldots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \ldots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{array} \right\} \text{Alle Fälle gleich wahrscheinlich!}$$

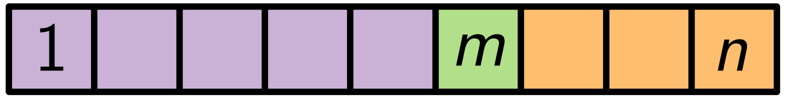
vorausgesetzt alle Elemente sind verschieden!

$\Rightarrow$ $E[V(n)] \leq$

# Laufzeitanalyse

Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n-1} + \left\{ \begin{array}{ll} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{array} \right\} \text{Alle Fälle gleich wahrscheinlich!}$$

vorausgesetzt alle Elemente sind verschieden!

$\Rightarrow$ $E[V(n)] \leq n - 1 +$

# Laufzeitanalyse

Anzahl Vergleiche von $\textsc{RandomizedSelect}$ ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | $m$ | | | $n$ |

$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n-1} + \left\{ \begin{array}{ll} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{array} \right\} \text{Alle Fälle gleich wahrscheinlich!}$$

vorausgesetzt alle Elemente sind verschieden!

$\Rightarrow$ $E[V(n)] \le n - 1 + 2 \cdot \frac{1}{n}$

# Laufzeitanalyse

Anzahl Vergleiche von RANDOMIZEDSELECT ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.

| 1 | | | | | $m$ | | | $n$ |
|---|---|---|---|---|---|---|---|---|

$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n-1} + \left\{ \begin{array}{ll} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \cdots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \cdots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{array} \right\}$$

Alle Fälle gleich wahrscheinlich!

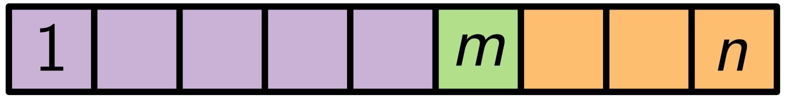
vorausgesetzt alle Elemente sind verschieden!

$$\Rightarrow \boxed{E[V(n)] \le n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1}}$$

# Laufzeitanalyse

Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n-1} + \begin{cases} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{cases} \Bigg\} \text{ Alle Fälle gleich wahrscheinlich!}$$

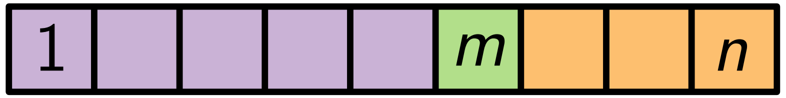
vorausgesetzt alle Elemente sind verschieden!

$$\Rightarrow \boxed{E[V(n)] \le n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]}$$

# Laufzeitanalyse

Anzahl Vergleiche von RandomizedSelect ist ZV; hängt von $n$ und $i$ ab.

**Trick.** Geh davon aus, dass das gesuchte $i$. Element immer im **größeren** Teilfeld liegt.




$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von $i$

$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n-1} + \left\{ \begin{array}{ll} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls } m = n \end{array} \right\} \text{Alle Fälle gleich wahrscheinlich!}$$

vorausgesetzt alle Elemente sind verschieden!

$$\Rightarrow \boxed{E[V(n)] \le n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]} \overset{?}{\le} c \cdot n \quad \left(\begin{array}{l}\text{für ein} \\ c > 0\end{array}\right)$$

# Substitutionsmethode

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$. Beweis per Induktion.

$$\text{Also:} \quad f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$$

laut Induktionsannahme

$$= n + \frac{2c}{n} \Big( \qquad \Big)$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$. Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$ laut Induktionsannahme

$$= n + \frac{2c}{n} \Big( \sum_{k=1}^{n-1} k \qquad\qquad \Big)$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$. Beweis per Induktion.

$$\text{Also:} \quad f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k \quad \text{laut Induktionsannahme}$$

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

$$
\begin{aligned}
\text{Also:} \quad f(n) &\leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k \\
&= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
&= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
&\leq n + \frac{c}{n} (n(n-1) - (n/2-1)(n/2-2))
\end{aligned}
$$

laut Induktionsannahme

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$
\begin{aligned}
&= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
&= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
&\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right) \\
&= n + c\left( n - 1 - n/4 + 3/2 - 2/n \right)
\end{aligned}
$$

$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$ laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$\leq n + \frac{c}{n} \left( n(n-1) - (n/2-1)(n/2-2) \right)$$

$$= n + c\left(n - 1 - n/4 + 3/2 - 2/n\right)$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$
\begin{aligned}
&= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
&= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
&\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right) \\
&= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c \cdot
\end{aligned}
$$

$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

$$\text{Also:} \quad f(n) \leq n + \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$$

laut Induktionsannahme

$$= n + \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k\right)$$

$$= n + \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor(\lfloor n/2 \rfloor - 1)}{2}\right)$$

$$\leq n + \frac{c}{n}(n(n-1) - (n/2-1)(n/2-2))$$

$$= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c \cdot$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right)$$

$$= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c \cdot \frac{3n+}{4}$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

$1) \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$
\begin{aligned}
&= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
&= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
&\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right) \\
&= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c \cdot \frac{3n+}{4}
\end{aligned}
$$

$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right)$$

$$= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c \cdot \frac{3n+2}{4}$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Beweis per Induktion.

laut Induktionsannahme

$$
\begin{aligned}
\text{Also:} \quad f(n) &\leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k \\
&= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
&= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
&\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right) \\
&= n + c\left( n - 1 - n/4 + 3/2 - 2/n \right) \leq n + c \cdot \frac{3n+2}{4} \\
&= cn + \left( n - c \cdot \frac{n-2}{4} \right)
\end{aligned}
$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \le n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \le cn$.

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Beweis per Induktion.

laut Induktionsannahme

$$
\begin{aligned}
\text{Also:} \quad f(n) &\le n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k \\
&= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
&= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
&\le n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right) \\
&= n + c(n - 1 - n/4 + 3/2 - 2/n) \le n + c \cdot \frac{3n+2}{4} \\
&= cn + \left(n - c \cdot \frac{n-2}{4}\right) \le cn
\end{aligned}
$$

$$E[V(n)] \le n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \le n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \le cn$.

Beweis per Induktion.

$$
\begin{aligned}
\text{Also:} \quad f(n) &\le n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k \qquad \text{laut Induktionsannahme} \\
&= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
&= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
&\le n + \frac{c}{n} \big( n(n-1) - (n/2 - 1)(n/2 - 2) \big) \\
&= n + c(n - 1 - n/4 + 3/2 - 2/n) \le n + c \cdot \frac{3n+2}{4} \\
&= cn + \underbrace{\left(n - c \cdot \frac{n-2}{4}\right)}_{\le 0?!} \le cn
\end{aligned}
$$

$$E[V(n)] \le n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

$$\text{Also:} \quad f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$$

laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right)$$

$$= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c \cdot \frac{3n+2}{4}$$

$$= cn + \left(n - c \cdot \frac{n-2}{4}\right) \leq cn \quad \text{falls } c \geq \frac{4n}{n-2} =$$

$\leq 0$?!

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$\leq n + \frac{c}{n} (n(n-1) - (n/2 - 1)(n/2 - 2))$$

$$= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c \cdot \frac{3n+2}{4}$$

$$= cn + \left(n - c \cdot \frac{n-2}{4}\right) \leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow[n \to \infty]{}$$

$\leq 0?!$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$ laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right)$$

$$= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c \cdot \frac{3n+2}{4}$$

$$= cn + \underbrace{\left(n - c \cdot \frac{n-2}{4}\right)}_{\leq 0?!} \leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow[n \to \infty]{} 4^{+}$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

$$1)\ \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$ **arithmetische Reihe**

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$. Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$ laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right)$$

$$= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c \cdot \frac{3n+2}{4}$$

$$= cn + \underbrace{\left(n - c \cdot \frac{n-2}{4}\right)}_{\leq 0?!} \leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow[n \to \infty]{} 4^+$$

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)] \leq \overbrace{(4+\varepsilon)}^{c :=} n$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \le n + \frac{2}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \le cn$.

Beweis per Induktion.

$$
\begin{aligned}
\text{Also:} \quad f(n) &\le n + \frac{2}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} c \cdot k \qquad \text{laut Induktionsannahme}\\
&= n + \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2\rfloor -1} k\right)\\
&= n + \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{\lfloor n/2\rfloor(\lfloor n/2\rfloor -1)}{2}\right)\\
&\le n + \frac{c}{n}\big(n(n-1) - (n/2-1)(n/2-2)\big)\\
&= n + c(n - 1 - n/4 + 3/2 - 2/n) \le n + c \cdot \frac{3n+2}{4}\\
&= cn + \underbrace{\left(n - c \cdot \frac{n-2}{4}\right)}_{\le 0?!} \le cn \quad \text{falls } c \ge \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow[n\to\infty]{} 4^+
\end{aligned}
$$

$$E[V(n)] \le n - 1 + 2\cdot\frac{1}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} E[V(k)] \le \overbrace{(4+\varepsilon)}^{c :=} n \qquad \text{falls } n \ge$$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $f(n) \leq n + \frac{2}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} c \cdot k$

laut Induktionsannahme

$$= n + \frac{2c}{n}\Big(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2\rfloor -1} k\Big)$$

$$= n + \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{\lfloor n/2\rfloor(\lfloor n/2\rfloor-1)}{2}\right)$$

$$\leq n + \frac{c}{n}\big(n(n-1) - (n/2-1)(n/2-2)\big)$$

$$= n + c(n - 1 - n/4 + 3/2 - 2/n) \leq n + c\cdot\frac{3n+2}{4}$$

$$= cn + \underbrace{\left(n - c\cdot\frac{n-2}{4}\right)}_{\leq 0?!} \leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow[n\to\infty]{} 4^{+}$$

$$E[V(n)] \leq n - 1 + 2\cdot\frac{1}{n}\sum_{k=\lfloor n/2\rfloor}^{n-1} E[V(k)] \leq \overbrace{(4+\varepsilon)}^{c:=}n$$

falls $n \geq \frac{8}{\varepsilon} + 2$

# Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

Dann gilt $f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} f(k)$

1) $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ **arithmetische Reihe**

Wir wollen prüfen, ob es ein $c > 0$ gibt, so dass $f(n) \leq cn$.

