

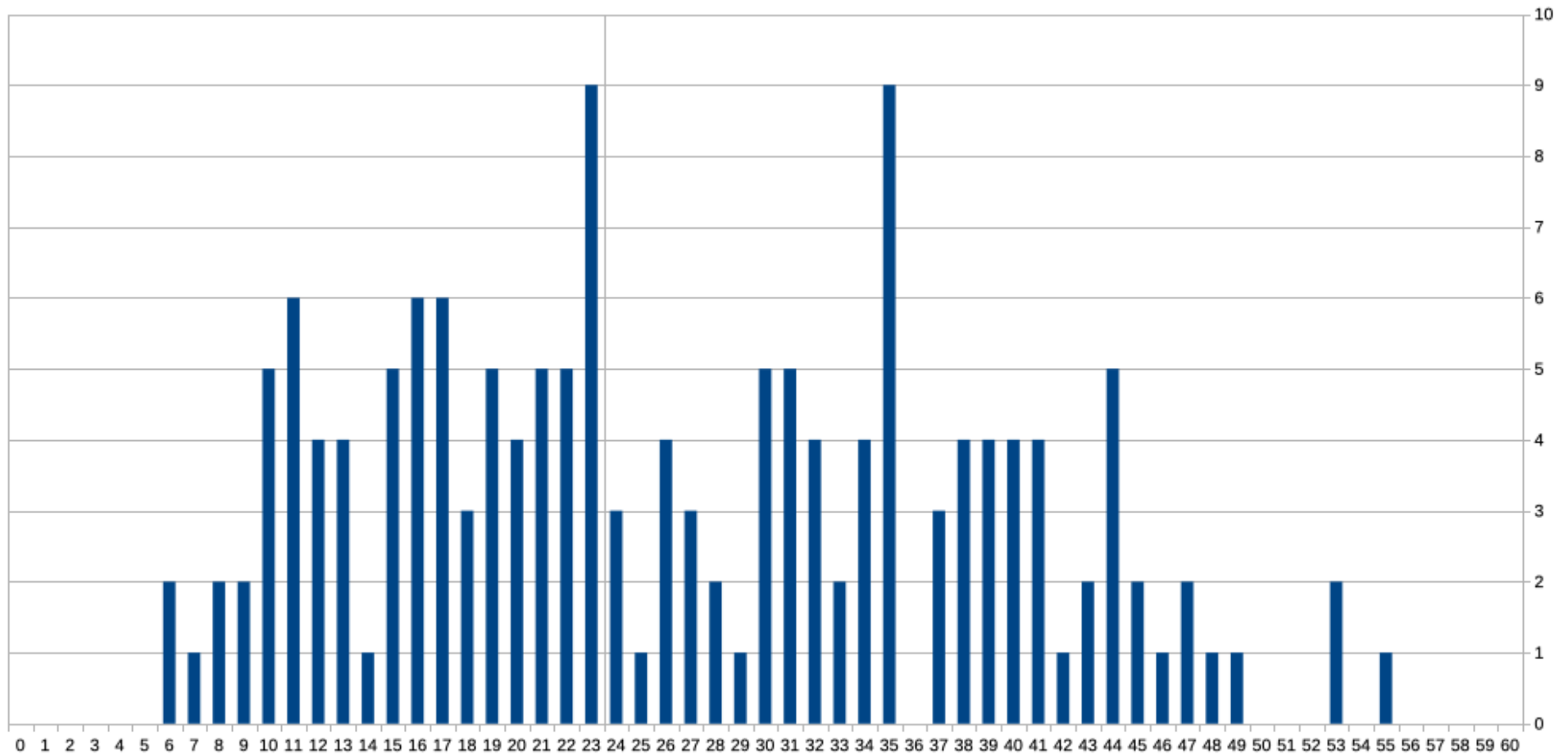
# Algorithmen und Datenstrukturen

Wintersemester 2023/24

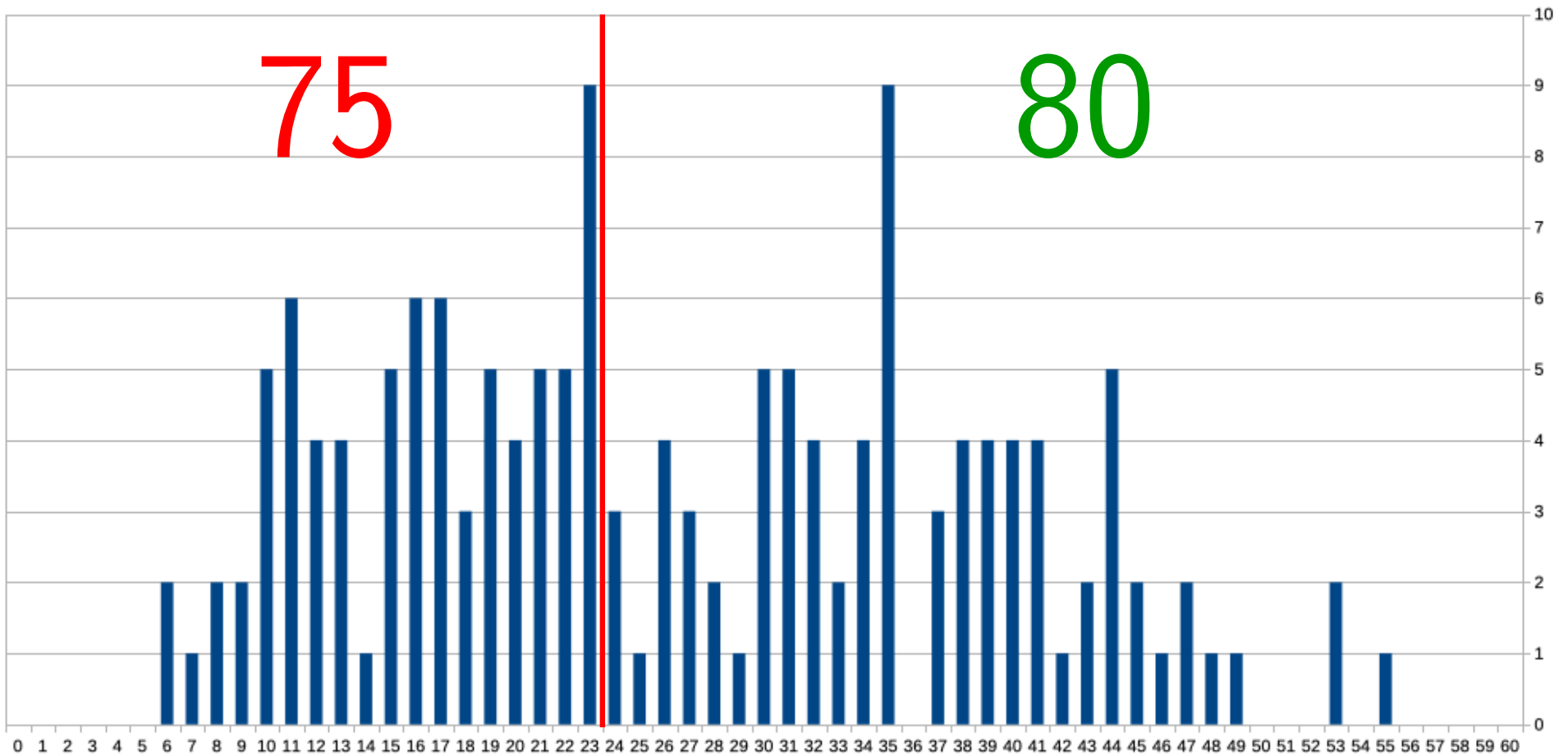
10. Vorlesung

## Das Auswahlproblem

# Ergebnisse 1. Zwischentest



# Ergebnisse 1. Zwischentest

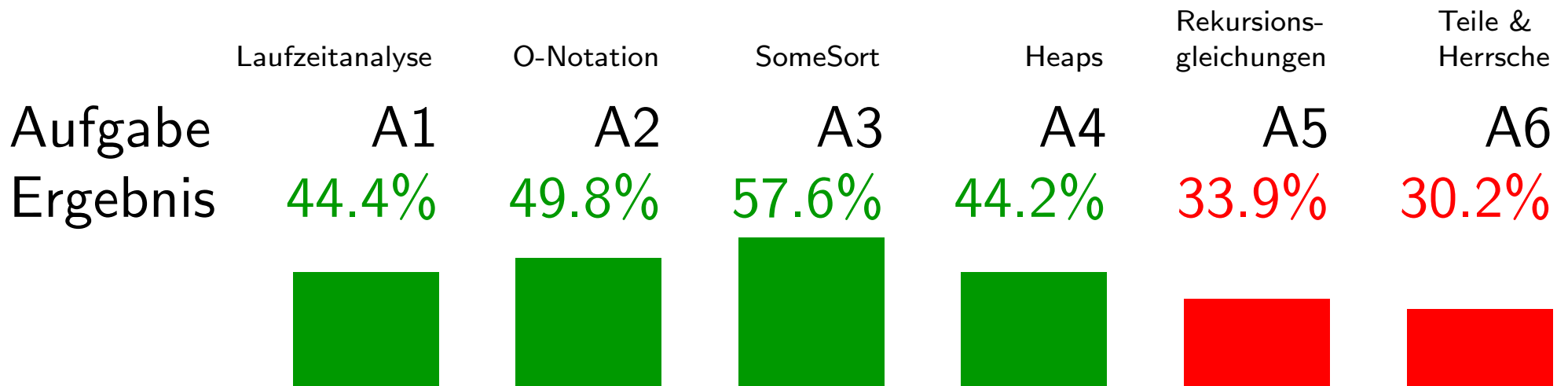


$n = 155$

arithmetisches Mittel = 26,7

Median = 24,5

# Ergebnisse nach Aufgabe



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

# Analyse von Messreihen

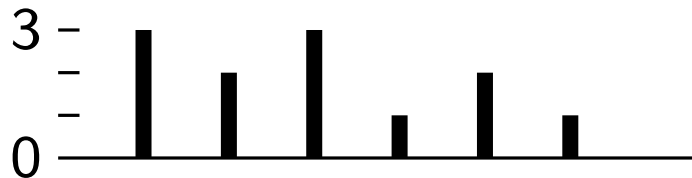
**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