Beweis per Induktion.

Also: $$f(n) \leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k$$

laut Induktionsannahme

$$= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right)$$

$$= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right)$$

$$\leq n + \frac{c}{n} \left( n(n-1) - (n/2 - 1)(n/2 - 2) \right)$$

$$= n + c\left(n - 1 - n/4 + 3/2 - 2/n\right) \leq n + c \cdot \frac{3n+2}{4}$$

$$= cn + \left(n - c \cdot \frac{n-2}{4}\right) \leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow[n \to \infty]{} 4^+$$

$\leq 0$?!

Für jedes $\varepsilon > 0$ gilt:

$$E[V(n)] \leq n - 1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)] \leq \underbrace{(4 + \varepsilon)}_{c :=} n$$

falls $n \geq \frac{8}{\varepsilon} + 2$

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann in erwartet linearer Zeit gelöst werden.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann in erwartet linearer Zeit gelöst werden.

**Genauer:**

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann in erwartet linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq \frac{8}{\varepsilon} + 2$ Zahlen die $i$.-kleinste Zahl $(1 \leq i \leq n)$ mit erwartet $(4 + \varepsilon)n$ Vergleichen finden kann.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann in erwartet linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq \frac{8}{\varepsilon} + 2$ Zahlen die $i$.-kleinste Zahl ($1 \leq i \leq n$) mit erwartet $(4 + \varepsilon)n$ Vergleichen finden kann.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann in erwartet linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq \frac{8}{\varepsilon} + 2$ Zahlen die $i$.-kleinste Zahl ($1 \leq i \leq n$) mit erwartet $(4 + \varepsilon)n$ Vergleichen finden kann.

**Frage:**

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann in erwartet linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq \frac{8}{\varepsilon} + 2$ Zahlen die $i$.-kleinste Zahl $(1 \leq i \leq n)$ mit erwartet $(4 + \varepsilon)n$ Vergleichen finden kann.

**Frage:** Geht das auch **deterministisch**, d.h. ohne Zufall?

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann in erwartet linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq \frac{8}{\varepsilon} + 2$ Zahlen die $i$.-kleinste Zahl $(1 \leq i \leq n)$ mit erwartet $(4 + \varepsilon)n$ Vergleichen finden kann.

**Frage:** Geht das auch **deterministisch**, d.h. ohne Zufall?

**M.a.W.** Kann man das Auswahlproblem auch im **schlechtesten Fall** in linearer Zeit lösen?

# Vorbereitung

Wir verwenden wieder Divide & Conquer

# Vorbereitung

Wir verwenden wieder Divide & Conquer –
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

# Vorbereitung

Wir verwenden wieder Divide & Conquer –
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**

# Vorbereitung

Wir verwenden wieder Divide & Conquer –

aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**

jede Seite sollte $\geq \gamma n$ Elem. enthalten, für ein festes $0 < \gamma \leq \frac{1}{2}$.

# Vorbereitung

Wir verwenden wieder Divide & Conquer –
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**
jede Seite sollte $\geq \gamma n$ Elem. enthalten, für ein festes $0 < \gamma \leq \frac{1}{2}$.

```
PARTITION(int[] A, int ℓ, int r)
  pivot = A[r]
  i = ℓ
  for j = ℓ to r − 1 do
      if A[j] ≤ pivot then
          A[i] ↔ A[j]
          i = i + 1
  A[i] ↔ A[r]
  return i
```

# Vorbereitung

Wir verwenden wieder Divide & Conquer –

aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**

jede Seite sollte $\geq \gamma n$ Elem. enthalten, für ein festes $0 < \gamma \leq \frac{1}{2}$.

```
Partition'(int[] A, int ℓ, int r)
  pivot = A[r]
  i = ℓ
  for j = ℓ to r − 1 do
      if A[j] ≤ pivot then
          A[i] ↔ A[j]
          i = i + 1
  A[i] ↔ A[r]
  return i
```

# Vorbereitung

Wir verwenden wieder Divide & Conquer –
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**

jede Seite sollte $\geq \gamma n$ Elem. enthalten, für ein festes $0 < \gamma \leq \frac{1}{2}$.

```
PARTITION'(int[] A, int ℓ, int r, int pivot)
  pivot = A[r]
  i = ℓ
  for j = ℓ to r − 1 do
      if A[j] ≤ pivot then
          A[i] ↔ A[j]
          i = i + 1
  A[i] ↔ A[r]
  return i
```

# Vorbereitung

Wir verwenden wieder Divide & Conquer –
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**

jede Seite sollte $\geq \gamma n$ Elem. enthalten, für ein festes $0 < \gamma \leq \frac{1}{2}$.

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

~~$pivot = A[r]$~~
$i = \ell$
**for** $j = \ell$ **to** $r \cancel{- 1}$ **do**
  **if** $A[j] \leq pivot$ **then**
    $A[i] \leftrightarrow A[j]$
    $i = i + 1$
~~$A[i] \leftrightarrow A[r]$~~
**return** $i$

# Vorbereitung

Wir verwenden wieder Divide & Conquer –
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**

jede Seite sollte $\geq \gamma n$ Elem. enthalten, für ein festes $0 < \gamma \leq \frac{1}{2}$.

PARTITION′(int[] $A$, int $\ell$, int $r$, int $pivot$)

$\cancel{pivot = A[r]}$
$i = \ell$
**for** $j = \ell$ **to** $r \cancel{- 1}$ **do**
  **if** $A[j] \leq pivot$ **then**
    $A[i] \leftrightarrow A[j]$
    $i = i + 1$
$\cancel{A[i] \leftrightarrow A[r]}$
**return** $i$

Wir gehen für die Analyse wieder davon aus, dass alle Elemente verschieden sind.

# Vorbereitung

Wir verwenden wieder Divide & Conquer –
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**

jede Seite sollte $\geq \gamma n$ Elem. enthalten, für ein festes $0 < \gamma \leq \frac{1}{2}$.

$\textsc{Partition}'$(int[] $A$, int $\ell$, int $r$, int $pivot$)

~~$pivot = A[r]$~~
$i = \ell$
**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** $i$

Wir gehen für die Analyse wieder davon aus, dass alle Elemente verschieden sind.

Anzahl der Vergleiche, die $\textsc{Partition}'$ macht:

# Vorbereitung

Wir verwenden wieder Divide & Conquer –

aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**

jede Seite sollte $\geq \gamma n$ Elem. enthalten, für ein festes $0 < \gamma \leq \frac{1}{2}$.

$\textsc{Partition}'$(int[] $A$, int $\ell$, int $r$, int $pivot$)

~~$pivot = A[r]$~~

$i = \ell$

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

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

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

    $i = i + 1$

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

**return** $i$

Wir gehen für die Analyse wieder davon aus, dass alle Elemente verschieden sind.

Anzahl der Vergleiche, die $\textsc{Partition}'$ macht: $r - \ell + 1 =$

# Vorbereitung

Wir verwenden wieder Divide & Conquer –
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. **balanciert:**

jede Seite sollte $\geq \gamma n$ Elem. enthalten, für ein festes $0 < \gamma \leq \frac{1}{2}$.

$\textsc{Partition}'$(int[] $A$, int $\ell$, int $r$, int $pivot$)

  $\cancel{pivot = A[r]}$
  $i = \ell$
  **for** $j = \ell$ **to** $r \cancel{- 1}$ **do**
    **if** $A[j] \leq pivot$ **then**
      $A[i] \leftrightarrow A[j]$
      $i = i + 1$
  $\cancel{A[i] \leftrightarrow A[r]}$
  **return** $i$

Wir gehen für die Analyse wieder davon aus, dass alle Elemente verschieden sind.

Anzahl der Vergleiche, die $\textsc{Partition}'$ macht: $r - \ell + 1 = n$

# SELECT: deterministisch

SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

# SELECT: deterministisch

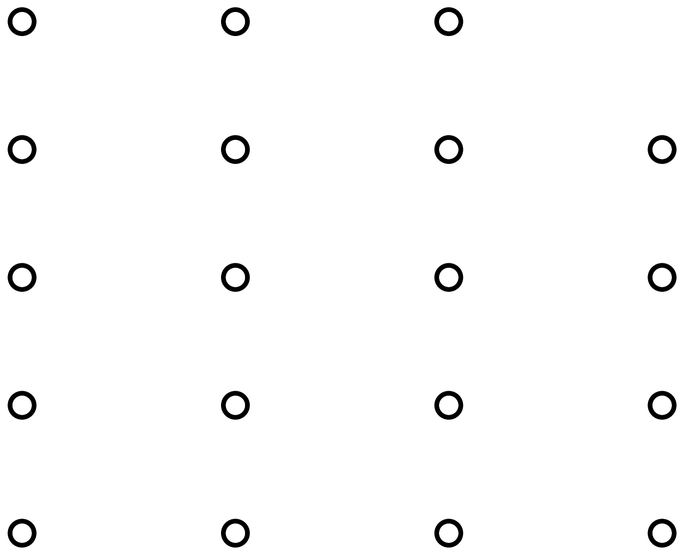
SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

# Select: deterministisch

Select(int[] $A$, int $\ell$, int $r$, int $i$)

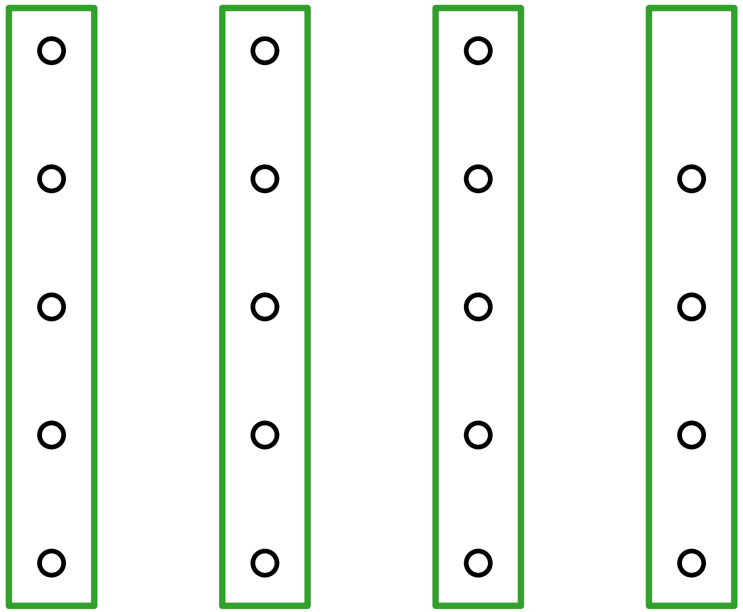
1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n \bmod 5$) Elementen.



# SELECT: deterministisch

SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n \bmod 5$) Elementen.

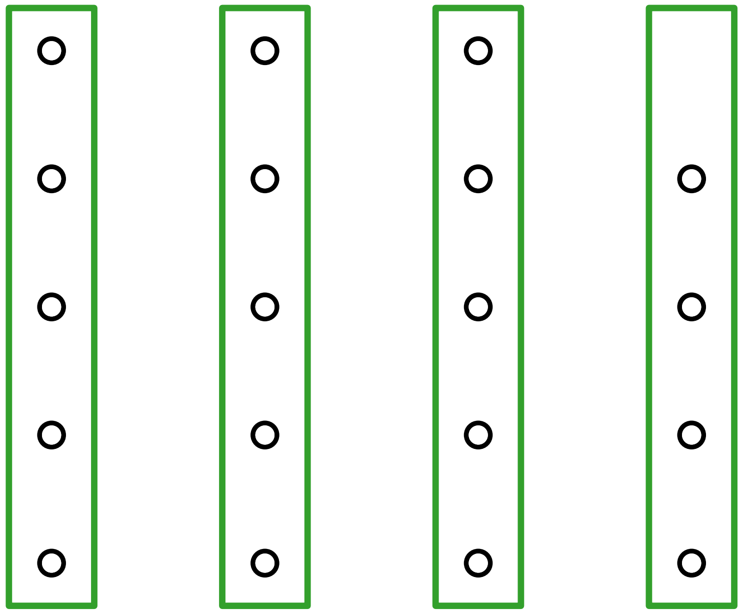


# SELECT: deterministisch

SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n \bmod 5$) Elementen.

2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

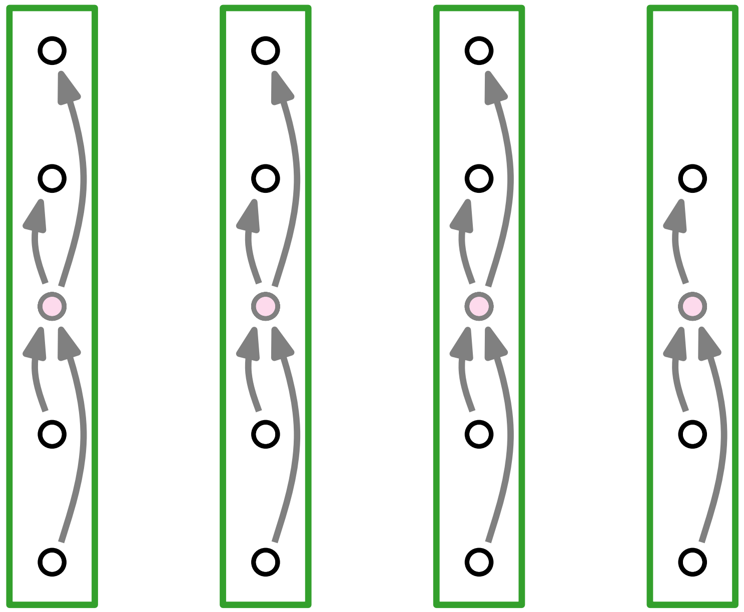


# SELECT: deterministisch

SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.



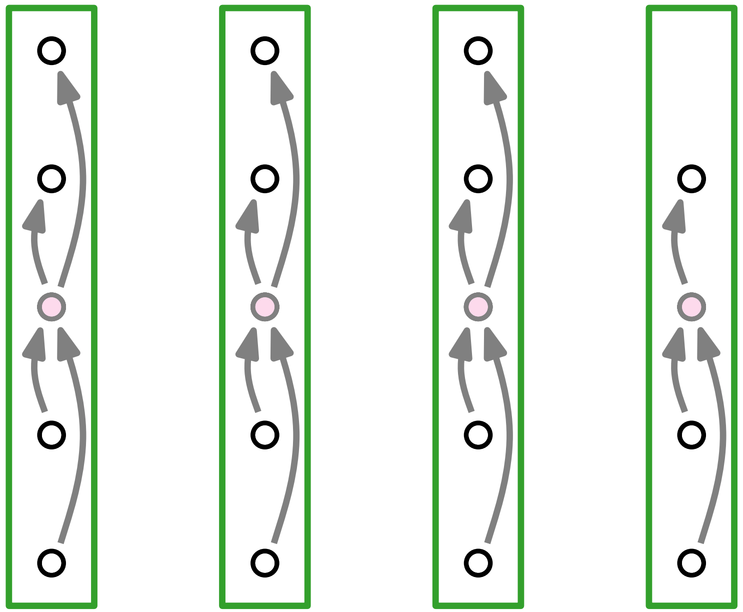
# SELECT: deterministisch

SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

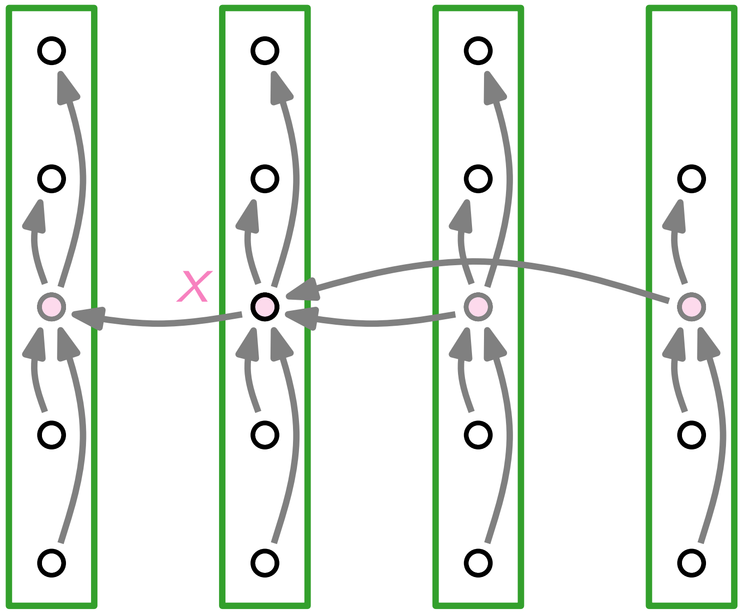
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.



# Select: deterministisch

Select(int[] $A$, int $\ell$, int $r$, int $i$)

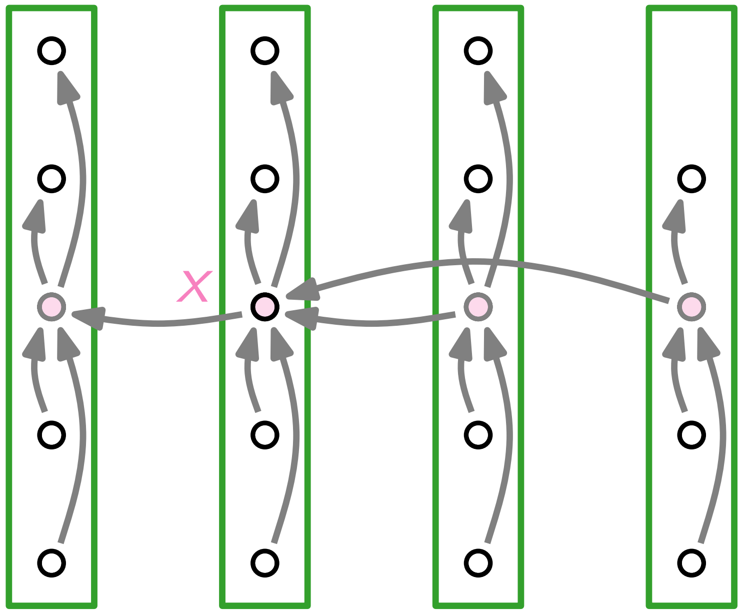
1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.