**Beispiel:**

# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

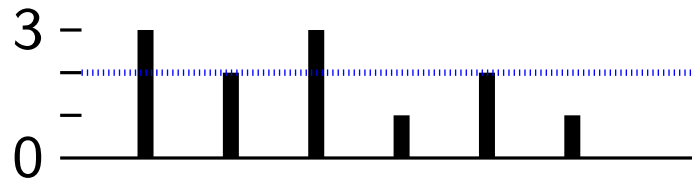
**Beispiel:**



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

**Beispiel:**





# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

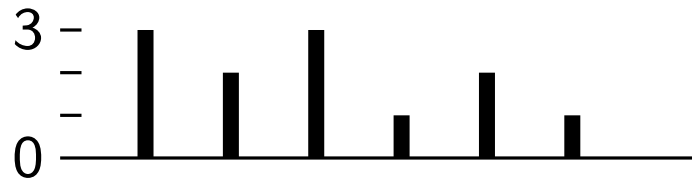
**Beispiel:**



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

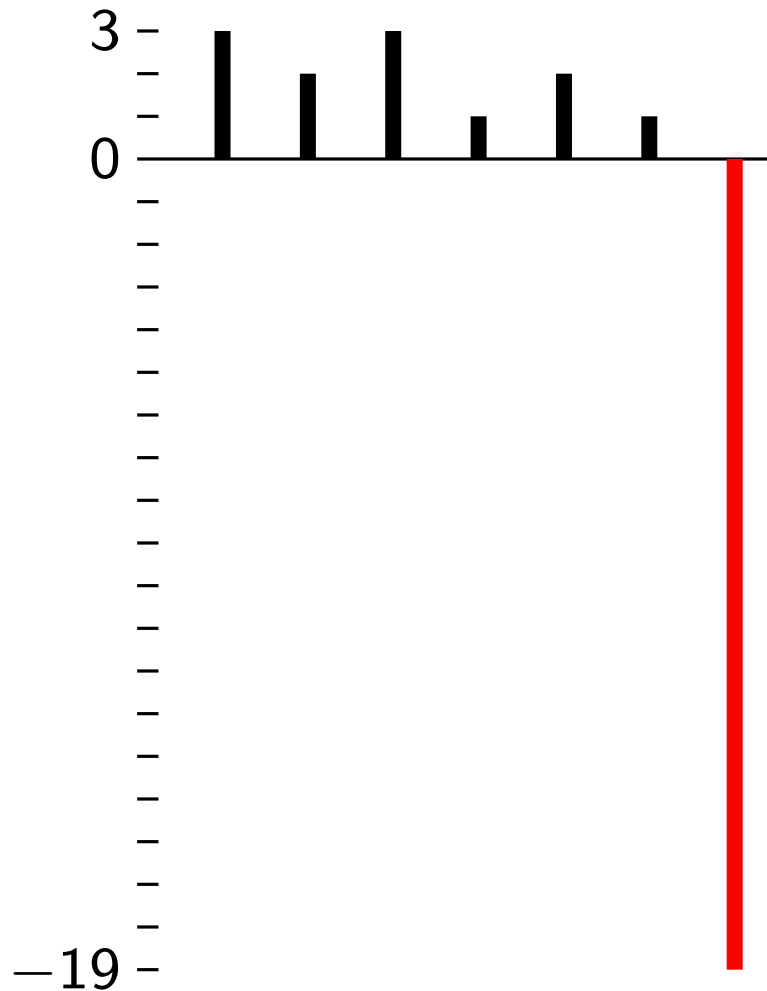
**Beispiel:**



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

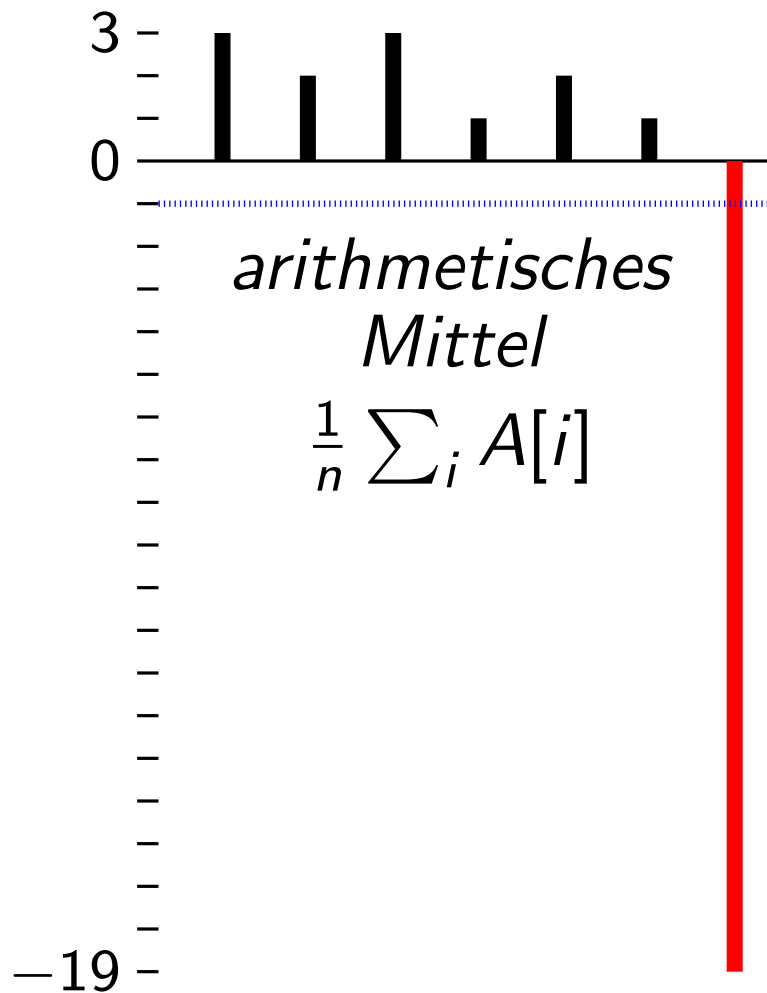
**Beispiel:**



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

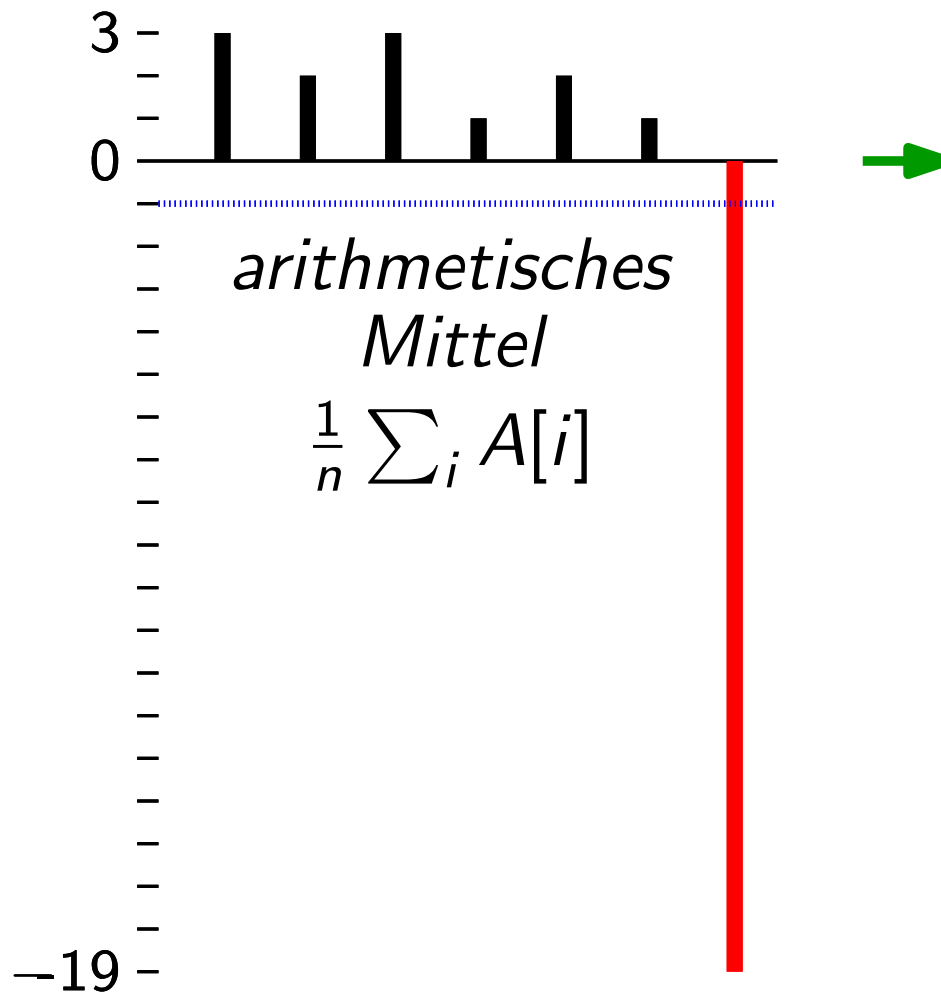
**Beispiel:**



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

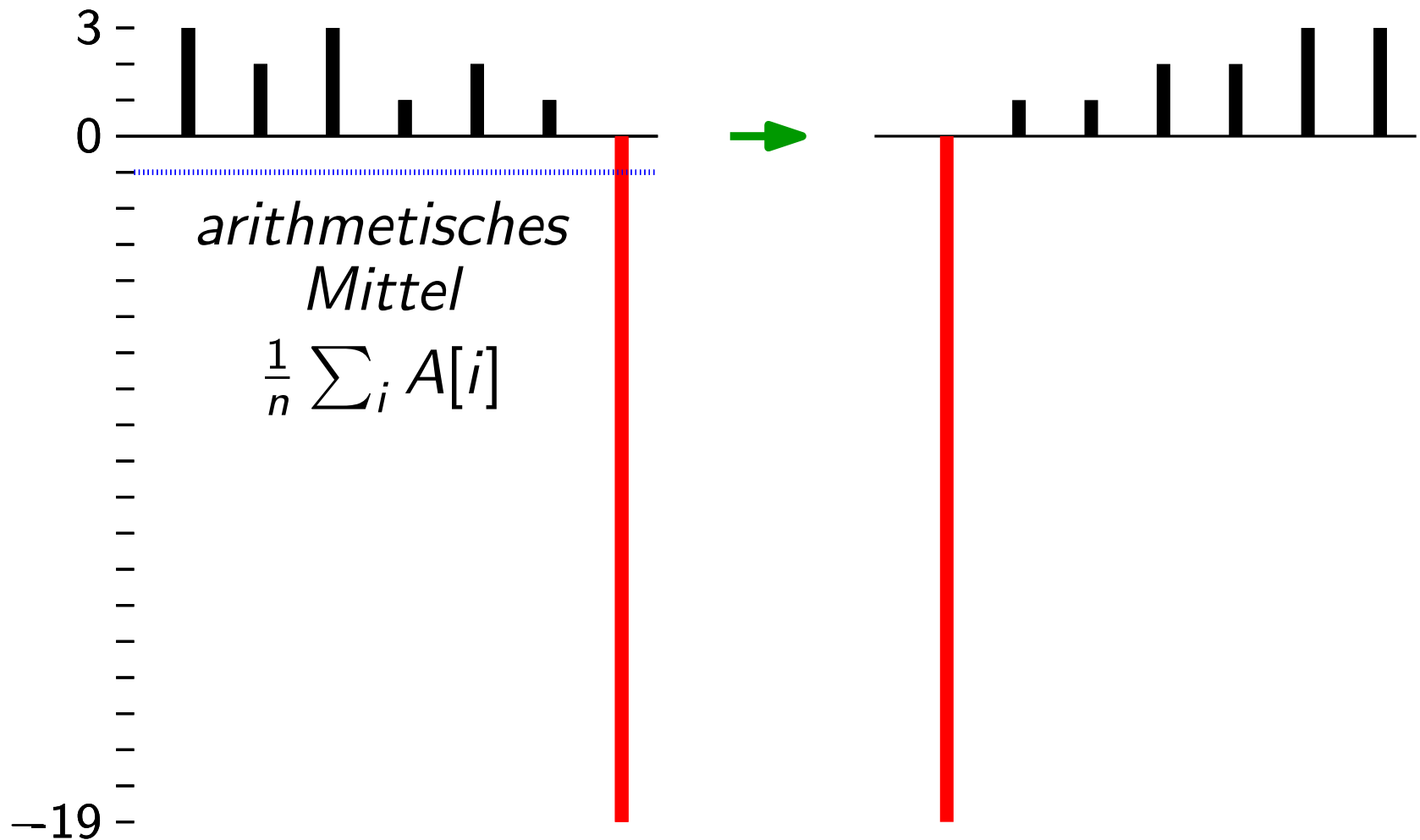
**Beispiel:**



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

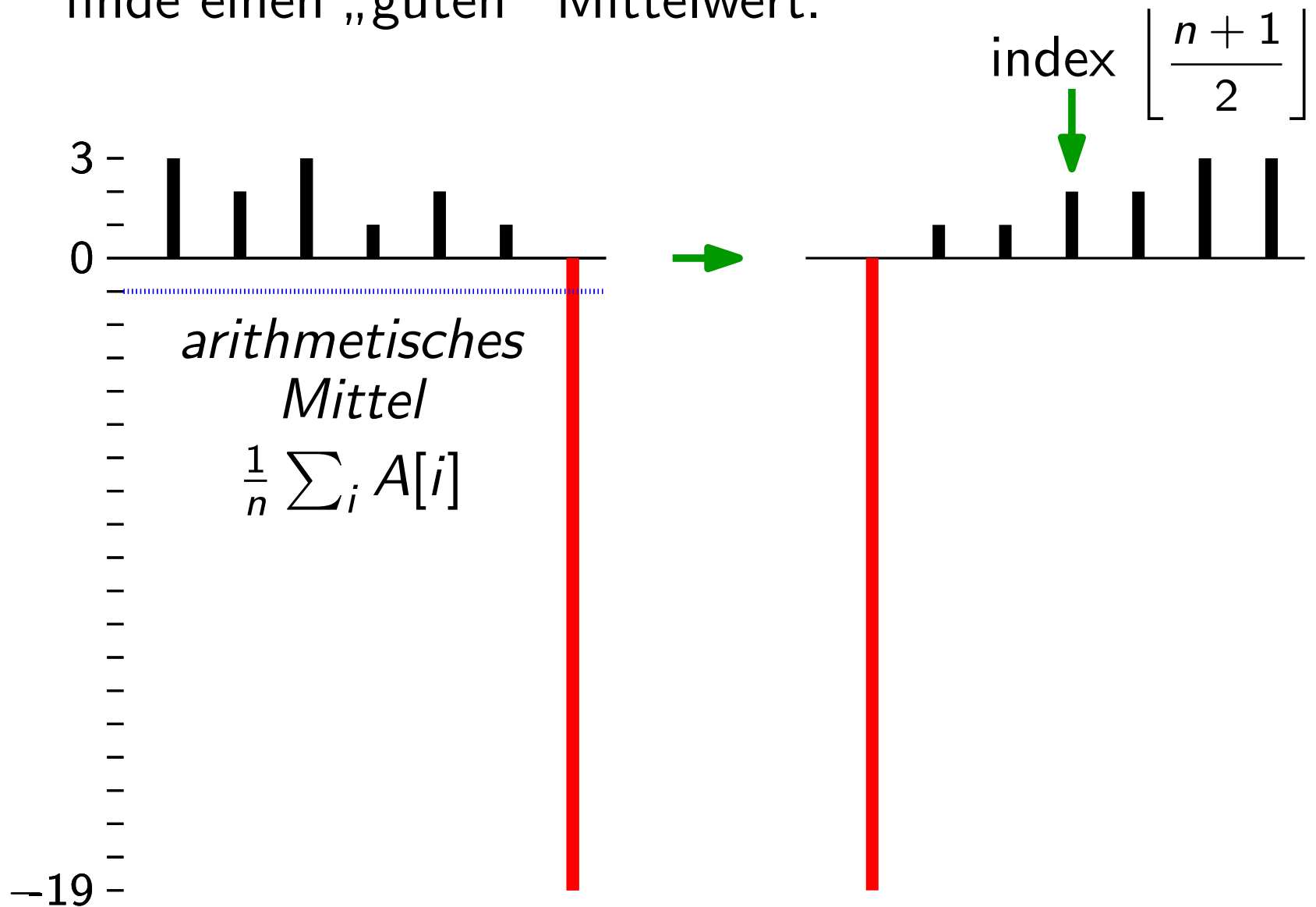
**Beispiel:**



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

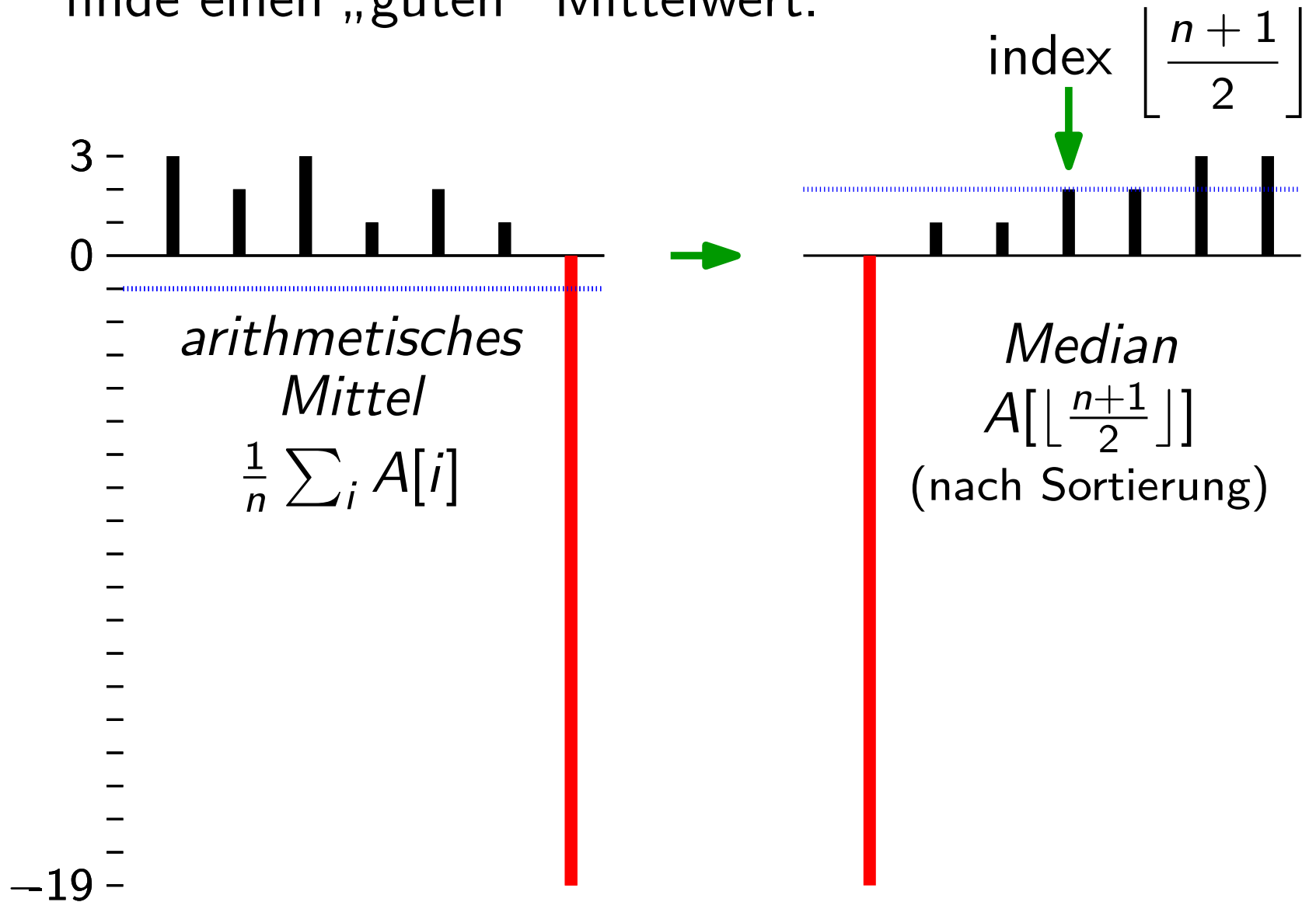
**Beispiel:**



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

**Beispiel:**

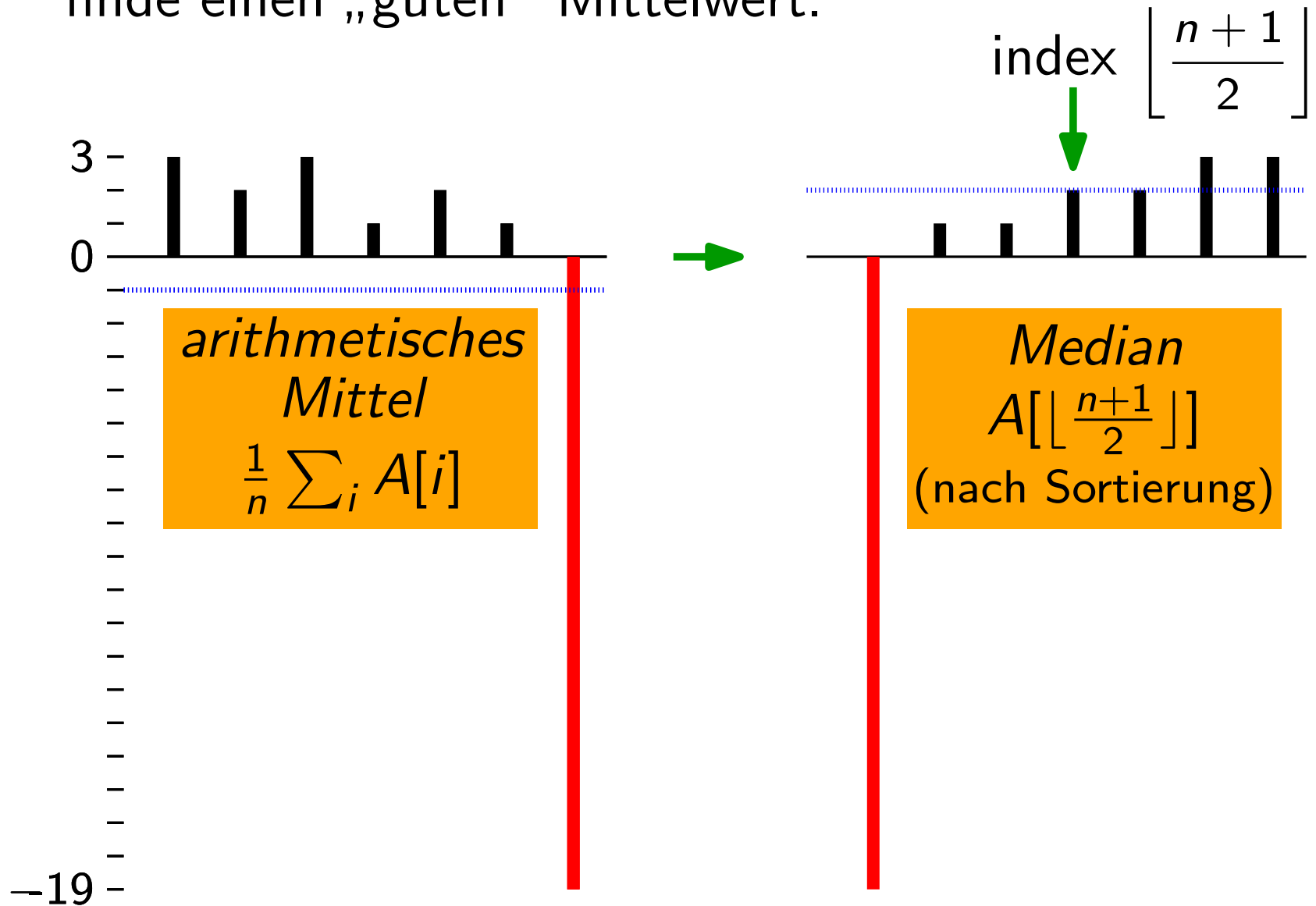




# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

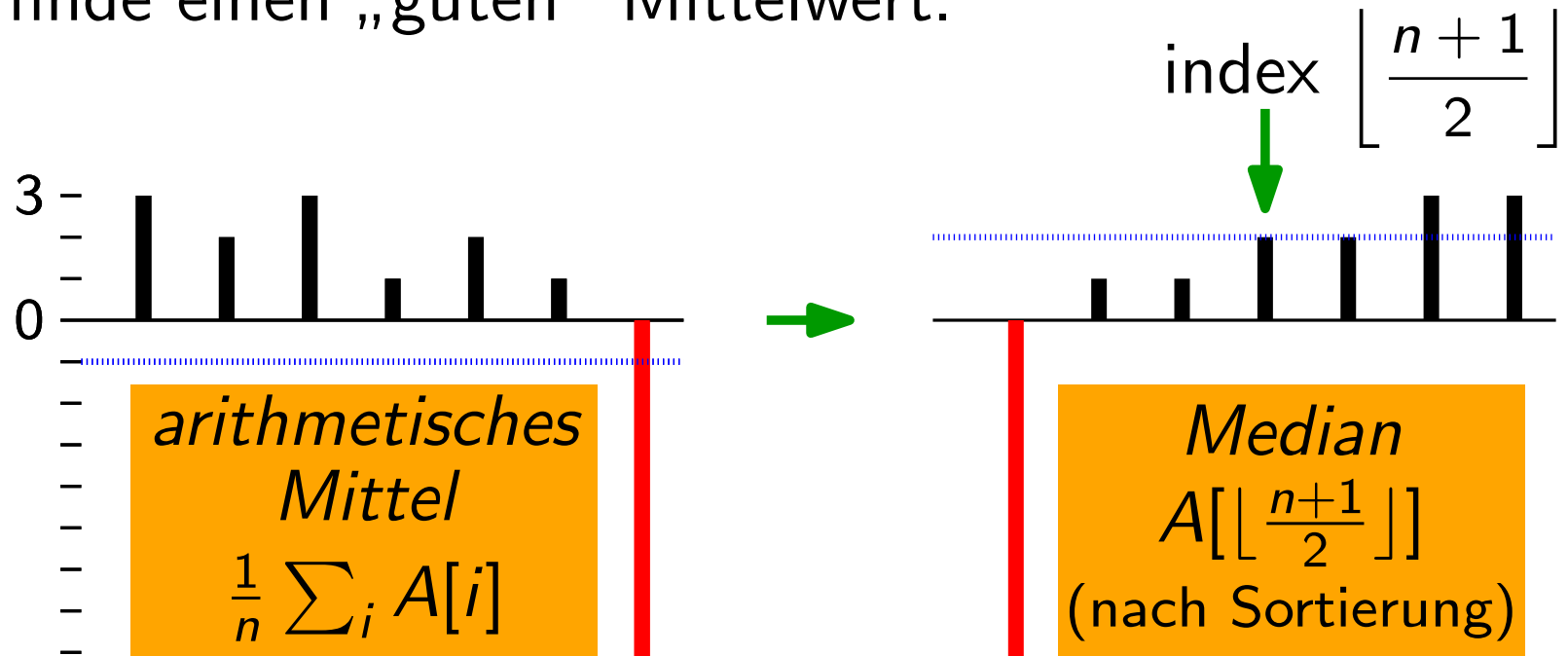
**Beispiel:**



# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

**Beispiel:**

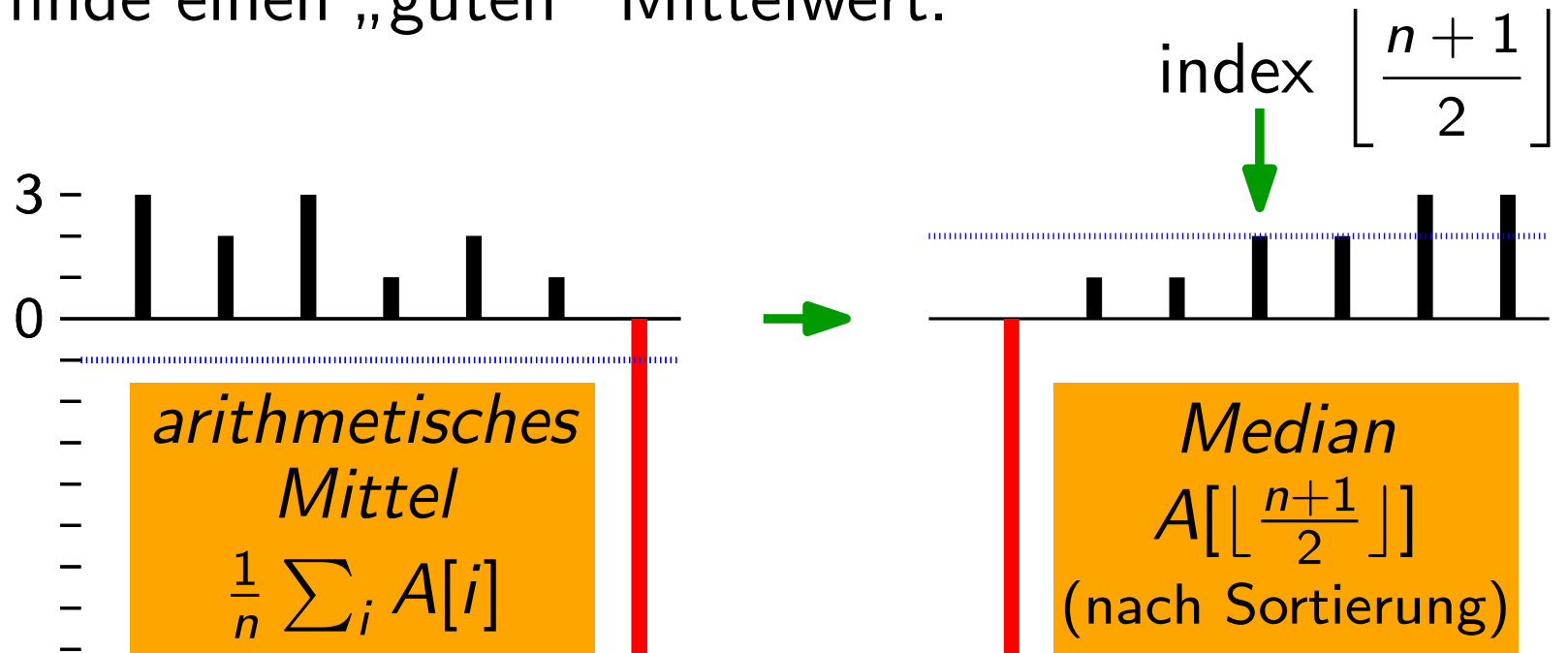


**Beob.:** Der Median ist stabiler gegen Ausreißer als das arithmetische Mittel.

# Analyse von Messreihen

**Problem:** Gegeben eine Reihe von  $n$  Messwerten  $A[1..n]$ , finde einen „guten“ Mittelwert.

**Beispiel:**



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

-19-

**Berechnung?**

# Das Auswahlproblem

**Aufgabe:** Gegeben ein Feld  $A[1..n]$ ,  
finde das  $i$ -kleinste Element von  $A$ .

# Das Auswahlproblem

**Aufgabe:** Gegeben ein Feld  $A[1..n]$ ,  
finde das  $i$ -kleinste Element von  $A$ .

**Lösung:**

# Das Auswahlproblem

**Aufgabe:** Gegeben ein Feld  $A[1..n]$ ,  
finde das  $i$ -kleinste Element von  $A$ .

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

# Das Auswahlproblem

**Aufgabe:** Gegeben ein Feld  $A[1..n]$ ,  
finde das  $i$ -kleinste Element von  $A$ .

**Lösung:** Sortiere und gib  $A[i]$  zurück!  
Worst-Case-Laufzeit:

# Das Auswahlproblem

**Aufgabe:** Gegeben ein Feld  $A[1..n]$ ,  
finde das  $i$ -kleinste Element von  $A$ .

**Lösung:** Sortiere und gib  $A[i]$  zurück!  
Worst-Case-Laufzeit:  $\Theta(n \log n)$



# Das Auswahlproblem

**Aufgabe:** Gegeben ein Feld  $A[1..n]$ ,  
finde das  $i$ -kleinste Element von  $A$ .

**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

**Aufgabe:** Gegeben ein Feld  $A[1..n]$ ,  
finde das  $i$ -kleinste Element von  $A$ .

**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

```
Minimum(int[] A)
```

```
  min = A[1]
```

```
  for  $i = 2$  to  $A.length$  do
```

```
     $\lfloor$  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 =  $n - 1$