# SELECT: deterministisch

SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.
4. $m = \text{PARTITION}'(A, \ell, r, x)$



# $\textsc{Select}$: deterministisch

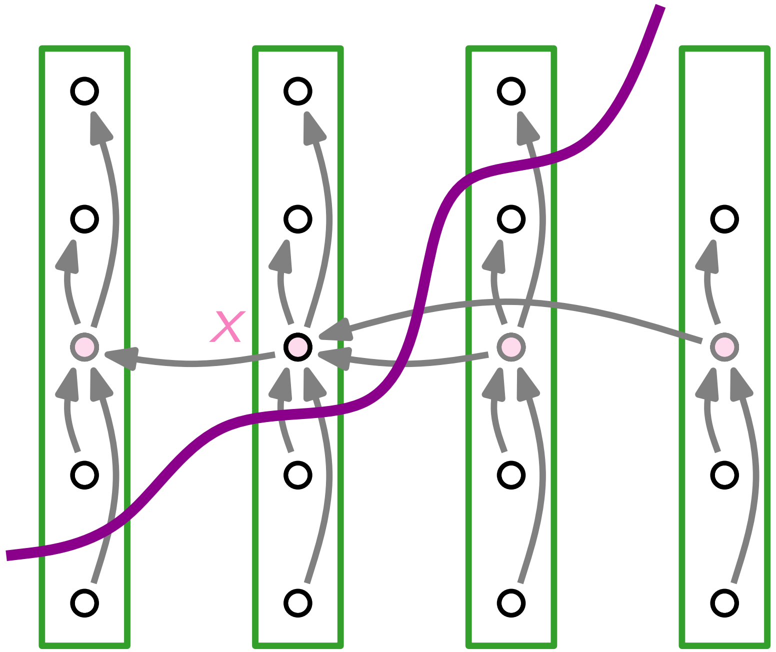
$\textsc{Select}$(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.

4. $m = \textsc{Partition}'(A, \ell, r, x)$



# Select: deterministisch

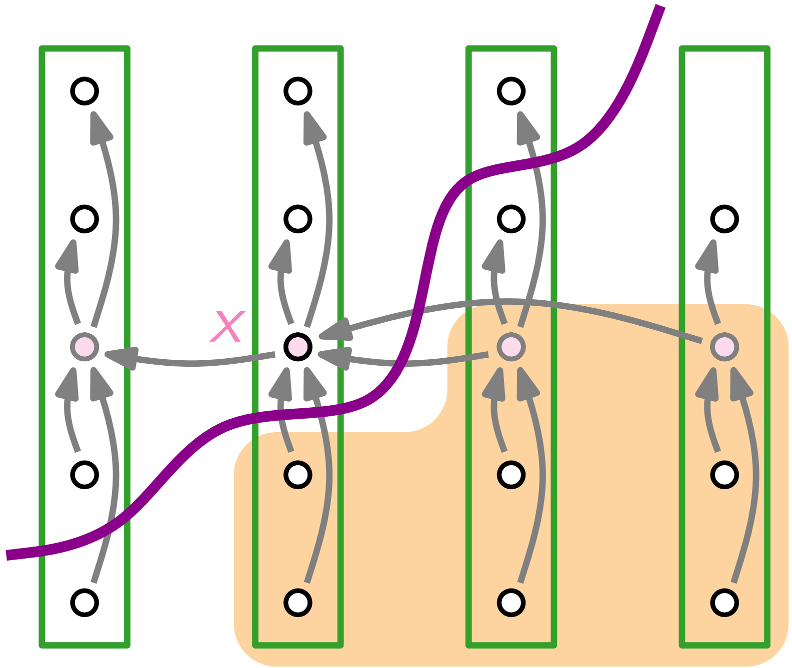
Select(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n \bmod 5$) Elementen.

2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.

4. $m = \textsc{Partition}'(A, \ell, r, x)$



# SELECT: deterministisch

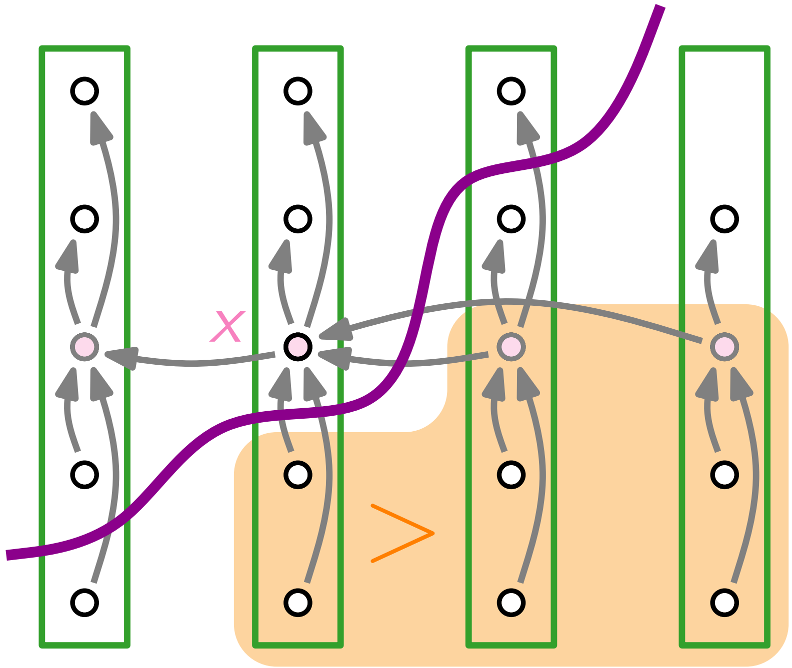
SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.

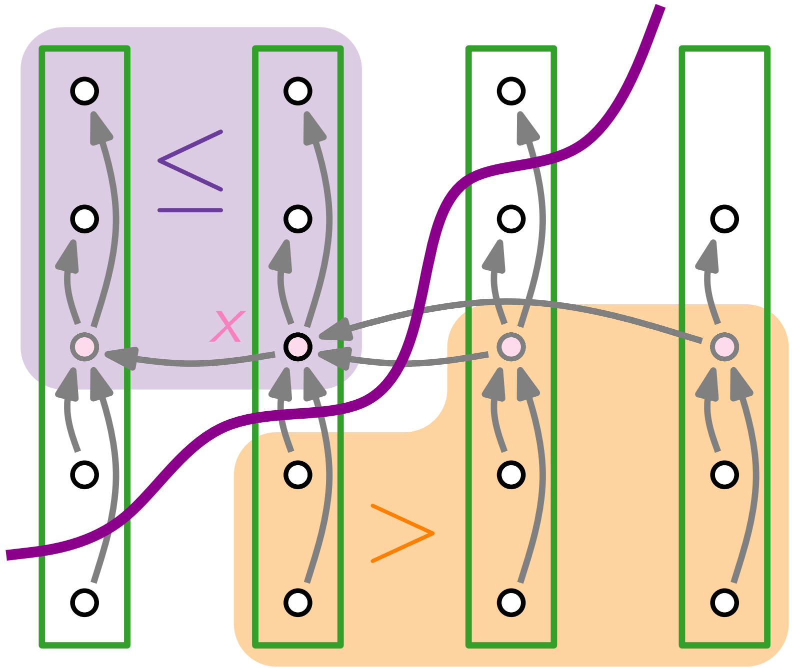
4. $m = \text{PARTITION}'(A, \ell, r, x)$



# SELECT: deterministisch

SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

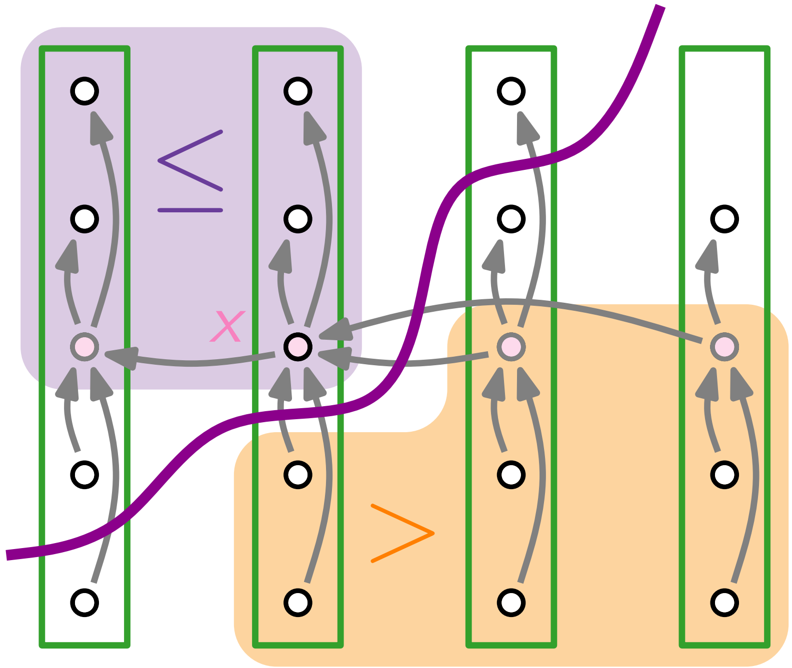
1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n \bmod 5$) Elementen.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.
4. $m = \text{PARTITION}'(A, \ell, r, x)$