# Spezialfälle

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

$i = 1$ : Minimum

$i = n$ : Maximum

} 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 =  $n - 1$



# Spezialfälle

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

$i = 1$ : Minimum

$i = n$ : Maximum

} 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

} Laufzeit  $\Theta(n)$

```
Minimum(int[] A)
```

```
  min = A[1]
```

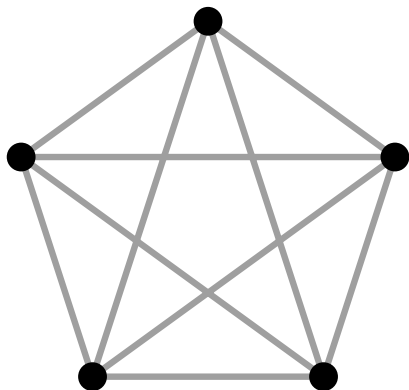
```
  for i = 2 to A.length do
```

```
    [ 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

$i = 1$ : Minimum

$i = n$ : Maximum

} Laufzeit  $\Theta(n)$

```
Minimum(int[] A)
```

```
  min = A[1]
```

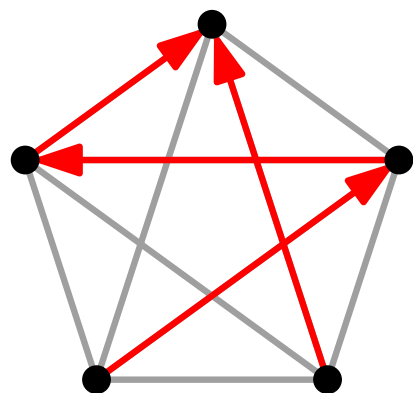
```
  for i = 2 to A.length do
```

```
    [ 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

} Laufzeit  $\Theta(n)$

```
Minimum(int[] A)
```

```
  min = A[1]
```

```
  for i = 2 to A.length do
```

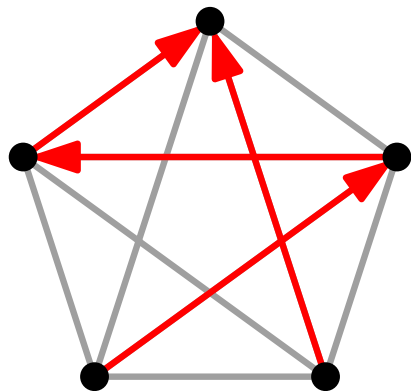
```
    [ if min > A[i] then min = A[i]
```

```
  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

$i = \lfloor \frac{n+1}{2} \rfloor$ : Median      ← *Geht das auch in linearer Zeit??*

$i = 1$ : Minimum  
 $i = n$ : Maximum      } Laufzeit  $\Theta(n)$

```
Minimum(int[] A)
```

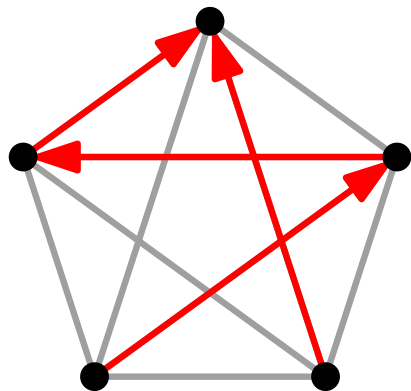
```
  min = A[1]
```

```
  for i = 2 to A.length do
```

```
    [ if min > A[i] then min = A[i] ]      Anzahl Vergleiche = n - 1
```

```
  return min
```

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

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq$



# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) =$

# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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

# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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



min

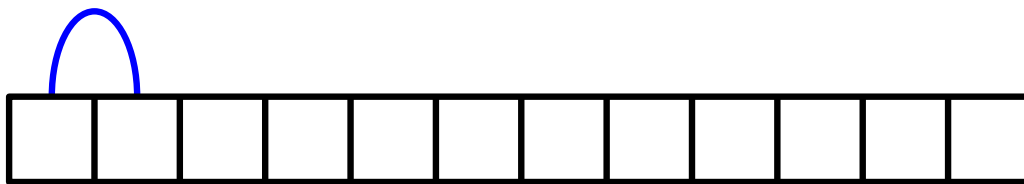
max

# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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



min

max

# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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



min

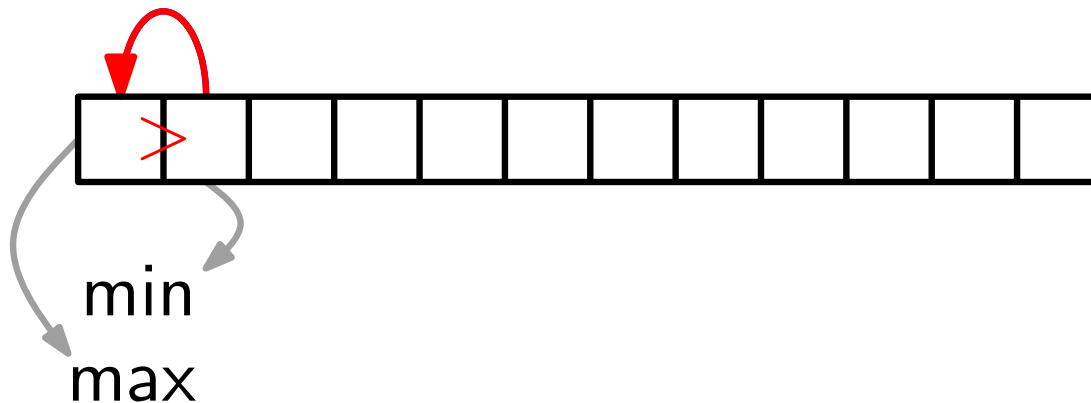
max

# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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



# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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



min

max

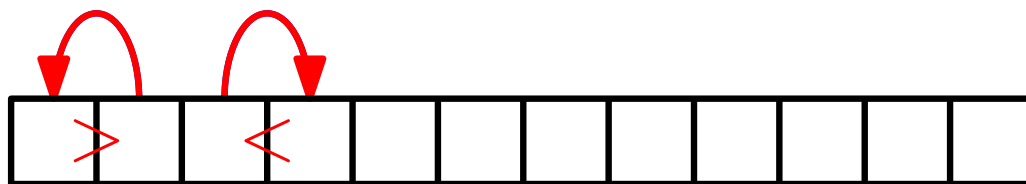


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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



min

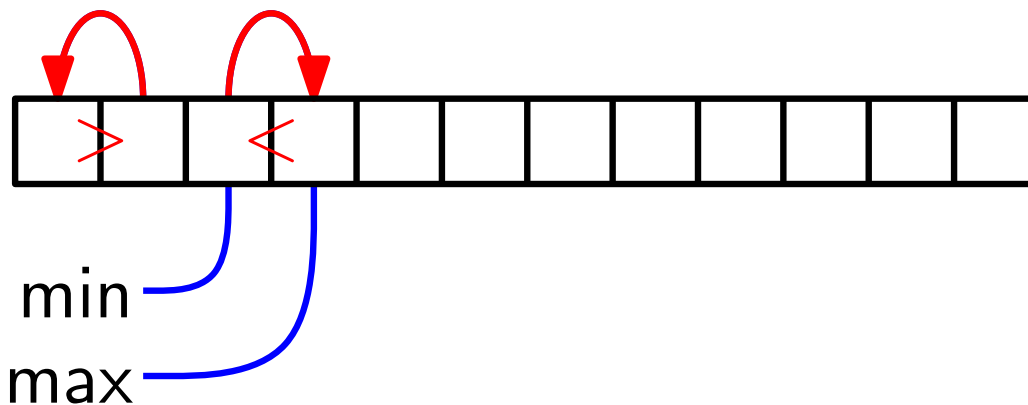
max

# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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

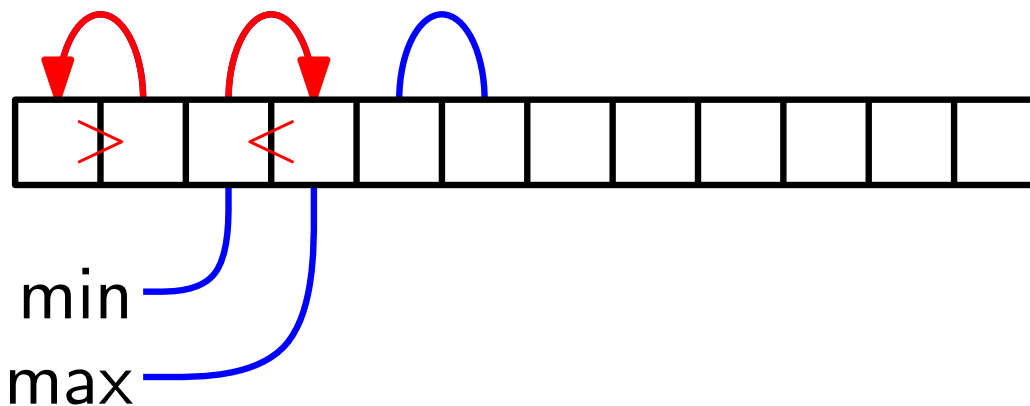


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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