# $\textsc{Select}$: deterministisch

$\textsc{Select}$(int[] $A$, int $\ell$, int $r$, int $i$)

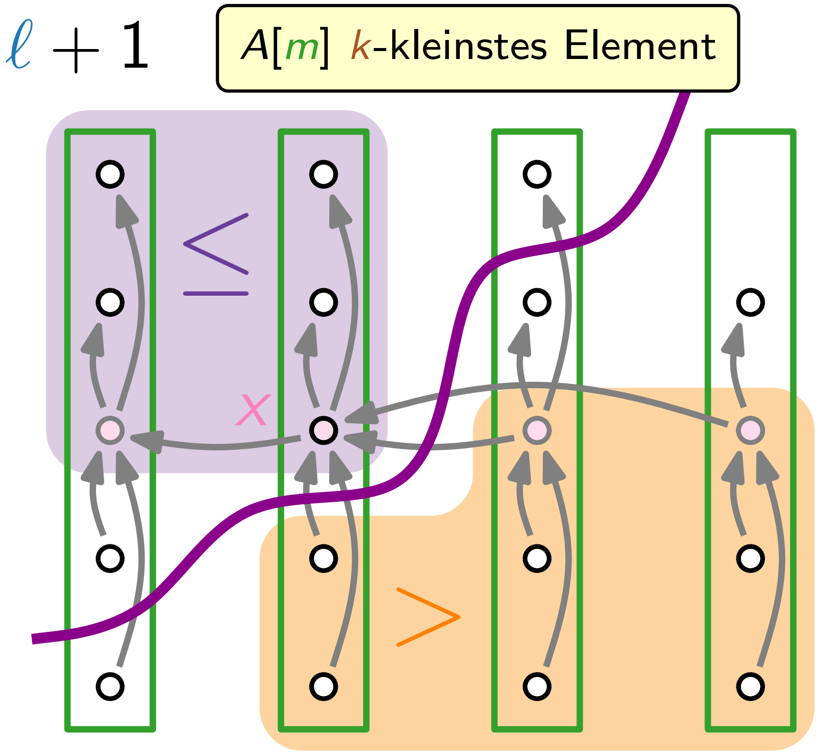
1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.
4. $m = \textsc{Partition}'(A, \ell, r, x)$; $k = m - \ell + 1$



# Select: deterministisch

Select(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.
4. $m = \text{Partition}'(A, \ell, r, x)$; $k = m - \ell + 1$



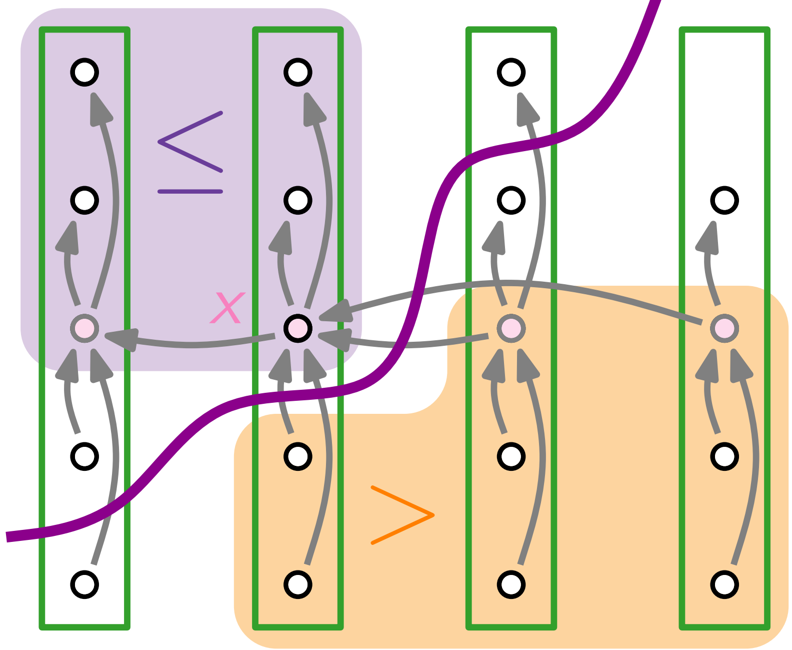
# SELECT: deterministisch

SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.
4. $m = \text{PARTITION}'(A, \ell, r, x)$; $k = m - \ell + 1$

$A[m]$ $k$-kleinstes Element

5. **if** $i == k$ **then return** $A[m]$
   **else**
   **if** $i < k$ **then**
   **return** SELECT$(A, \ell, m-1, i)$
   **else**
   **return** SELECT$(A, m+1, r, i-k)$



# Select: deterministisch

$\textsc{Select}(\text{int}[\,]\ A,\ \text{int}\ \ell,\ \text{int}\ r,\ \text{int}\ i)$

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

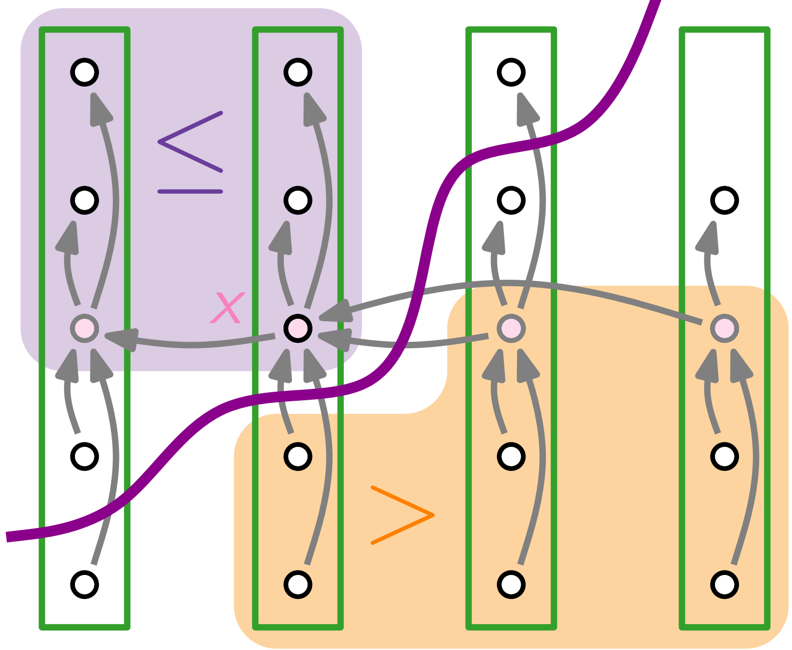
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.

4. $m = \textsc{Partition}'(A, \ell, r, x)$; $k = m - \ell + 1$

   $A[m]$ $k$-kleinstes Element

5. **if** $i == k$ **then return** $A[m]$
   
   $x$
   
   **else**
   
     **if** $i < k$ **then**
   
       **return** $\textsc{Select}(A, \ell, m-1, i)$
   
     **else**
   
       **return** $\textsc{Select}(A, m+1, r, i-k)$



# Select: deterministisch

Select(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

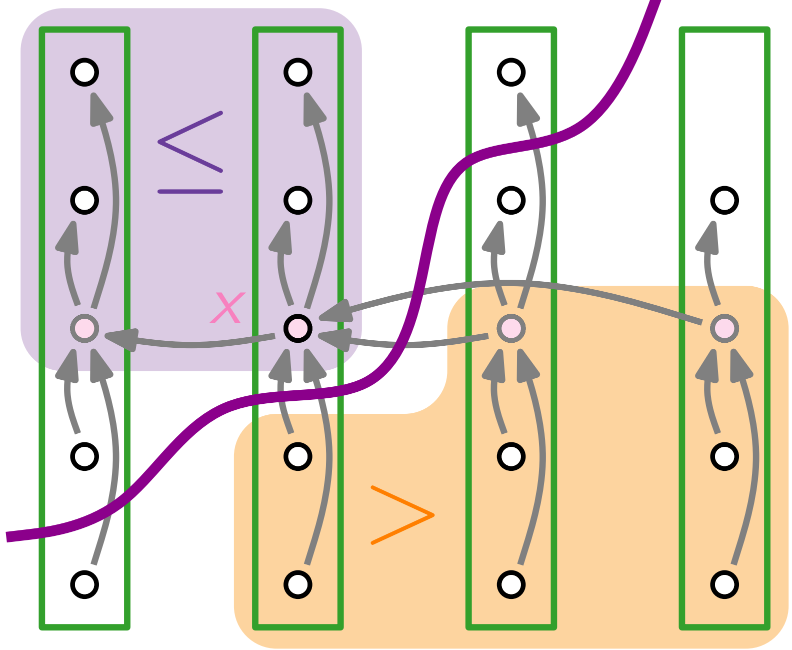
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.