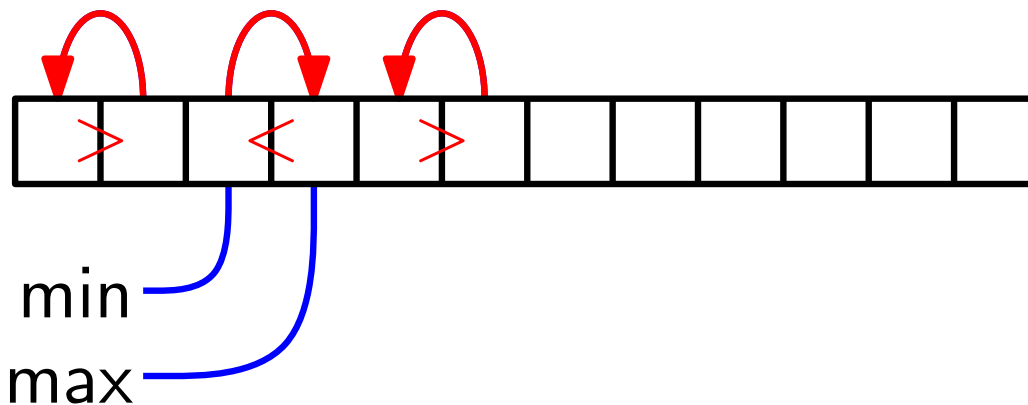


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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

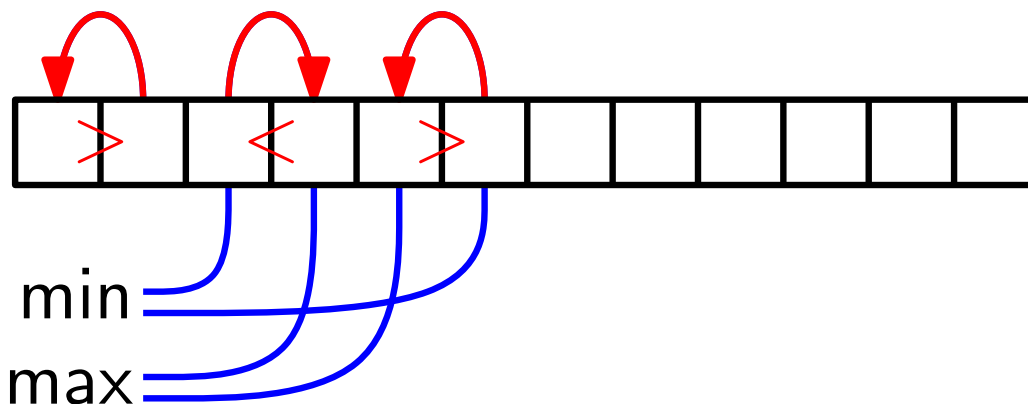


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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

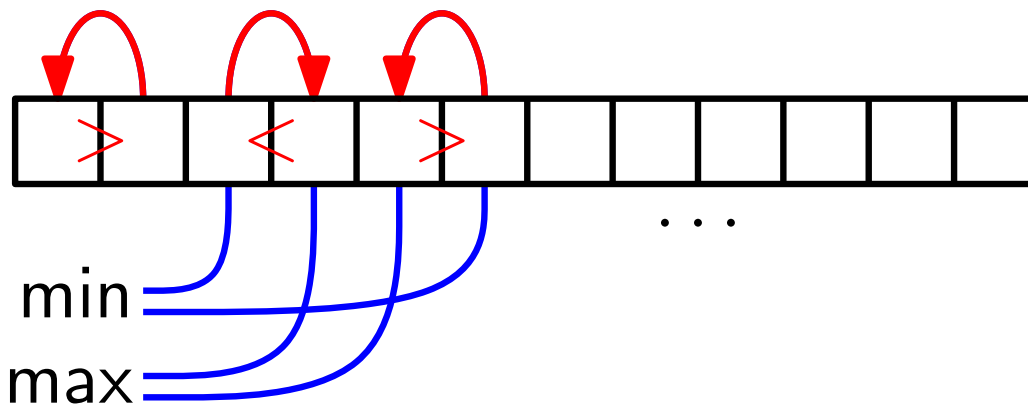


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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

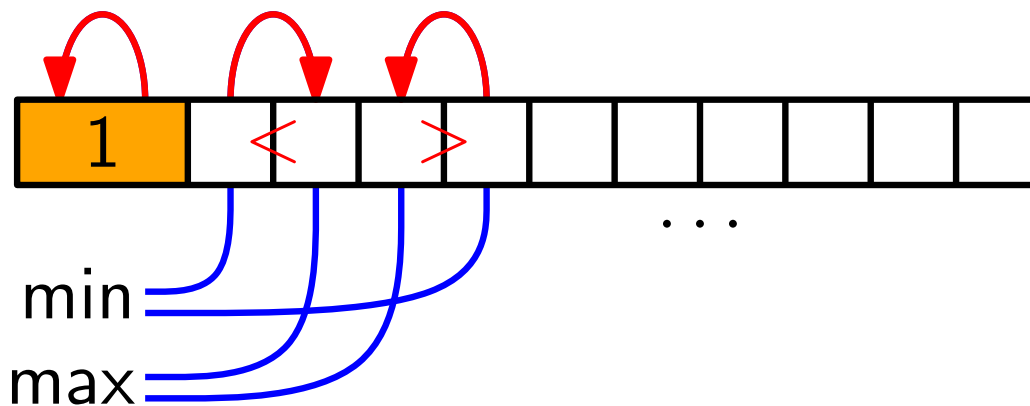


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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

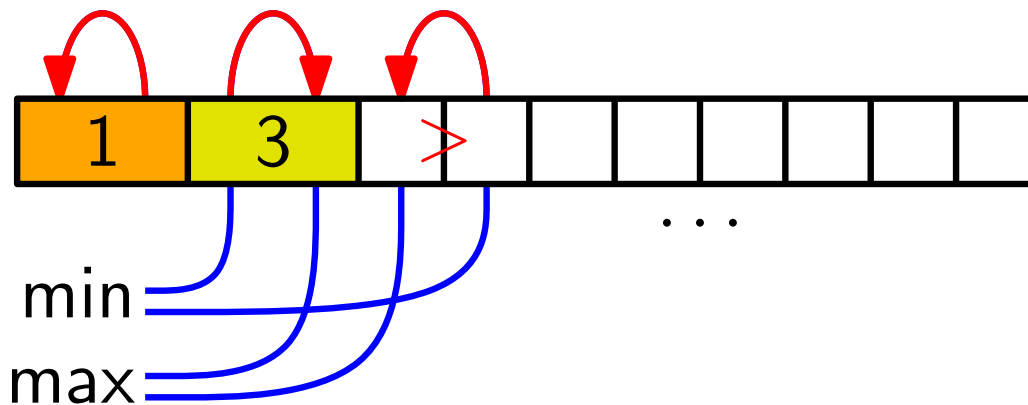


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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



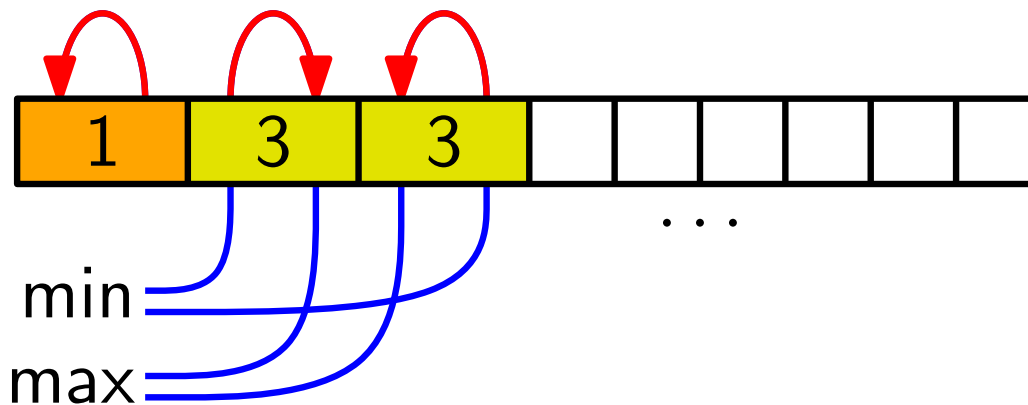


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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

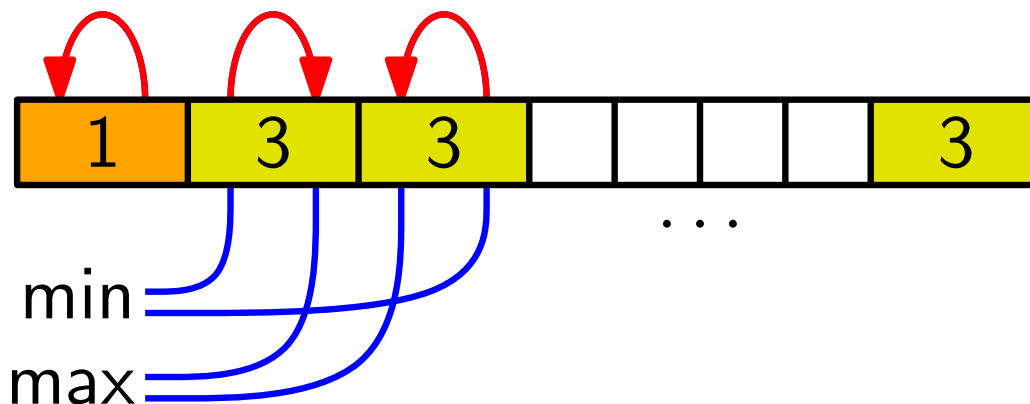


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

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

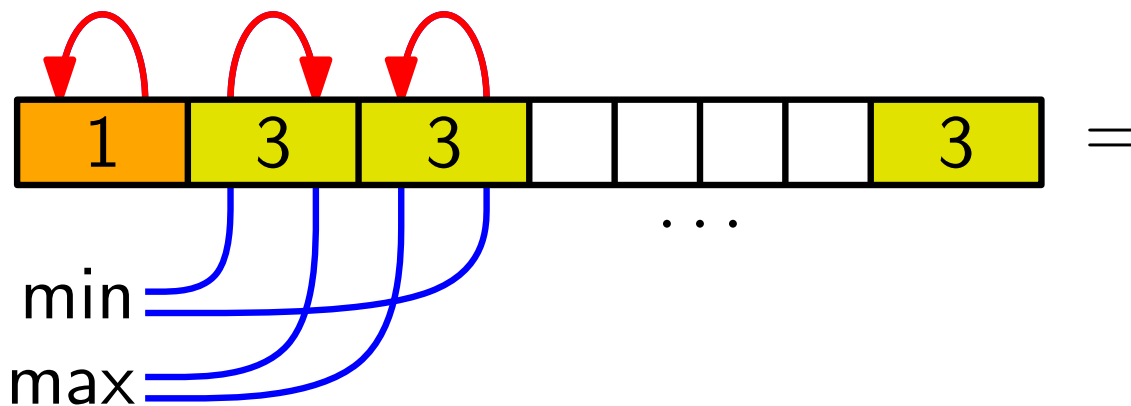


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

**Frage:** Geht es auch mit weniger Vergleichen? ( $n$  gerade)

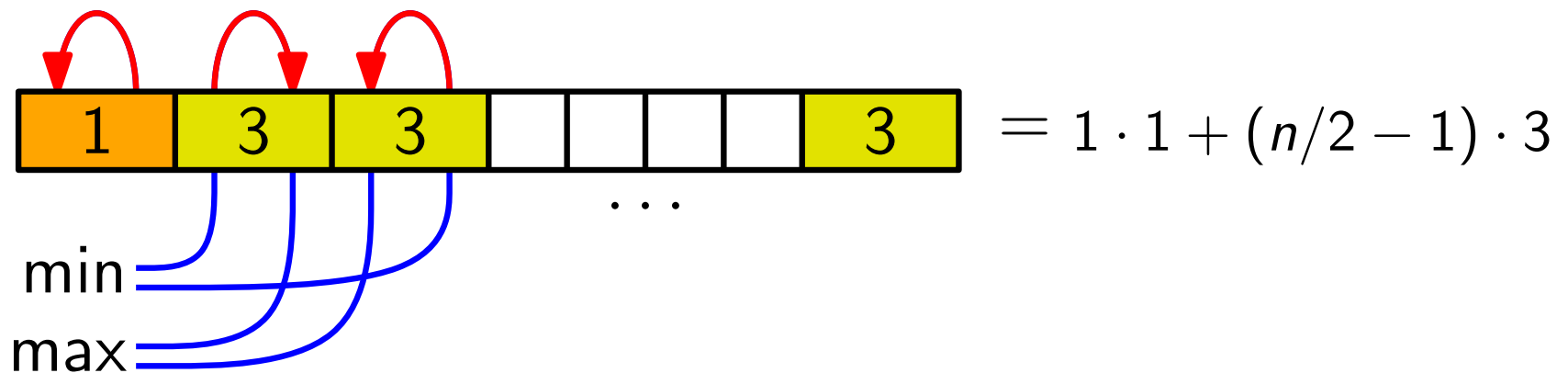


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

**Frage:** Geht es auch mit weniger Vergleichen? ( $n$  gerade)

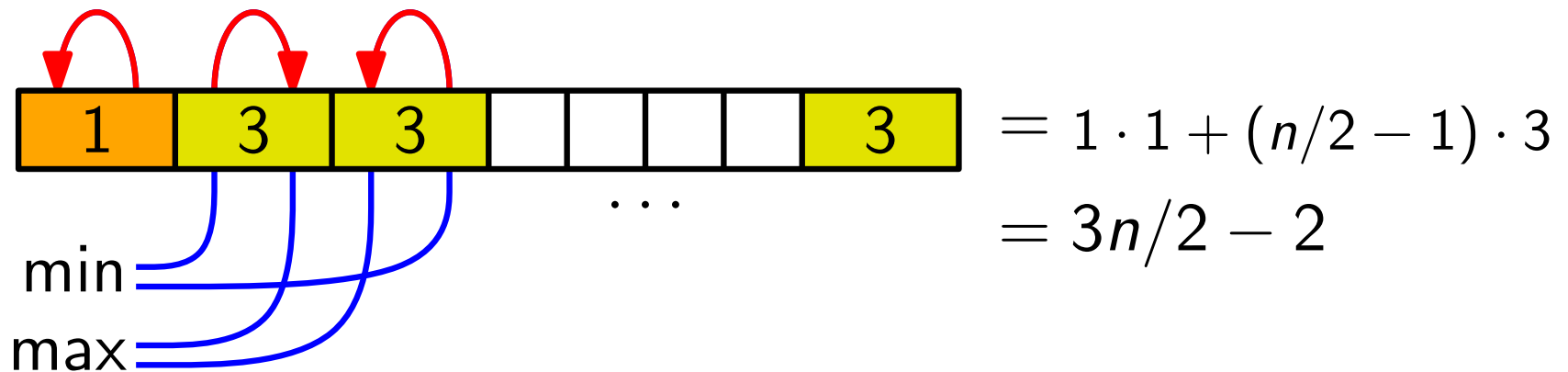


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

**Frage:** Geht es auch mit weniger Vergleichen? ( $n$  gerade)

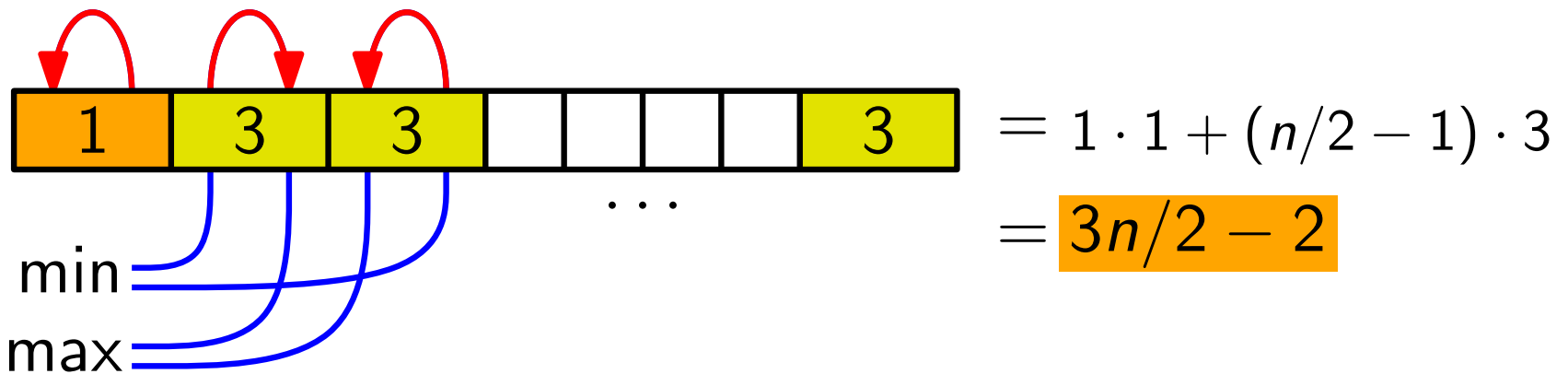


# Zuerst zur zweiten Frage

**Def.** Sei  $V_{\min\max}(n)$  die Anz. der Vgl., die man braucht, um Minimum *und* Maximum von  $n$  Zahlen zu bestimmen.