4. $m = \text{Partition}'(A, \ell, r, x)$; $k = m - \ell + 1$

   $A[m]$ $k$-kleinstes Element

5. **if** $i == k$ **then return** $A[m]$
   **else**
       **if** $i < k$ **then**
           **return** $\text{Select}(A, \ell, m-1, i)$
       **else**
           **return** $\text{Select}(A, m+1, r, i-k)$

   $x$




Anzahl(●) $\geq$

# Select: deterministisch

Select(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

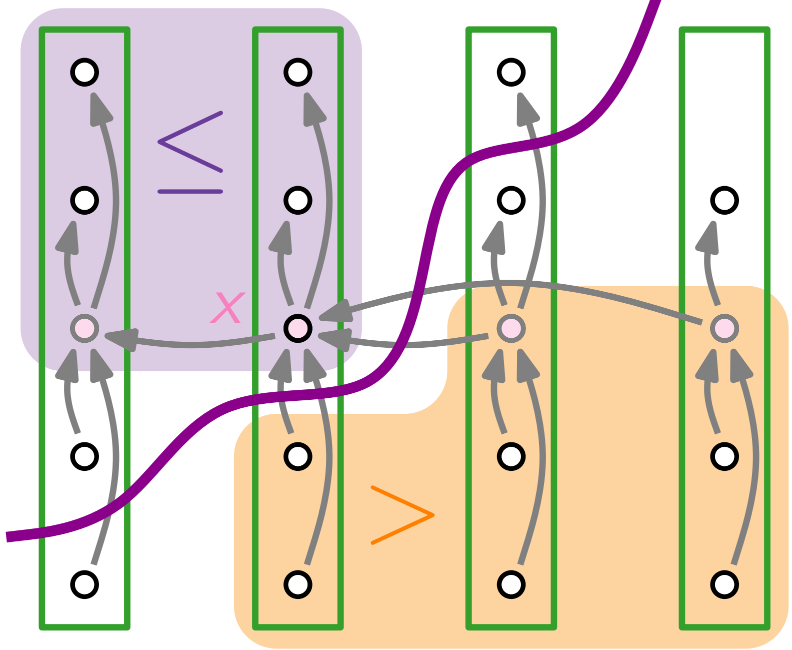
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.

4. $m = \text{Partition}'(A, \ell, r, x)$; $k = m - \ell + 1$

   $A[m]$ $k$-kleinstes Element

5. **if** $i == k$ **then return** $A[m]$
   
   $x$
   
   **else**
   
       **if** $i < k$ **then**
   
           **return** Select($A, \ell, m-1, i$)
   
       **else**
   
           **return** Select($A, m+1, r, i-k$)




$\text{Anzahl}(\bullet) \geq 3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right)$

# Select: deterministisch

$\textsc{Select}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ r, \text{int}\ i)$

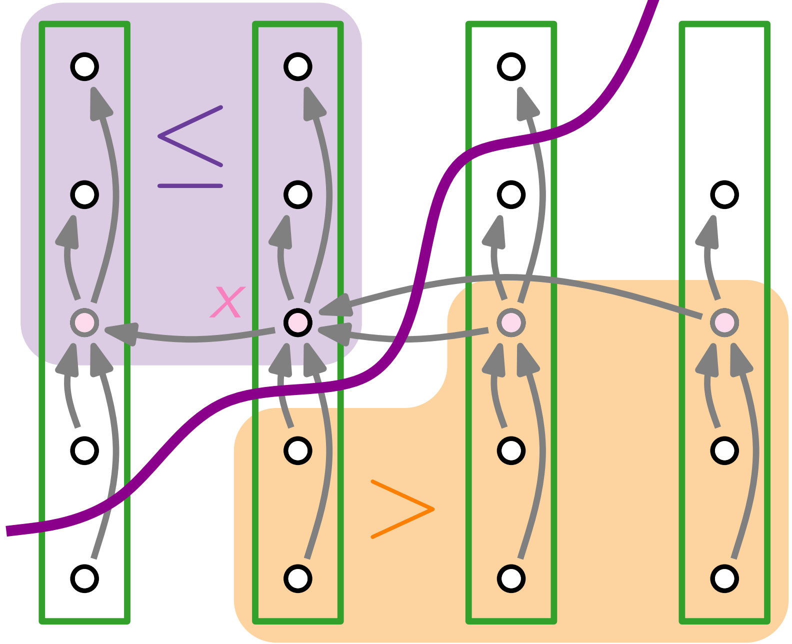
1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.

4. $m = \textsc{Partition}'(A, \ell, r, x)$; $k = m - \ell + 1$ $\quad$ $A[m]$ $k$-kleinstes Element

5. **if** $i == k$ **then return** $A[m]$ $\quad$ $x$
   **else**
   $\quad$ **if** $i < k$ **then**
   $\quad\quad$ **return** $\textsc{Select}(A, \ell, m-1, i)$
   $\quad$ **else**
   $\quad\quad$ **return** $\textsc{Select}(A, m+1, r, i-k)$

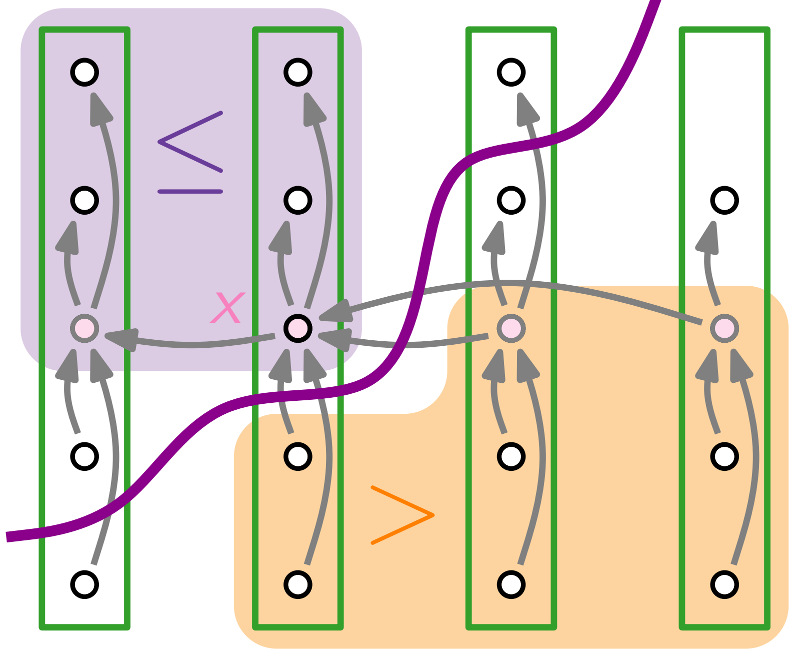



Anzahl(●) $\geq 3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq \frac{3n}{10} - 3$

# Select: deterministisch

Select(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n \bmod 5$) Elementen.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.
4. $m = \text{Partition}'(A, \ell, r, x)$; $k = m - \ell + 1$ $A[m]$ $k$-kleinstes Element
5. **if** $i == k$ **then return** $A[m]$ $x$
   **else**
       **if** $i < k$ **then**
           **return** Select($A, \ell, m-1, i$)
       **else**
           **return** Select($A, m+1, r, i-k$)

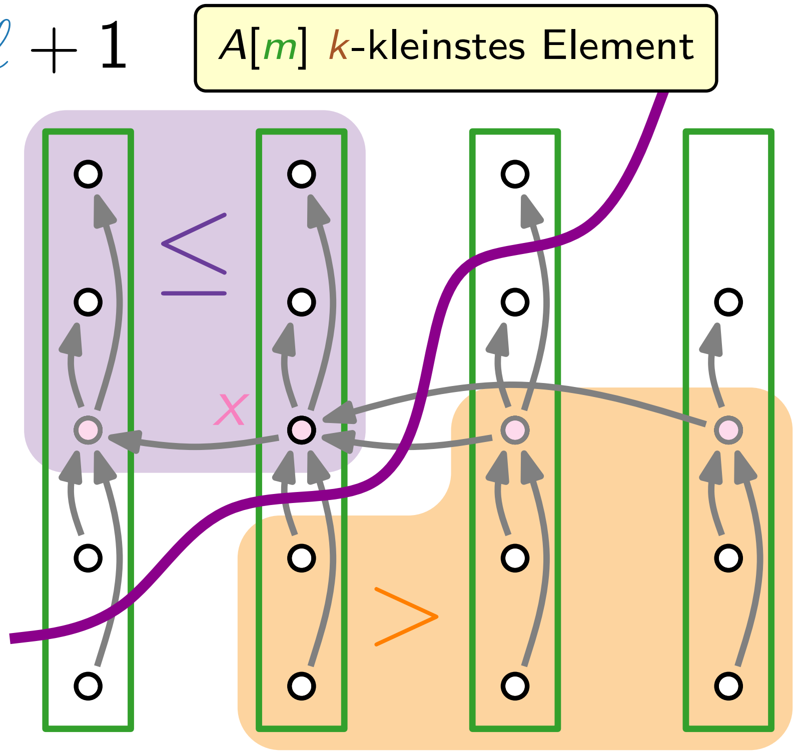



$\text{Anzahl}(\bullet) \geq 3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq \frac{3n}{10} - 3$

# Select: deterministisch

$\textsc{Select}(\text{int}[\,]\ A, \text{int}\ \ell, \text{int}\ r, \text{int}\ i)$

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.
4. $m = \textsc{Partition}'(A, \ell, r, x)$; $k = m - \ell + 1$
5. **if** $i == k$ **then return** $A[m]$
   
   $x$
   
   **else**
   
   **if** $i < k$ **then**
   
   **return** $\textsc{Select}(A, \underbrace{\ell, m-1}, i)$
   
   **else**
   
   **return** $\textsc{Select}(A, \overbrace{m+1, r}, i-k)$




$\text{Anzahl}(\bullet) \geq 3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq \frac{3n}{10} - 3$

# SELECT: deterministisch

SELECT(int[] $A$, int $\ell$, int $r$, int $i$)

1. Teile die $n$ Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n$ mod 5) Elementen.

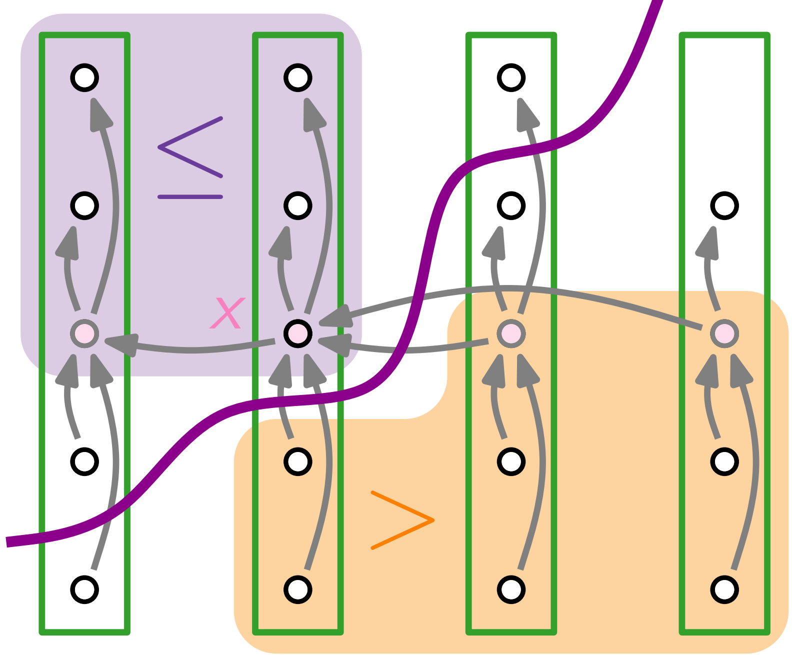
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.

3. Bestimme **rekursiv** den Median $x$ der Gruppen-Mediane.

4. $m = \text{PARTITION}'(A, \ell, r, x)$; $k = m - \ell + 1$ 　 $A[m]$ $k$-kleinstes Element

5. **if** $i == k$ **then return** $A[m]$ 　 $x$
   **else**
   　**if** $i < k$ **then**
   　　**return** $\text{SELECT}(A, \underbrace{\ell, m-1}, i)$
   　**else**
   　　**return** $\text{SELECT}(A, \overbrace{m+1, r}, i-k)$

   $\le 7n/10 + 3$ Elemente




Anzahl($\bullet$) $\ge 3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \ge \frac{3n}{10} - 3$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$:

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren:

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathsf{IS}}(5) =$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathsf{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$V(n) \leq$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} & \text{falls } n \geq n_0, \\ & \end{cases}$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} & \text{falls } n \ge n_0, \\ & \end{cases}$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathsf{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} & \text{falls } n \geq n_0, \\ & \end{cases}$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} & \text{falls } n \geq n_0, \\ & \end{cases}$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \end{cases}$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$\Rightarrow \; V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \quad V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

PARTITION′: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$$\Rightarrow \; V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 \qquad )$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

PARTITION′: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \; V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 \qquad )$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \quad V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4)$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$$\Rightarrow \quad V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4)$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \quad V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10+3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le {\color{red}cn}$.

$$\Rightarrow \quad V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \quad = {\color{red}cn} + \big(3n - c \cdot (n/10 - 4)\big)$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \; V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \; = cn + \big(3n - c \cdot (n/10 - 4)\big)$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

Partition′: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10+3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \; V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \;\; = cn + \big(\underbrace{3n - c \cdot (n/10 - 4)}_{\le 0?!}\big)$$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10+3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \quad V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \quad = cn + \big(\overbrace{3n - c \cdot (n/10 - 4)}^{\le 0?!}\big)$$

falls $c \ge \frac{3n}{n/10-4} =$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

PARTITION$'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n = cn + \underbrace{(3n - c \cdot (n/10 - 4))}_{\le 0?!}$$

falls $c \ge \frac{3n}{n/10 - 4} = \frac{30}{1 - 40/n} \xrightarrow[n \to \infty]{}$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10+3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$$\Rightarrow V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n = cn + \big(\underbrace{3n - c \cdot (n/10 - 4)}_{\leq 0?!}\big)$$

falls $c \geq \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n \to \infty]{} 30^+$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10+3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$$\Rightarrow \quad V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \quad = cn + \big(3n - c \cdot (n/10 - 4)\big)$$

$\leq 0$?!

falls $c \geq \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n\to\infty]{} 30^+$

$\boxed{\text{bzw. } n \geq \frac{40c}{c-30}}$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$$\Rightarrow V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n = cn + \big(\underbrace{3n - c \cdot (n/10 - 4)}_{\leq 0?!}\big)$$

falls $c \geq \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n \to \infty]{} 30^+$ $\boxed{\text{bzw. } n \geq \frac{40c}{c-30}}$

$c \geq \frac{3n}{n/10-4}$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \; V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \; = cn + \big(\underbrace{3n - c \cdot (n/10 - 4)}_{\le 0?!}\big)$$

falls $c \ge \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n \to \infty]{} 30^+$

$\boxed{\text{bzw. } \; n \ge \frac{40c}{c-30}}$

$c \ge \frac{3n}{n/10-4}$

$\Leftrightarrow c(n/10 - 4) \ge 3n$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \quad V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \;=\; cn + \big(\underbrace{3n - c \cdot (n/10 - 4)}_{\le 0?!}\big)$$

falls $c \ge \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n \to \infty]{} 30^+$

$\boxed{\text{bzw. } n \ge \frac{40c}{c-30}}$

$c \ge \frac{3n}{n/10-4}$
$\Leftrightarrow c(n/10 - 4) \ge 3n$
$\Leftrightarrow cn/10 - 4c - 3n \ge 0$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10+3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n = cn + \big(\overbrace{3n - c \cdot (n/10 - 4)}^{\le 0?!}\big)$$

falls $c \ge \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n \to \infty]{} 30^+$

$\boxed{\text{bzw. } n \ge \frac{40c}{c-30}}$

$c \ge \frac{3n}{n/10-4}$
$\Leftrightarrow c(n/10-4) \ge 3n$
$\Leftrightarrow cn/10 - 4c - 3n \ge 0$
$\Leftrightarrow n(c/10-3) \ge 4c$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$$\Rightarrow V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n = cn + \big(\underbrace{3n - c \cdot (n/10 - 4)}_{\leq 0?!}\big)$$

falls $c \geq \frac{3n}{n/10 - 4} = \frac{30}{1 - 40/n} \xrightarrow[n \to \infty]{} 30^{+}$

bzw. $n \geq \frac{40c}{c - 30}$

$c \geq \frac{3n}{n/10-4}$
$\Leftrightarrow c(n/10 - 4) \geq 3n$
$\Leftrightarrow cn/10 - 4c - 3n \geq 0$
$\Leftrightarrow n(c/10 - 3) \geq 4c$
$\Leftrightarrow n \geq \frac{4c}{c/10-3}$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \le \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \ge n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \ge n_0$ gilt: $V(n) \le cn$.

$$\Rightarrow \quad V(n) \le c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \quad = cn + \big(3n - c \cdot (n/10 - 4)\big)$$

$\le 0?!$

falls $c \ge \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n \to \infty]{} 30^+$

$\boxed{\text{bzw.} \quad n \ge \frac{40c}{c-30}}$

$c \ge \frac{3n}{n/10-4}$
$\Leftrightarrow c(n/10 - 4) \ge 3n$
$\Leftrightarrow cn/10 - 4c - 3n \ge 0$
$\Leftrightarrow n(c/10 - 3) \ge 4c$
$\Leftrightarrow n \ge \frac{4c}{c/10-3}$
$\Leftrightarrow n \ge \frac{40c}{c-30}$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

PARTITION′: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10+3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$\Rightarrow$ $V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$

$= c \cdot (9n/10 + 4) + 3n = cn + \big(3n - c \cdot (n/10 - 4)\big)$

falls $c \geq \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n \to \infty]{} 30^{+}$ bzw. $n \geq \frac{40c}{c-30}$

$\Rightarrow$ für jedes $\varepsilon > 0$ gilt: $V(n) \leq \underbrace{(30+\varepsilon)}_{c} \cdot n$

$c \geq \frac{3n}{n/10-4}$
$\Leftrightarrow c(n/10-4) \geq 3n$
$\Leftrightarrow cn/10 - 4c - 3n \geq 0$
$\Leftrightarrow n(c/10-3) \geq 4c$
$\Leftrightarrow n \geq \frac{4c}{c/10-3}$
$\Leftrightarrow n \geq \frac{40c}{c-30}$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

PARTITION$'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10+3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$$\Rightarrow V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n = cn + (3n - c \cdot (n/10 - 4))$$

falls $c \geq \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n \to \infty]{} 30^+$ bzw. $n \geq \frac{40c}{c-30}$

$$\Rightarrow \text{für jedes } \varepsilon > 0 \text{ und } n \geq \frac{1200}{\varepsilon} + 40 \text{ gilt: } V(n) \leq \underbrace{(30+\varepsilon)}_{c} \cdot n$$