**Klar:**  $V_{\min\max}(n) \leq 2 \cdot V_{\min}(n) = 2(n - 1)$

**Frage:** Geht es auch mit weniger Vergleichen? ( $n$  gerade)



Ist das *optimal*?

# Auswahl per Teile & Herrsche

Zur Erinnerung...

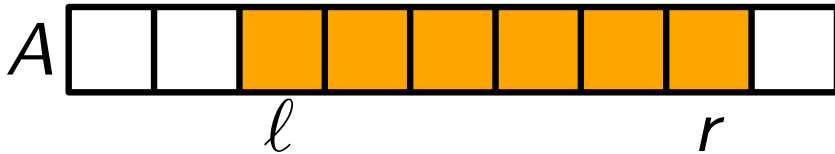
```
QuickSort(int[] A, int  $\ell, r$ )
```

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

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

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

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



# Auswahl per Teile & Herrsche

Zur Erinnerung...

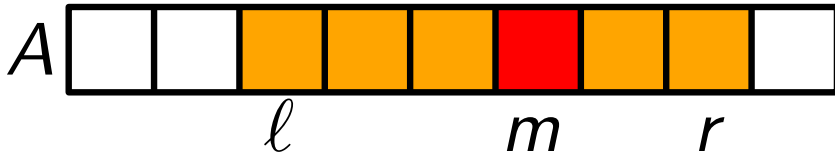
```
QuickSort(int[] A, int  $\ell, r$ )
```

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

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

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

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





# Auswahl per Teile & Herrsche

Zur Erinnerung...

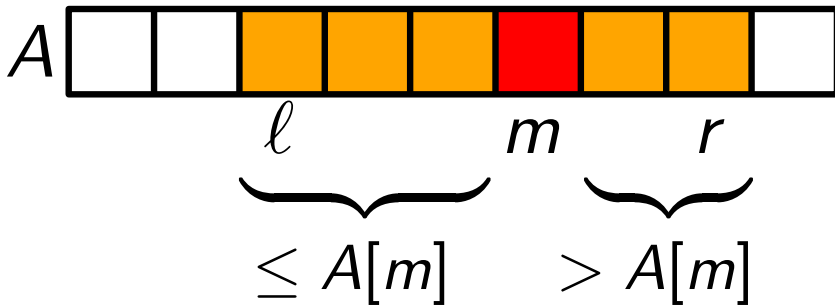
```
QuickSort(int[] A, int  $\ell, r$ )
```

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

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

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

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



# Auswahl per Teile & Herrsche

Zur Erinnerung...

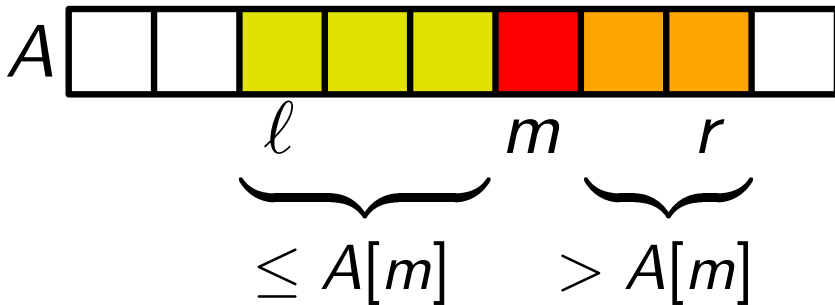
```
QuickSort(int[] A, int  $\ell, r$ )
```

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

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

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

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



# Auswahl per Teile & Herrsche

Zur Erinnerung...

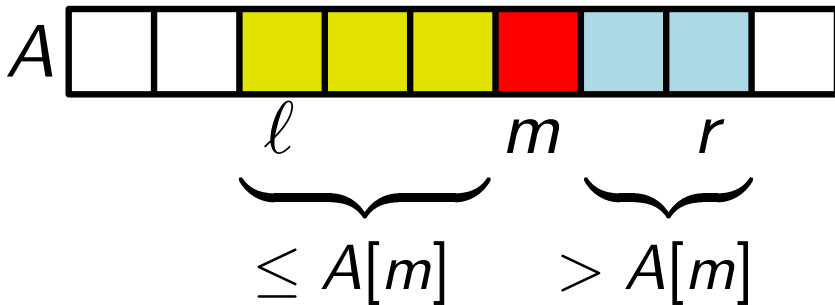
```
QuickSort(int[] A, int  $\ell, r$ )
```

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

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

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

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



# Auswahl per Teile & Herrsche

Zur Erinnerung...

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

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

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

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

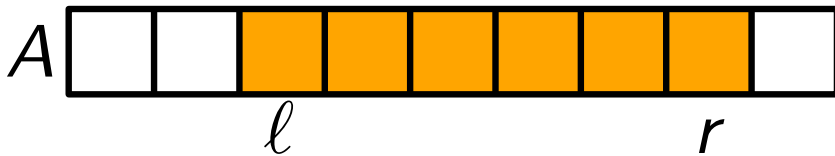
```
    QuickSort(A,  $m + 1, r$ )
```



# Auswahl per Teile & Herrsche

Zur Erinnerung...

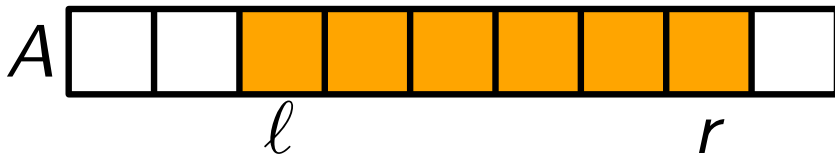
```
Randomized  
QuickSort(int[] A, int  $\ell, r$ )  
if  $\ell < r$  then  
    Randomized  
     $m = \text{Partition}(A, \ell, r)$   
    QuickSort(A,  $\ell, m - 1$ )  
    QuickSort(A,  $m + 1, r$ )
```



# Auswahl per Teile & Herrsche

Zur Erinnerung...

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

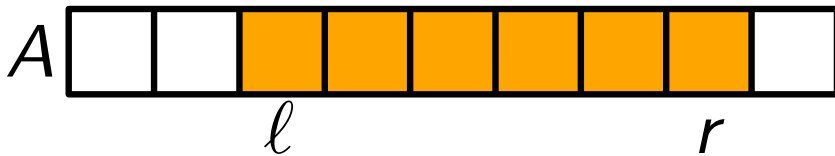


Finde  $i$ -kleinstes Element in  $A[\ell..r]$ !

# Auswahl per Teile & Herrsche

Zur Erinnerung...

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



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

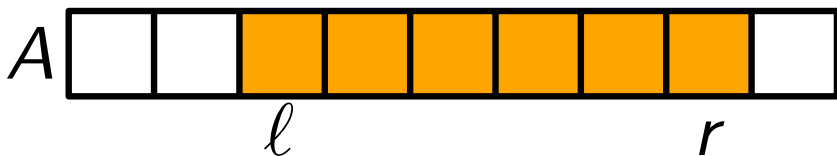
```
RandomizedSelect(int[] A, int  $\ell, r, i$ )
```

# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
if  $\ell < r$  then
     $m =$  RandomizedPartition( $A, \ell, r$ )
    QuickSort( $A, \ell, m - 1$ )
    QuickSort( $A, m + 1, r$ )
  
```



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

```

RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m =$  RandomizedPartition( $A, \ell, r$ )
 $k = m - \ell + 1$ 
if  $i == k$  then
    | return 
else
    | if  $i < k$  then
    | | return 
    | else
    | | return 
  
```

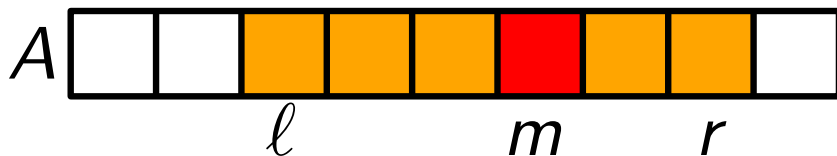


# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
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  $A[\ell..r]$ !

```

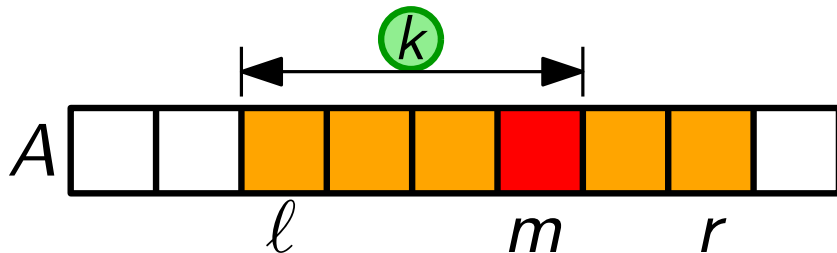
RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = Randomized Partition( $A, \ell, r$ )
 $k = m - \ell + 1$ 
if  $i == k$  then
    | return 
else
    | if  $i < k$  then
    | | return 
    | else
    | | return 
  
```

# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
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  $A[\ell..r]$ !

```

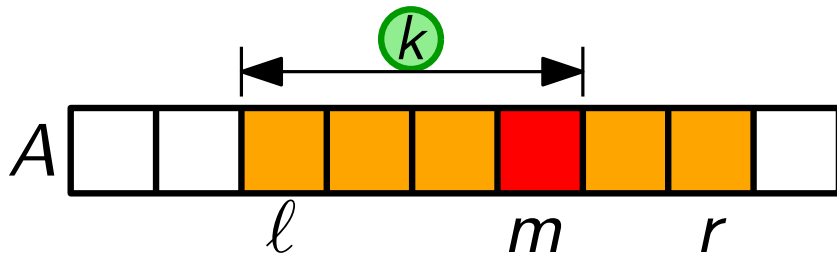
RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = Randomized Partition( $A, \ell, r$ )
 $k$  =  $m - \ell + 1$ 
if  $i == k$  then
    | return [ ]
else
    | if  $i < k$  then
    | | return [ ]
    | else
    | | return [ ]
  
```

# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
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  $A[\ell..r]$ !

```

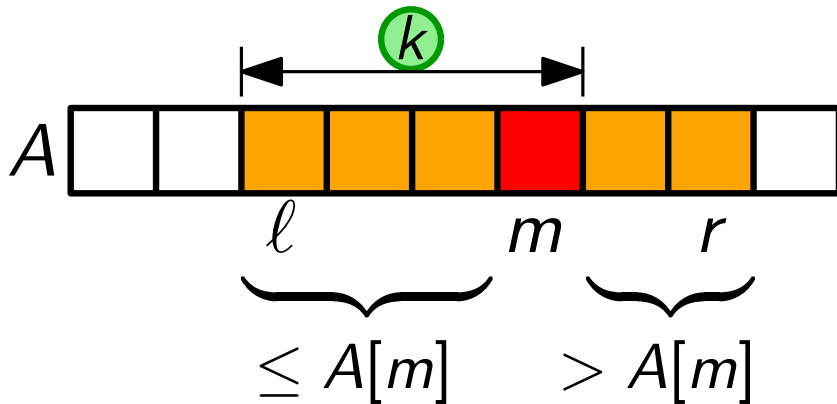
RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = Randomized Partition( $A, \ell, r$ )
 $k$  =  $m - \ell + 1$  // Rang v.  $A[m]$  in  $A[\ell..r]$ 
if  $i == k$  then
    | return 
else
    | if  $i < k$  then
    | | return 
    | else
    | | return 
  
```

# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
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  $A[\ell..r]$ !

```

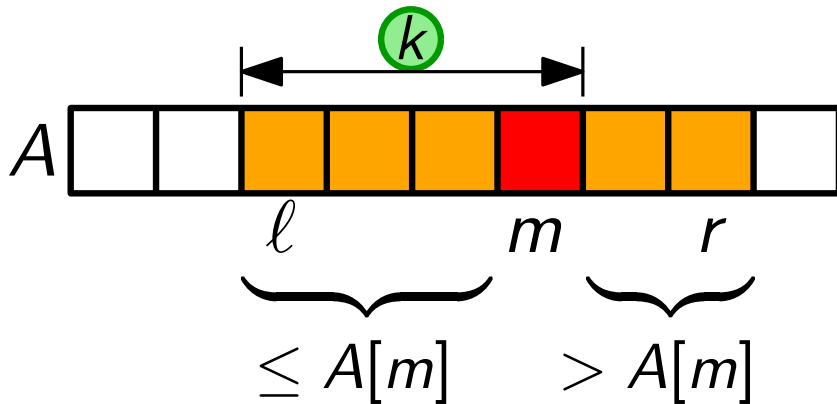
RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = Randomized Partition( $A, \ell, r$ )
 $k$  =  $m - \ell + 1$  // Rang v.  $A[m]$  in  $A[\ell..r]$ 
if  $i == k$  then
    | return 
else
    | if  $i < k$  then
    | | return 
    | else
    | | return 
  
```

# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
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  $A[\ell..r]$ !

```

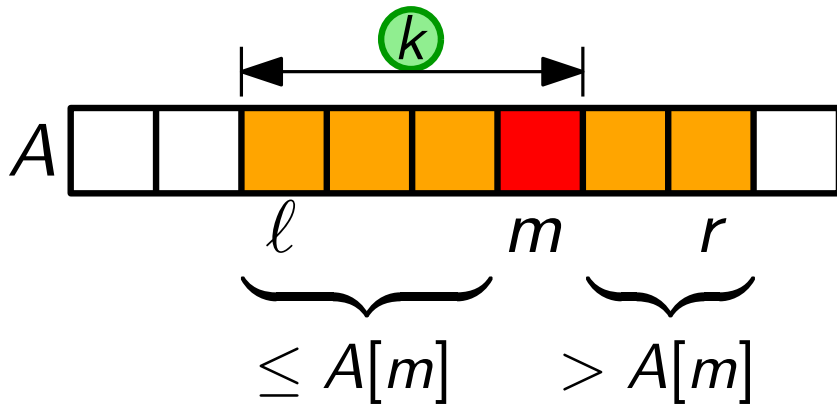
RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = Randomized Partition( $A, \ell, r$ )
 $k$  =  $m - \ell + 1$  // Rang v.  $A[m]$  in  $A[\ell..r]$ 
if  $i == k$  then
    | return  $A[m]$ 
else
    | if  $i < k$  then
    | | return [ ]
    | else
    | | return [ ]
  
```

# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
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  $A[\ell..r]$ !

```

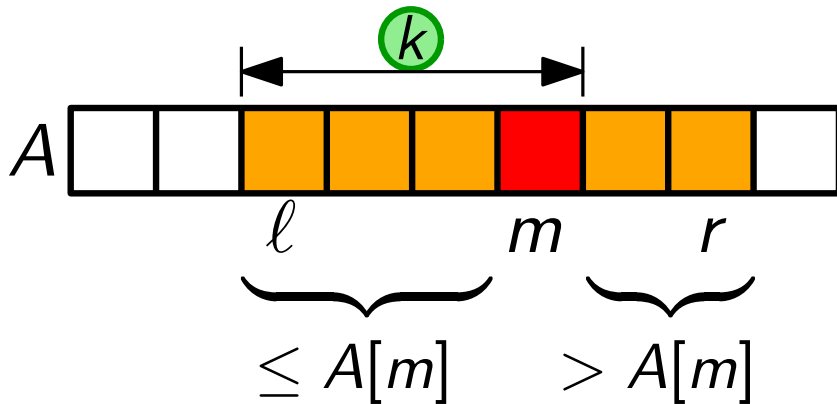
RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = Randomized Partition( $A, \ell, r$ )
 $k$  =  $m - \ell + 1$  // Rang v.  $A[m]$  in  $A[\ell..r]$ 
if  $i == k$  then
    | return  $A[m]$ 
else
    | if  $i < k$  then
    | | return RSelect( $A, \ell, m - 1, i$ )
    | else
    | | return 
  
```

# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
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  $A[\ell..r]$ !

```

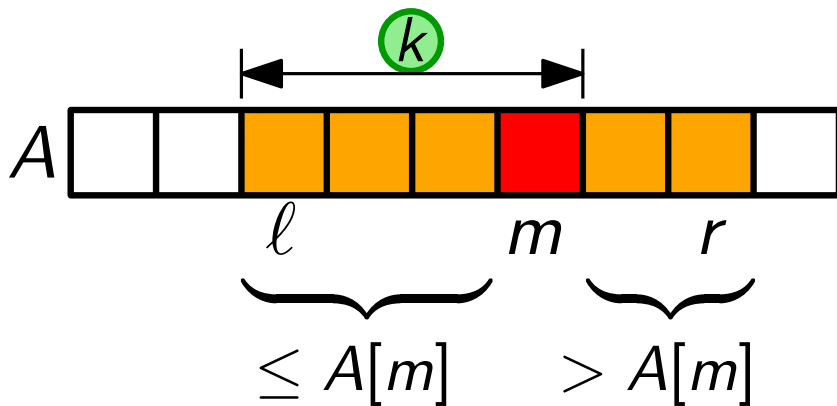
RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = Randomized Partition( $A, \ell, r$ )
 $k$  =  $m - \ell + 1$  // Rang v.  $A[m]$  in  $A[\ell..r]$ 
if  $i == k$  then
    | return  $A[m]$ 
else
    | if  $i < k$  then
    | | return RSelect( $A, \ell, m - 1, i$ )
    | else
    | | return RSelect( $A, m + 1, r, i - k$ )
  
```

# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
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  $A[\ell..r]$ !

```

RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = Randomized Partition( $A, \ell, r$ )
 $k$  =  $m - \ell + 1$  // Rang v.  $A[m]$  in  $A[\ell..r]$ 
if  $i == k$  then
    | return  $A[m]$ 
else
    | if  $i < k$  then
    | | return RSelect( $A, \ell, m - 1, i$ )
    | else
    | | return RSelect( $A, m + 1, r, i - k$ )
  
```

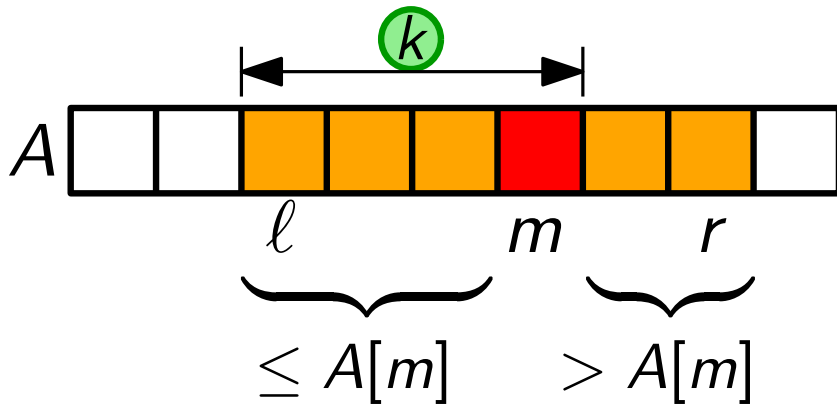


# Auswahl per Teile & Herrsche

Zur Erinnerung...

```

Randomized
QuickSort(int[] A, int  $\ell, r$ )
if  $\ell < r$  then
     $m$  = RandomizedPartition(A,  $\ell, r$ )
    QuickSort(A,  $\ell, m - 1$ )
    QuickSort(A,  $m + 1, r$ )
  
```



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

```

RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = RandomizedPartition(A,  $\ell, r$ )
 $k$  =  $m - \ell + 1$  // Rang v.  $A[m]$  in  $A[\ell..r]$ 
if  $i == k$  then
    | return  $A[m]$ 
else
    | if  $i < k$  then
    | | return RSelect(A,  $\ell, m - 1, i$ )
    | else
    | | return RSelect(A,  $m + 1, r, i - k$ )
  
```

Ist Ihnen klar warum?

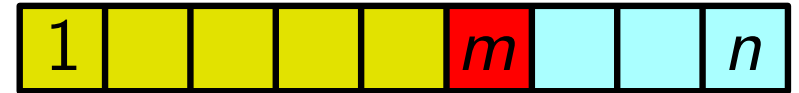
# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

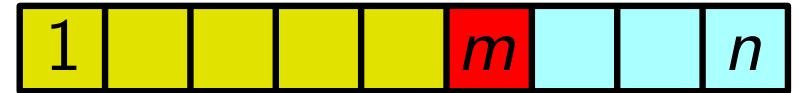
**Trick:** Geh davon aus, dass das gesuchte  $i$ . Element immer im *größeren* Teilfeld liegt.



# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

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

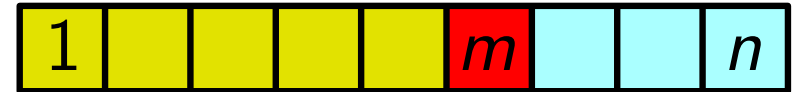


- ⇒ resultierende Zufallsvariable  $V(n)$  ist
- obere Schranke für tatsächliche Anzahl von Vergleichen
  - unabhängig von  $i$

# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

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



⇒ resultierende Zufallsvariable  $V(n)$  ist

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

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

# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

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



⇒ resultierende Zufallsvariable  $V(n)$  ist

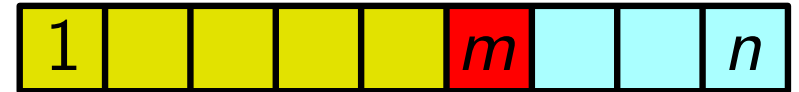
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von  $i$

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

# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

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



⇒ resultierende Zufallsvariable  $V(n)$  ist

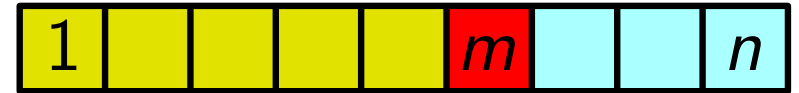
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von  $i$

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

# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

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



⇒ resultierende Zufallsvariable  $V(n)$  ist

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

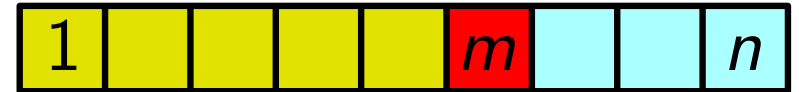
$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n - 1} + \left. \begin{array}{l} V(n-1) \quad \text{falls } m = 1 \\ V(n-2) \quad \text{falls } m = 2 \\ \dots \\ V(\lfloor \frac{n}{2} \rfloor) \quad \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots \\ V(n-2) \quad \text{falls } m = n-1 \\ V(n-1)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



# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

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



⇒ resultierende Zufallsvariable  $V(n)$  ist

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

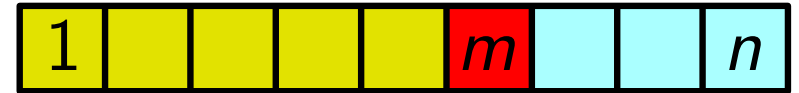
$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n - 1} + \left\{ \begin{array}{ll} V(n-1) & \text{falls } m = 1 \\ V(n-2) & \text{falls } m = 2 \\ \dots & \\ V(\lfloor \frac{n}{2} \rfloor) & \text{falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots & \\ V(n-2) & \text{falls } m = n-1 \\ V(n-1) & \text{falls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

Alle Fälle gleich wahrscheinlich!  
[vorausgesetzt alle Elem. sind verschieden!]

# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

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



⇒ resultierende Zufallsvariable  $V(n)$  ist

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

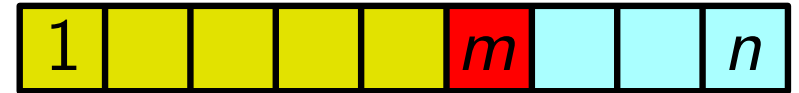
$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n - 1} + \left\{ \begin{array}{l} V(n-1) \text{ falls } m = 1 \\ V(n-2) \text{ falls } m = 2 \\ \dots \\ V(\lfloor \frac{n}{2} \rfloor) \text{ falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots \\ V(n-2) \text{ falls } m = n-1 \\ V(n-1) \text{ falls } m =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

Alle Fälle gleich wahrscheinlich!  
[vorausgesetzt alle Elem. sind verschieden!]

# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

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



⇒ resultierende Zufallsvariable  $V(n)$  ist

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

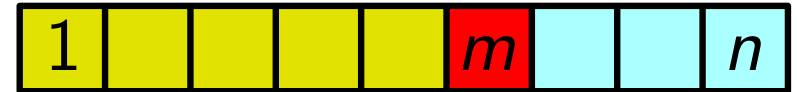
$$V(n) = \underbrace{V_{\text{Part}}(n)}_{= n - 1} + \left. \begin{array}{l} V(n-1) \text{ falls } m = 1 \\ V(n-2) \text{ falls } m = 2 \\ \dots \\ V(\lfloor \frac{n}{2} \rfloor) \text{ falls } m = \lfloor \frac{n}{2} \rfloor + 1 \\ \dots \\ V(n-2) \text{ falls } m = n-1 \\ V(n-1) \text{ falls } m = n \end{array} \right\} \begin{array}{l} \text{Alle Fälle gleich} \\ \text{wahrscheinlich!} \\ \text{[vorausgesetzt} \\ \text{alle Elem. sind} \\ \text{verschieden!]} \end{array}$$

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

# Laufzeit-Analyse

Anz. Vgl. von RandomizedSelect ist ZV; hängt von  $n$  und  $i$  ab.

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



⇒ resultierende Zufallsvariable  $V(n)$  ist

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

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

Alle Fälle gleich wahrscheinlich!  
[vorausgesetzt alle Elem. sind verschieden!]

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

# 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  $f(n) \leq cn$ .

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

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

# 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  $f(n) \leq cn$ .

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

## Aufgabe:

Bestimmen Sie ein  $c$ ,  
so dass  $f(n) \leq cn$ .  
(Ignorieren Sie das  
Abrunden  $\lfloor \dots \rfloor$ .)

Bem.: Wir sind *nicht* an  $\sum_{k=1}^{n/2} c \cdot k$   
interessiert – siehe letzte Folie.  
Die Indizes sind wichtig!

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

# 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  $f(n) \leq cn$ .

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

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

# 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  $f(n) \leq cn$ .

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

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

# 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  $f(n) \leq cn$ .

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

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

# 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  $f(n) \leq cn$ .

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

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

# Substitutionsmethode

Wir schreiben  $f(n)$  für  $E[V(n)]$ .

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

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

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

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

# 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  $f(n) \leq cn$ .

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

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



# 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  $f(n) \leq cn$ .

$$\begin{aligned}
 \text{Also: } f(n) &\leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k \quad [\text{laut Annahme}] \\
 &= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
 &= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
 &\leq n + \frac{c}{n} (n(n-1) - (n/2 - 1)(n/2 - 2)) \\
 &\leq n + c \cdot \frac{3n+2}{4} = cn - \left( c \cdot \frac{n-2}{4} - n \right) \stackrel{?!}{\geq} 0 \\
 &\leq cn
 \end{aligned}$$

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

# 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  $f(n) \leq cn$ .

$$\begin{aligned}
 \text{Also: } f(n) &\leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k \quad [\text{laut Annahme}] \\
 &= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
 &= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
 &\leq n + \frac{c}{n} (n(n-1) - (n/2 - 1)(n/2 - 2)) \\
 &\leq n + c \cdot \frac{3n+2}{4} = cn - \left( c \cdot \frac{n-2}{4} - n \right) \stackrel{?!}{\geq} 0 \\
 &\leq cn \quad \text{falls } c \geq \frac{4n}{n-2} =
 \end{aligned}$$

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

# 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  $f(n) \leq cn$ .

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

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

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

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

$$\leq n + c \cdot \frac{3n+2}{4} = cn - \left( c \cdot \frac{n-2}{4} - n \right) \stackrel{?!}{\geq} 0$$

$$\leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow{n \rightarrow \infty}$$

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

# 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  $f(n) \leq cn$ .

$$\begin{aligned}
 \text{Also: } f(n) &\leq n + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c \cdot k \quad [\text{laut Annahme}] \\
 &= n + \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) \\
 &= n + \frac{2c}{n} \left( \frac{n(n-1)}{2} - \frac{\lfloor n/2 \rfloor (\lfloor n/2 \rfloor - 1)}{2} \right) \\
 &\leq n + \frac{c}{n} (n(n-1) - (n/2 - 1)(n/2 - 2)) \\
 &\leq n + c \cdot \frac{3n+2}{4} = cn - \left( c \cdot \frac{n-2}{4} - n \right) \stackrel{?!}{\geq} 0 \\
 &\leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow{n \rightarrow \infty} 4^+
 \end{aligned}$$

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

# Substitutionsmethode

Wir schreiben  $f(n)$  für  $E[V(n)]$ .

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

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

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

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

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

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

$$\leq n + c \cdot \frac{3n+2}{4} = cn - \left( c \cdot \frac{n-2}{4} - n \right) \stackrel{?!}{\geq} 0$$

$$\leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow{n \rightarrow \infty} 4^+$$

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

# Substitutionsmethode

Wir schreiben  $f(n)$  für  $E[V(n)]$ .

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

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

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

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

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

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

$$\leq n + c \cdot \frac{3n+2}{4} = cn - \left( c \cdot \frac{n-2}{4} - n \right) \stackrel{?!}{\geq} 0$$

$$\leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow{n \rightarrow \infty} 4^+$$

$$E[V(n)] \leq n-1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)] \leq \overbrace{(4 + \varepsilon)n}^{c :=} \left[ \text{falls } n \geq \frac{8}{\varepsilon} + 2 \right]$$

# 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  $f(n) \leq cn$ .

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

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

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

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

$$\leq n + c \cdot \frac{3n+2}{4} = cn - \left( c \cdot \frac{n-2}{4} - n \right) \stackrel{?!}{\geq} 0$$

$$\leq cn \quad \text{falls } c \geq \frac{4n}{n-2} = \frac{4}{1-2/n} \xrightarrow{n \rightarrow \infty} 4^+$$

Für jedes  $\varepsilon > 0$  gilt:

$$E[V(n)] \leq n-1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)] \leq \overbrace{(4 + \varepsilon)n}^{c :=} \left[ \text{falls } n \geq \frac{8}{\varepsilon} + 2 \right]$$

# Ergebnis und Diskussion

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



# Ergebnis und Diskussion

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

**Genauer:**

# Ergebnis und Diskussion

**Satz.** Das Auswahlproblem kann in erwarteter 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 erwarteter 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 erwarteter 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 erwarteter 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 Teile-und-Herrsche –

# Vorbereitung

Wir verwenden wieder Teile-und-Herrsche –  
aber diesmal mit einer garantiert *guten* Aufteilung in Teilfelder.

# Vorbereitung

Wir verwenden wieder Teile-und-Herrsche –  
aber diesmal mit einer garantiert *guten* Aufteilung in Teilfelder.



# Vorbereitung

Wir verwenden wieder Teile-und-Herrsche –  
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.  
d.h. *balanciert*:

# Vorbereitung

Wir verwenden wieder Teile-und-Herrsche –  
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 Teile-und-Herrsche –  
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 ( $A, \ell, r$ )
   $pivot = A[r]$ 
   $i = \ell - 1$ 
  for  $j = \ell$  to  $r - 1$  do
    if  $A[j] \leq pivot$  then
       $i = i + 1$ 
      Swap( $A, i, j$ )
  Swap( $A, i + 1, r$ )
  return  $i + 1$ 
  
```

# Vorbereitung

Wir verwenden wieder Teile-und-Herrsche –  
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'(A, l, r)
  pivot = A[r]
  i = l - 1
  for j = l to r - 1 do
    if A[j] ≤ pivot then
      i = i + 1
      Swap(A, i, j)
  Swap(A, i + 1, r)
  return i + 1
```

# Vorbereitung

Wir verwenden wieder Teile-und-Herrsche –  
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'(A, l, r, pivot)
  pivot = A[r]
  i = l - 1
  for j = l to r - 1 do
    if A[j] ≤ pivot then
      i = i + 1
      Swap(A, i, j)
  Swap(A, i + 1, r)
  return i + 1
  
```

# Vorbereitung

Wir verwenden wieder Teile-und-Herrsche –  
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'(A, l, r, pivot)
  pivot = A[r]
  i = l - 1
  for j = l to r >= 1 do
    if A[j] ≤ pivot then
      i = i + 1
      Swap(A, i, j)
  Swap(A, i + 1, r)
  return i + 1

```

# Vorbereitung

Wir verwenden wieder Teile-und-Herrsche –  
aber diesmal mit einer garantiert **guten** Aufteilung in Teilfelder.

d.h. *balanciert*:

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

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