$c \geq \frac{3n}{n/10-4}$
$\Leftrightarrow c(n/10-4) \geq 3n$
$\Leftrightarrow cn/10 - 4c - 3n \geq 0$
$\Leftrightarrow n(c/10-3) \geq 4c$
$\Leftrightarrow n \geq \frac{4c}{c/10-3}$
$\Leftrightarrow n \geq \frac{40c}{c-30}$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

$\textsc{Partition}'$: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\text{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10 + 3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$$\Rightarrow \; V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \; = cn + \big(3n - c \cdot (n/10 - 4)\big)$$

falls $c \geq \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n\to\infty]{} 30^+$ $\quad\boxed{\text{bzw. } n \geq \frac{40c}{c-30}}$

$$\Rightarrow \text{ für jedes } \varepsilon > 0 \text{ und } n \geq \frac{1200}{\varepsilon} + 40 \text{ gilt: } V(n) \leq (30 + \varepsilon) \cdot n$$

$c \geq \frac{3n}{n/10-4}$
$\Leftrightarrow c(n/10 - 4) \geq 3n$
$\Leftrightarrow cn/10 - 4c - 3n \geq 0$
$\Leftrightarrow n(c/10 - 3) \geq 4c$
$\Leftrightarrow n \geq \frac{4c}{c/10-3}$
$\Leftrightarrow n \geq \frac{40c}{c-30}$

# Laufzeitanalyse

**Beobachtung.** Es genügt wieder, Vergleiche zu zählen!

PARTITION′: $1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ Vergleiche

**Ansatz.**

$$V(n) \leq \begin{cases} \overbrace{V(\lceil n/5 \rceil)}^{\text{Schritt 3}} + \underbrace{V(7n/10+3)}_{\text{Schritt 5}} + 3n & \text{falls } n \geq n_0, \\ \mathcal{O}(1) & \text{sonst.} \end{cases}$$

**Behauptung.** Es gibt $c, n_0 > 0$, so dass für alle $n \geq n_0$ gilt: $V(n) \leq cn$.

$$\Rightarrow \quad V(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 3) + 3n$$

$$= c \cdot (9n/10 + 4) + 3n \quad = cn + \big(3n - c \cdot (n/10 - 4)\big)$$

falls $c \geq \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n \to \infty]{} 30^{+}$ $\quad\boxed{\text{bzw. } n \geq \frac{40c}{c-30}}$

$$\Rightarrow \quad \text{für jedes } \varepsilon > 0 \text{ und } n \geq \frac{1200}{\varepsilon} + 40 \text{ gilt: } V(n) \leq (30 + \varepsilon) \cdot n$$

$c \geq \frac{3n}{n/10-4}$
$\Leftrightarrow c(n/10 - 4) \geq 3n$
$\Leftrightarrow cn/10 - 4c - 3n \geq 0$
$\Leftrightarrow n(c/10 - 3) \geq 4c$
$\Leftrightarrow n \geq \frac{4c}{c/10-3}$
$\Leftrightarrow n \geq \frac{40c}{c-30}$

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq 1200/\varepsilon + 40$ Zahlen die $i$-kleinste Zahl mit höchstens $(30 + \varepsilon)n$ Vergleichen finden kann.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq 1200/\varepsilon + 40$ Zahlen die $i$-kleinste Zahl mit höchstens $(30 + \varepsilon)n$ Vergleichen finden kann.

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit Wahrscheinlichkeit $1 - \mathcal{O}(1/\sqrt[4]{n})$ mit $\frac{3}{2}n + o(n)$ Vergleichen

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq 1200/\varepsilon + 40$ Zahlen die $i$-kleinste Zahl mit höchstens $(30 + \varepsilon)n$ Vergleichen finden kann.

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit Wahrscheinlichkeit $1 - \mathcal{O}(1/\sqrt[4]{n})$ mit $\frac{3}{2}n + o(n)$ Vergleichen
- Die besten deterministischen Auswahl-Algorithmen (*sehr kompliziert!*) benötigen $3n$ Vergleiche im schlechtesten Fall.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq 1200/\varepsilon + 40$ Zahlen die $i$-kleinste Zahl mit höchstens $(30 + \varepsilon)n$ Vergleichen finden kann.

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit Wahrscheinlichkeit $1 - \mathcal{O}(1/\sqrt[4]{n})$ mit $\frac{3}{2}n + o(n)$ Vergleichen
- Die besten deterministischen Auswahl-Algorithmen (*sehr kompliziert!*) benötigen $3n$ Vergleiche im schlechtesten Fall.
- **Jeder** deterministische Auswahl-Algorithmus benötigt im schlechtesten Fall mindestens $2n$ Vergleiche.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq 1200/\varepsilon + 40$ Zahlen die $i$-kleinste Zahl mit höchstens $(30 + \varepsilon)n$ Vergleichen finden kann.

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit Wahrscheinlichkeit $1 - \mathcal{O}(1/\sqrt[4]{n})$ mit $\frac{3}{2}n + o(n)$ Vergleichen


- Die besten deterministischen Auswahl-Algorithmen (*sehr kompliziert!*) benötigen $3n$ Vergleiche im schlechtesten Fall.
- **Jeder** deterministische Auswahl-Algorithmus benötigt im schlechtesten Fall mindestens $2n$ Vergleiche.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq 1200/\varepsilon + 40$ Zahlen die $i$-kleinste Zahl mit höchstens $(30 + \varepsilon)n$ Vergleichen finden kann.

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit Wahrscheinlichkeit $1 - \mathcal{O}(1/\sqrt[4]{n})$ mit $\frac{3}{2}n + o(n)$ Vergleichen


- Die besten deterministischen Auswahl-Algorithmen (*sehr kompliziert!*) benötigen $3n$ Vergleiche im schlechtesten Fall.
- **Jeder** deterministische Auswahl-Algorithmus benötigt im schlechtesten Fall mindestens $2n$ Vergleiche.

**Satz.** Durch Suche des Medians in $\mathcal{O}(n)$ Zeit kann QuickSort im *Worst-Case* in Zeit sortieren.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq 1200/\varepsilon + 40$ Zahlen die $i$-kleinste Zahl mit höchstens $(30 + \varepsilon)n$ Vergleichen finden kann.

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit Wahrscheinlichkeit $1 - \mathcal{O}(1/\sqrt[4]{n})$ mit $\frac{3}{2}n + o(n)$ Vergleichen


- Die besten deterministischen Auswahl-Algorithmen (*sehr kompliziert!*) benötigen $3n$ Vergleiche im schlechtesten Fall.
- **Jeder** deterministische Auswahl-Algorithmus benötigt im schlechtesten Fall mindestens $2n$ Vergleiche.

**Satz.** Durch Suche des Medians in $\mathcal{O}(n)$ Zeit kann QuickSort im *Worst-Case* in $\mathcal{O}(n \log n)$ Zeit sortieren.

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

**Genauer:** Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq 1200/\varepsilon + 40$ Zahlen die $i$-kleinste Zahl mit höchstens $(30 + \varepsilon)n$ Vergleichen finden kann.

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit Wahrscheinlichkeit $1 - \mathcal{O}(1/\sqrt[4]{n})$ mit $\frac{3}{2}n + o(n)$ Vergleichen


- Die besten deterministischen Auswahl-Algorithmen (*sehr kompliziert!*) benötigen $3n$ Vergleiche im schlechtesten Fall.
- **Jeder** deterministische Auswahl-Algorithmus benötigt im schlechtesten Fall mindestens $2n$ Vergleiche.

**Satz.** Durch Suche des Medians in $\mathcal{O}(n)$ Zeit kann QuickSort im *Worst-Case* in $\mathcal{O}(n \log n)$ Zeit sortieren. (Aber die Konstanten in der Laufzeit sind hoch.)

# Vergleich Sortieralgorithmen

| | **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 | $\Theta(n \log n)$ | $\Theta(n \log n)$ | $\Theta(n^2)$ | ✓ | ✗ |
| CountingSort | $\mathcal{O}(n+k)$ | $\mathcal{O}(n+k)$ | $\mathcal{O}(n+k)$ | ✗ | ✓ |
| RadixSort | $\mathcal{O}(s \cdot (n+b))$ | $\mathcal{O}(s \cdot (n+b))$ | $\mathcal{O}(s \cdot (n+b))$ | ✗ | ✓ |
| BucketSort | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ wenn Eingabe zufällig gleichverteilt | $\mathcal{O}(n^2)$ | ✗ | ✓ |

# Vergleich Sortieralgorithmen

| | **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 | $\Theta(n \log n)$ | $\Theta(n \log n)$ | $\Theta(n \log n)$ | ✓ | ✗ |
| CountingSort | $\mathcal{O}(n + k)$ | $\mathcal{O}(n + k)$ | $\mathcal{O}(n + k)$ | ✗ | ✓ |
| RadixSort | $\mathcal{O}(s \cdot (n + b))$ | $\mathcal{O}(s \cdot (n + b))$ | $\mathcal{O}(s \cdot (n + b))$ | ✗ | ✓ |
| BucketSort | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ wenn Eingabe zufällig gleichverteilt | $\mathcal{O}(n^2)$ | ✗ | ✓ |