```

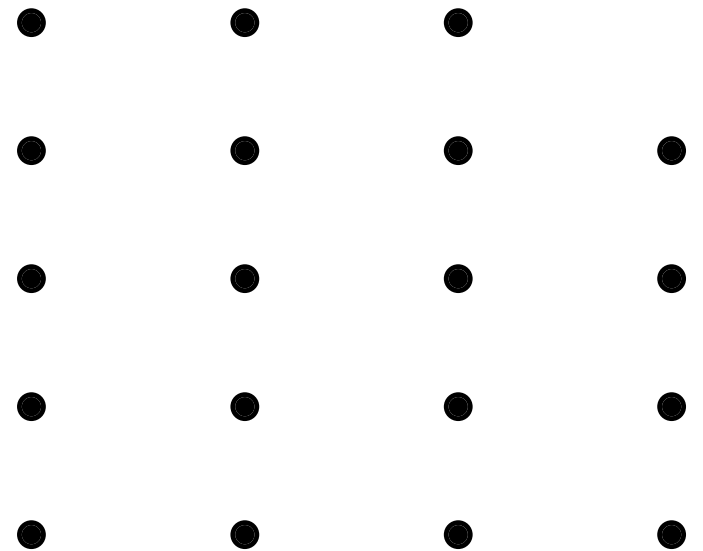
Partition'(A, l, r, pivot)
  pivot = A[r]
  i = l - 1
  for j = l to r >= 1 do
    if A[j] ≤ pivot then
      i = i + 1
      Swap(A, i, j)
  Swap(A, i + 1, r)
  return i + 1

```

# Select: deterministisch

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

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

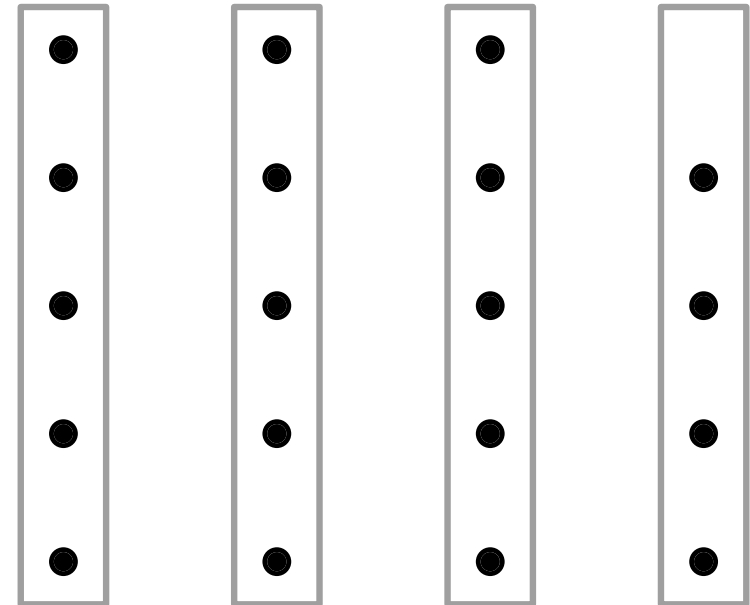




# Select: deterministisch

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

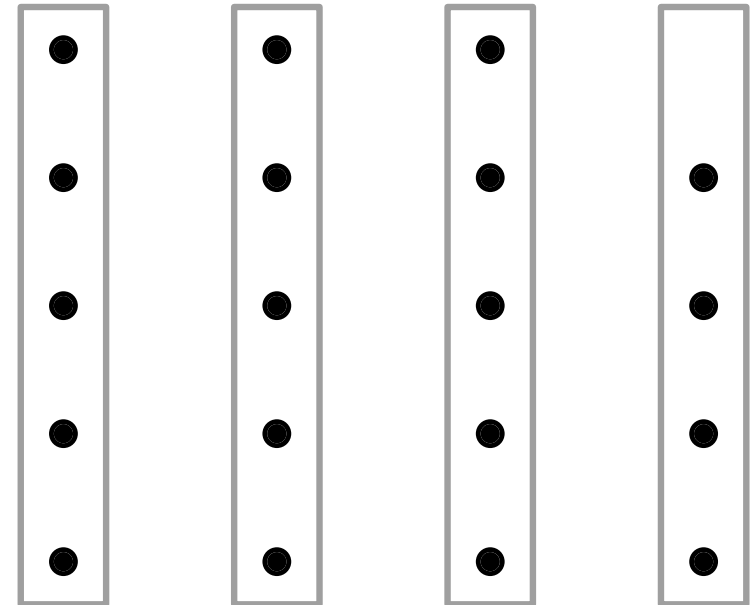
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen  $(n \bmod 5)$  Elem.



# Select: deterministisch

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

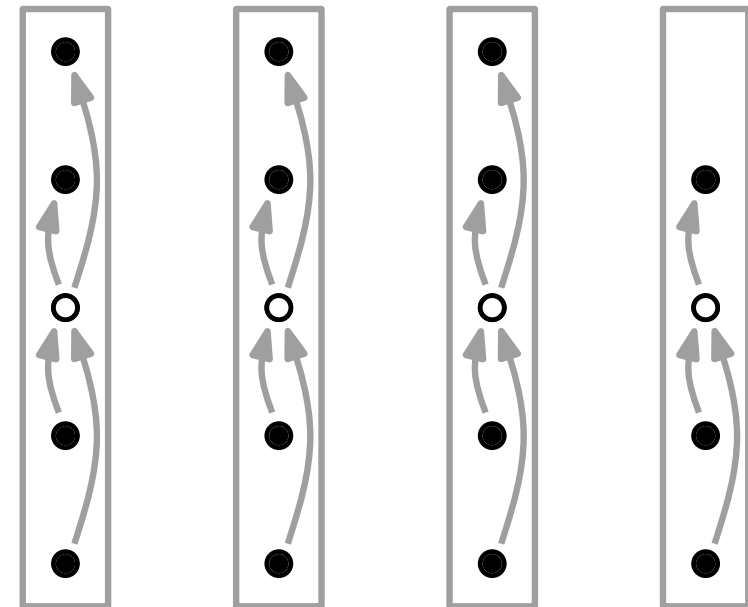
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen  $(n \bmod 5)$  Elem.
2. Sortiere jede der  $\lceil n/5 \rceil$  Gruppen und bestimme ihren Median.



# Select: deterministisch

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

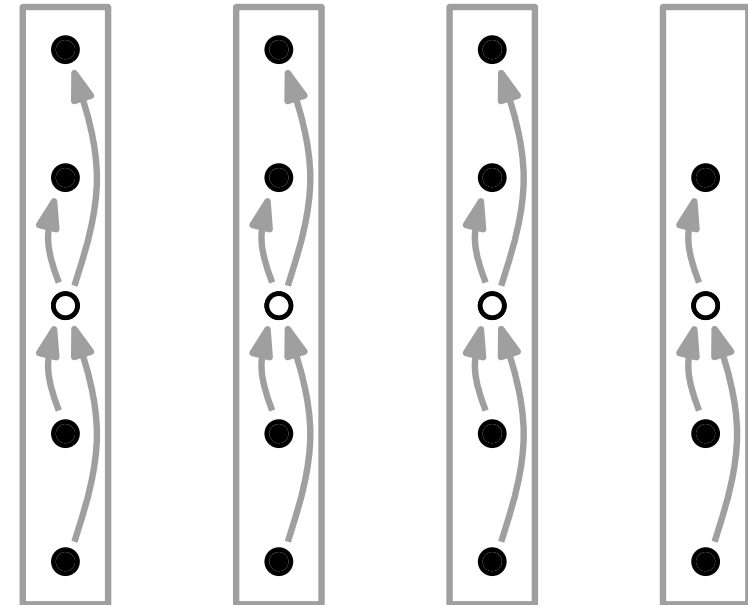
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
2. Sortiere jede der  $\lceil n/5 \rceil$  Gruppen und bestimme ihren Median.



# Select: deterministisch

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

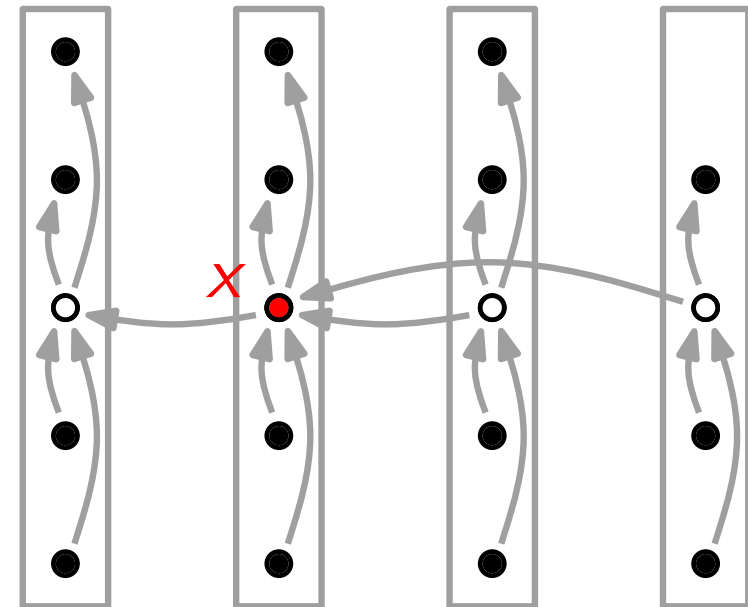
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
2. Sortiere jede der  $\lfloor n/5 \rfloor$  Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median  $x$  der Gruppen-Mediane.



# Select: deterministisch

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

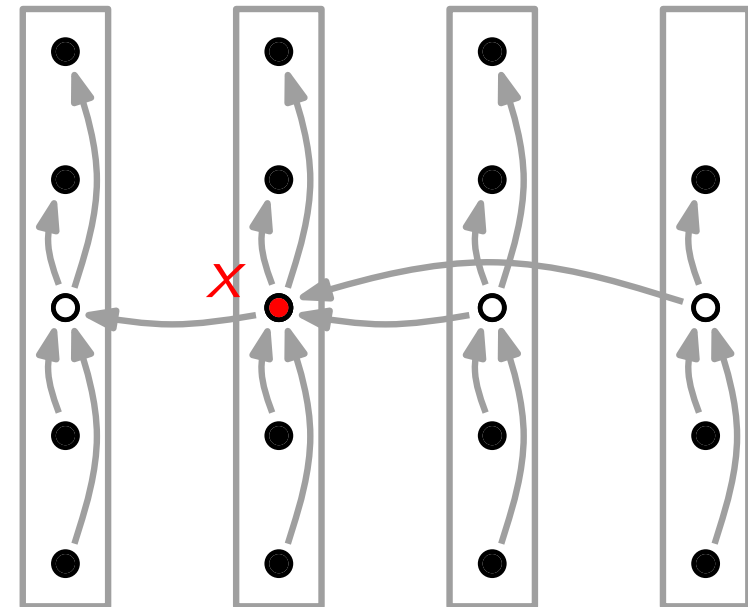
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
2. Sortiere jede der  $\lfloor n/5 \rfloor$  Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median  $x$  der Gruppen-Mediane.



# Select: deterministisch

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

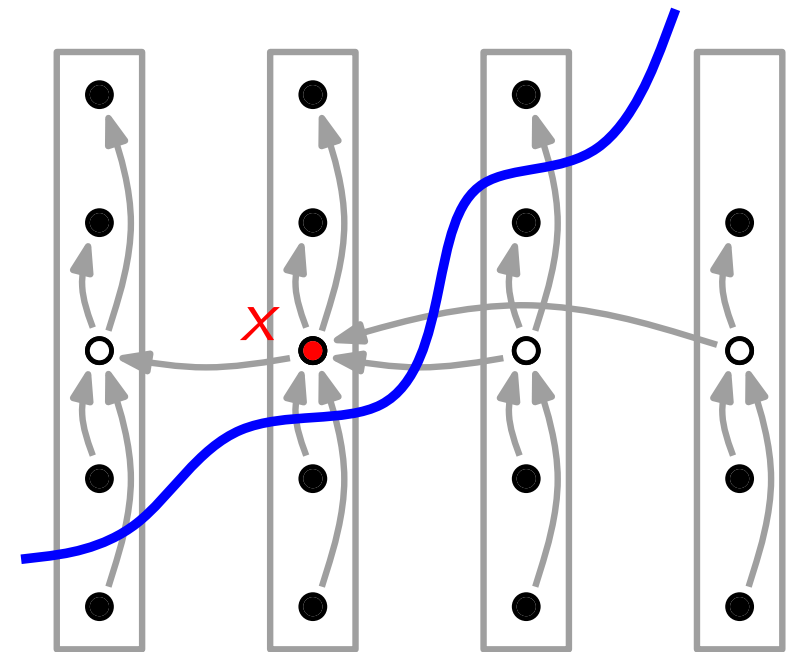
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
2. Sortiere jede der  $\lfloor n/5 \rfloor$  Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median  $x$  der Gruppen-Mediane.
4.  $m = \text{Partition}'(A,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



# Select: deterministisch

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

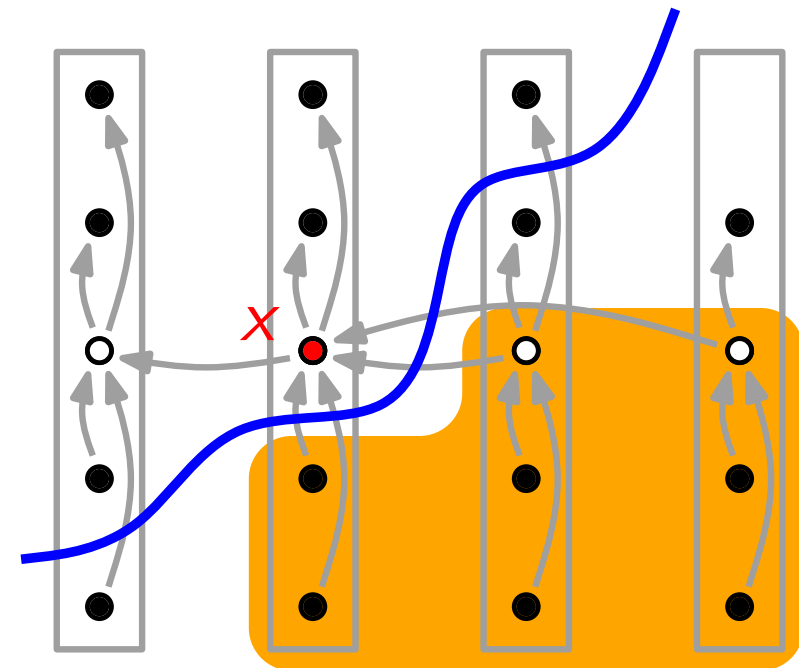
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
2. Sortiere jede der  $\lfloor n/5 \rfloor$  Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median  $x$  der Gruppen-Mediane.
4.  $m = \text{Partition}'(A,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



# Select: deterministisch

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

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

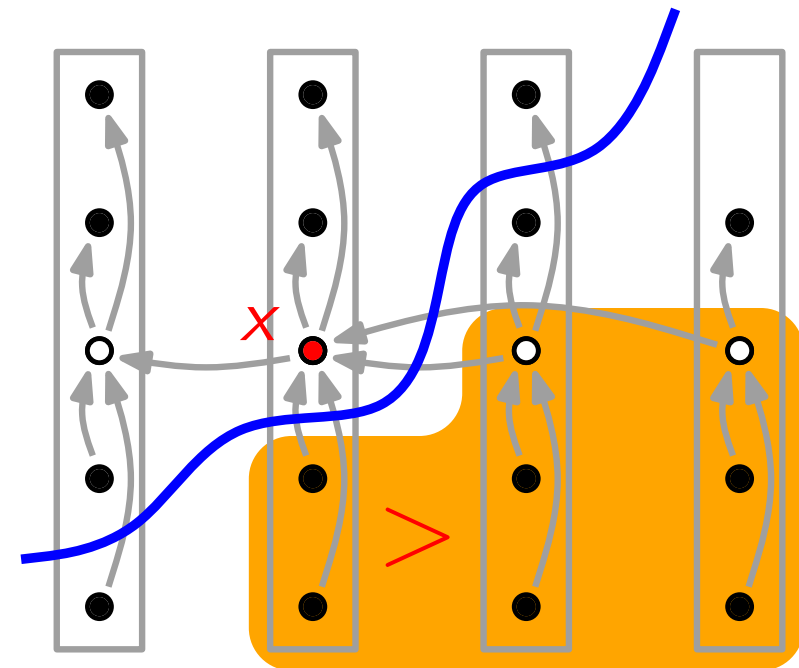




# Select: deterministisch

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

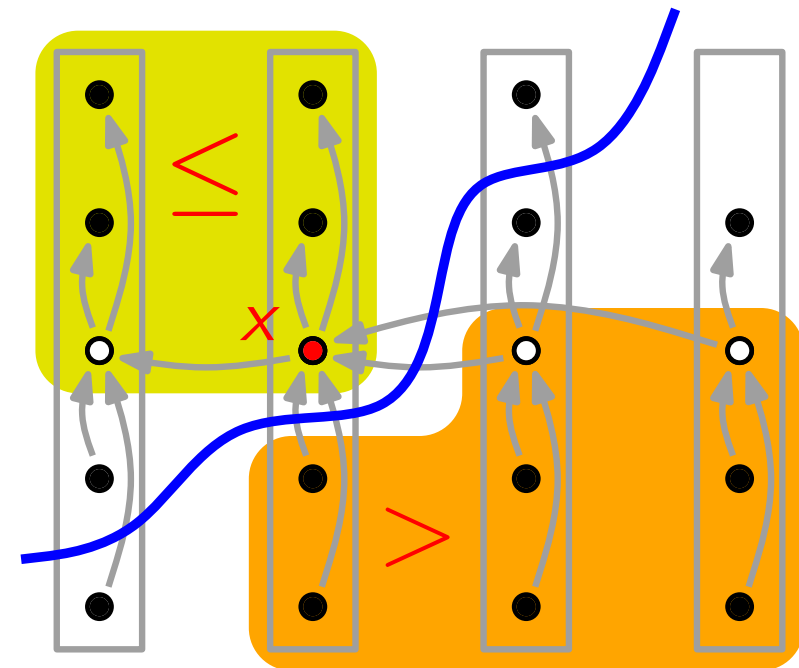
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
2. Sortiere jede der  $\lceil n/5 \rceil$  Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median  $x$  der Gruppen-Mediane.
4.  $m = \text{Partition}'(A,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



# Select: deterministisch

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

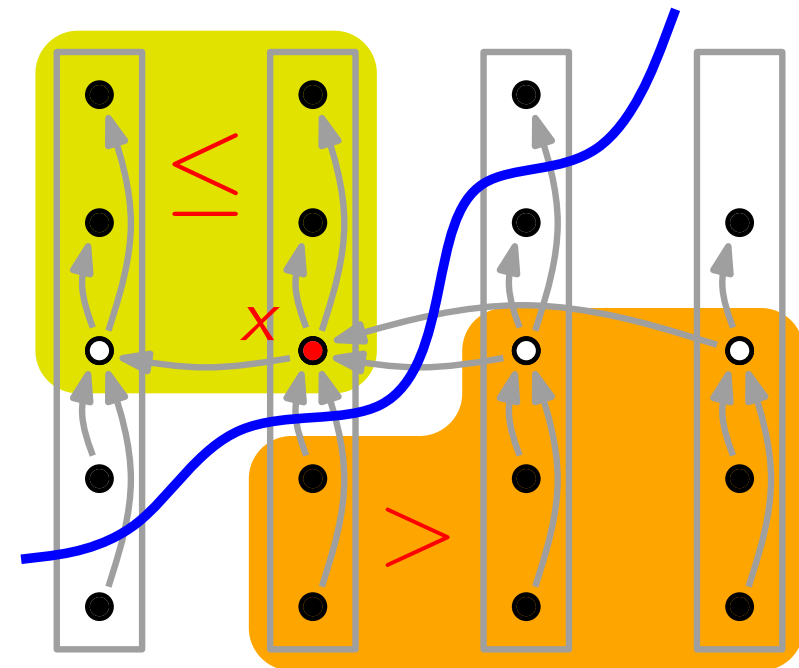
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
2. Sortiere jede der  $\lceil n/5 \rceil$  Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median  $x$  der Gruppen-Mediane.
4.  $m = \text{Partition}'(A,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



# Select: deterministisch

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

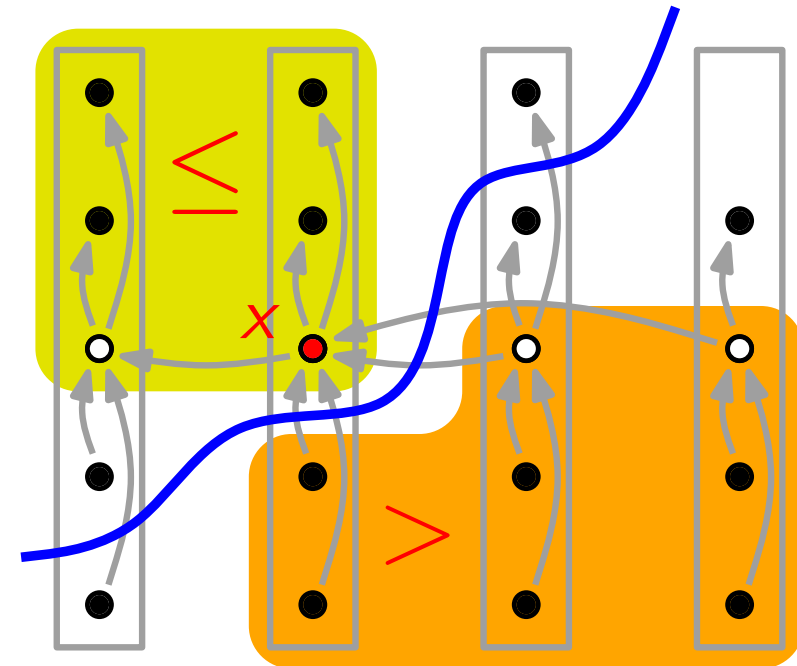
1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
2. Sortiere jede der  $\lceil n/5 \rceil$  Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median  $x$  der Gruppen-Mediane.
4.  $m = \text{Partition}'(A,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



# Select: deterministisch

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

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



# Select: deterministisch

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

1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
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 El.
5. if  $i == k$  then return  $x$

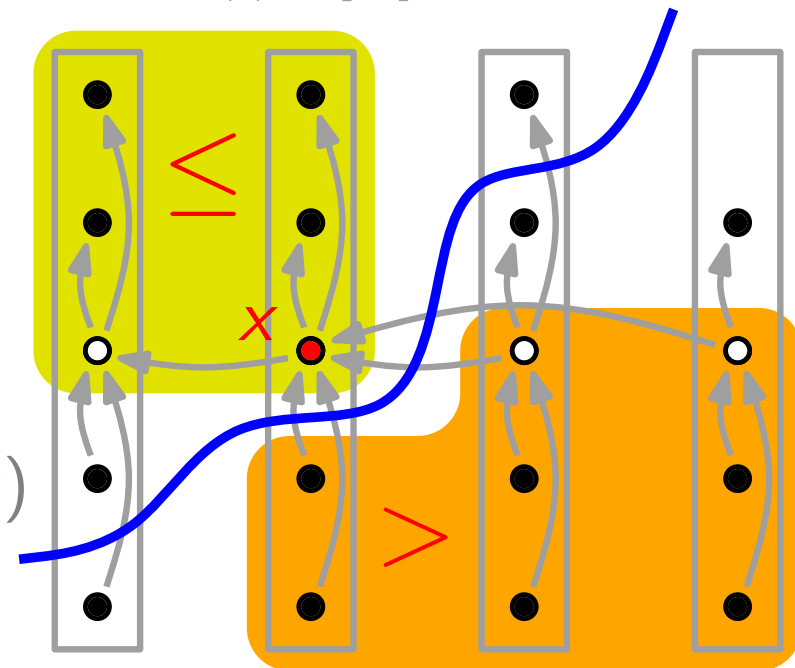
else

if  $i < k$  then

return Select( $A, \ell, m, i$ )

else

return Select( $A, m + 1, r, i - k$ )



# Select: deterministisch

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

1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
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 El.
5. if  $i == k$  then return  $x$

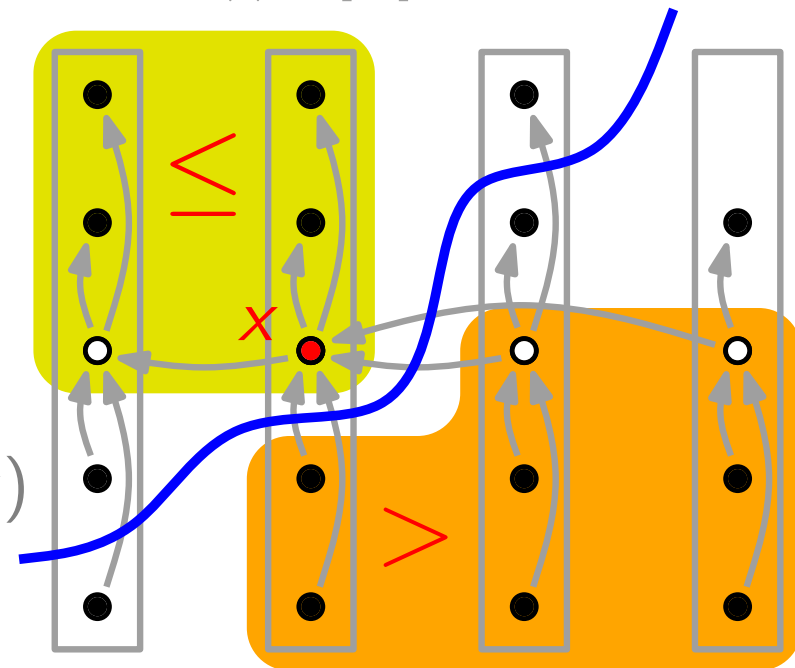
else

if  $i < k$  then

return Select( $A, \ell, m, i$ )

else

return Select( $A, m + 1, r, i - k$ )



# Select: deterministisch

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

1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
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 El.
5. if  $i == k$  then return  $x$

else

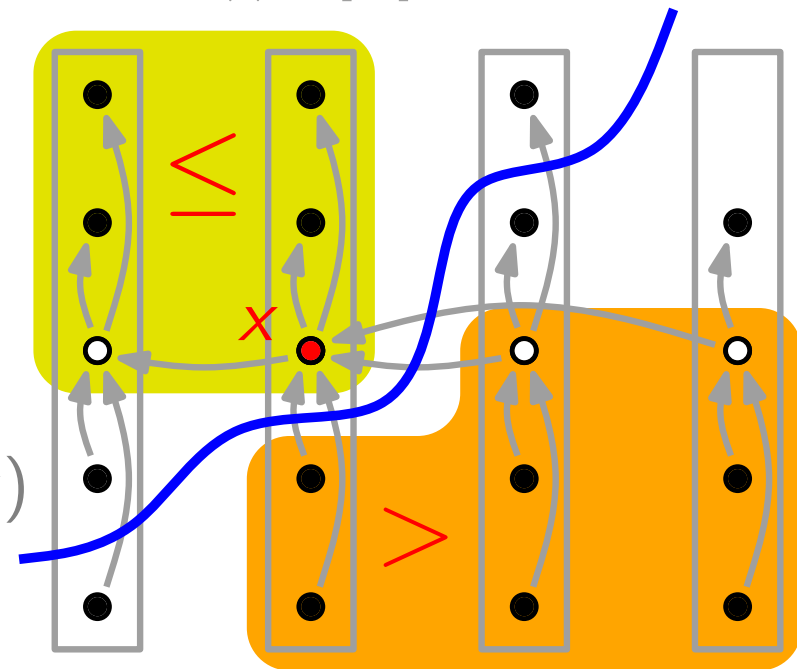
if  $i < k$  then

return Select( $A, \ell, m, i$ )

else

return Select( $A, m + 1, r, i - k$ )

Anzahl()  $\geq$



# Select: deterministisch

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

1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
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 El.
5. if  $i == k$  then return  $x$


else

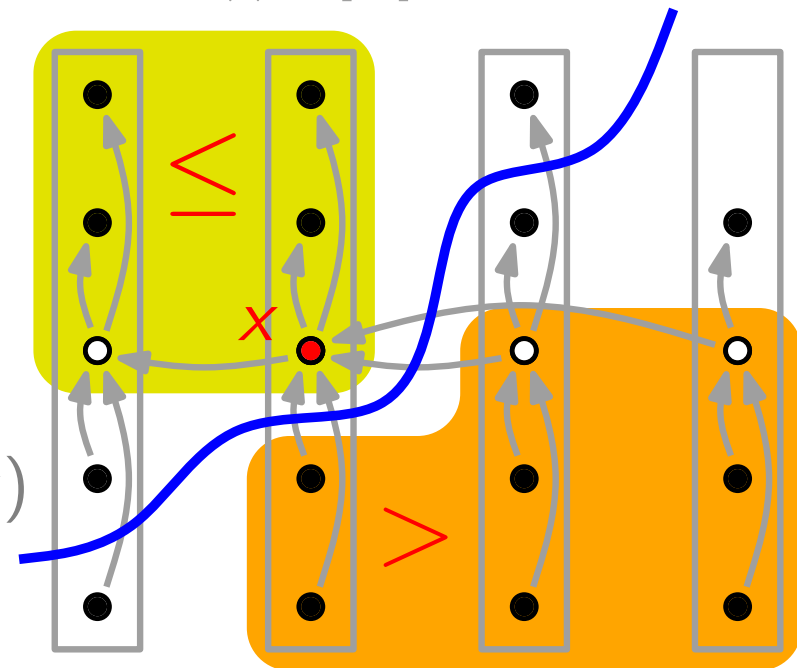
if  $i < k$  then

return Select( $A, \ell, m, i$ )

else

return Select( $A, m + 1, r, i - k$ )

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





# Select: deterministisch

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

1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
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 El.
5. if  $i == k$  then return  $x$


else

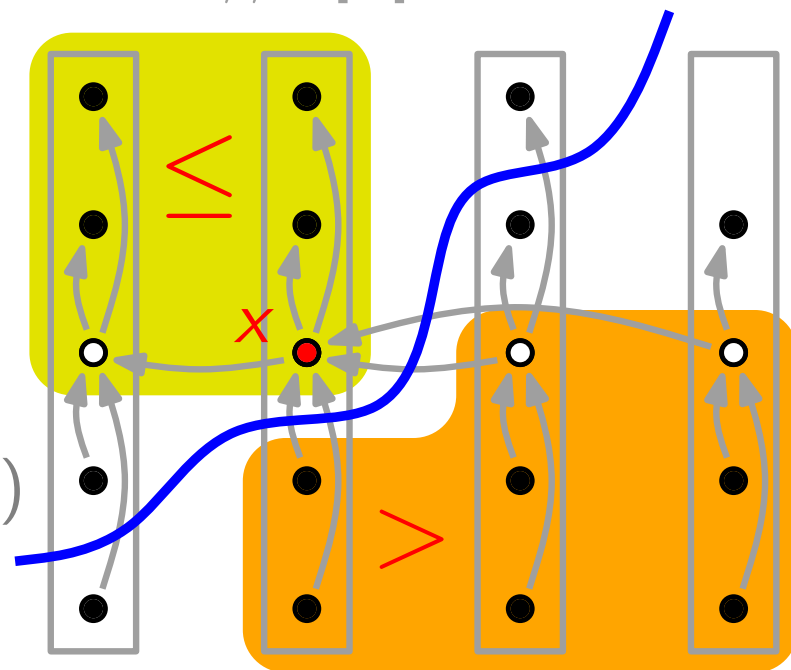
if  $i < k$  then

return Select( $A, \ell, m, i$ )

else

return Select( $A, m + 1, r, i - k$ )

Anzahl()  $\geq 3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6$



# Select: deterministisch

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

1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
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 El.
5. if  $i == k$  then return  $x$


else

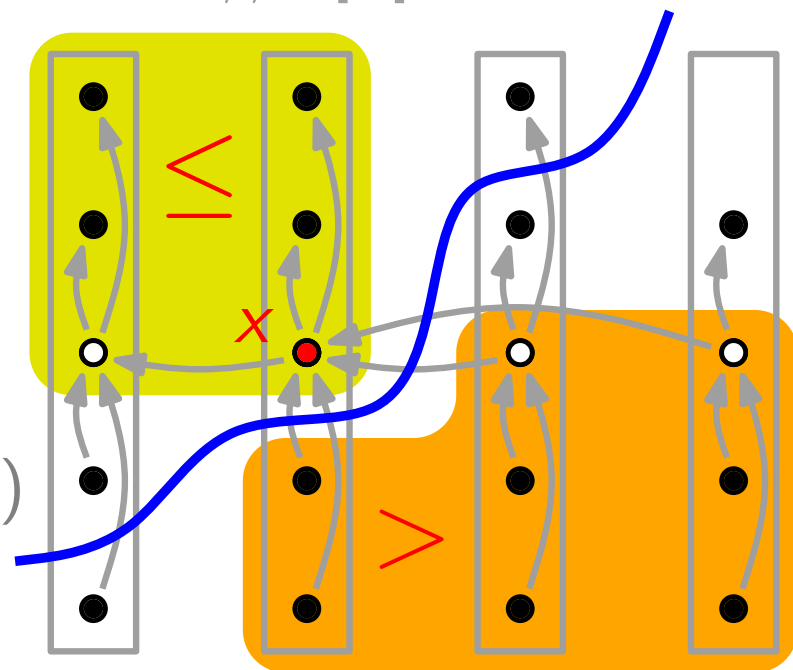
if  $i < k$  then

return Select( $A, \ell, m, i$ )

else

return Select( $A, m + 1, r, i - k$ )

Anzahl()  $\geq 3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6$



# Select: deterministisch

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

1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
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 El.
5. if  $i == k$  then return  $x$


else

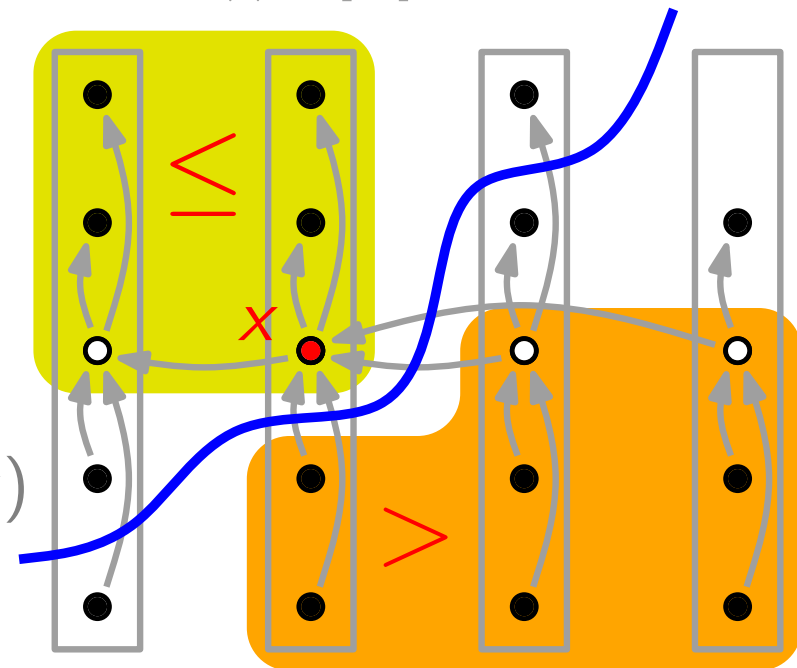
if  $i < k$  then

return Select( $A, \ell, m, i$ )

else

return Select( $A, m + 1, r, i - k$ )

Anzahl()  $\geq 3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6$



# Select: deterministisch

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

1. Teile die  $n$  Elem. der Eingabe in  $\lfloor n/5 \rfloor$  5er-Gruppen und eine Gruppe mit den restlichen ( $n \bmod 5$ ) Elem.
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 El.
5. if  $i == k$  then return  $x$


else

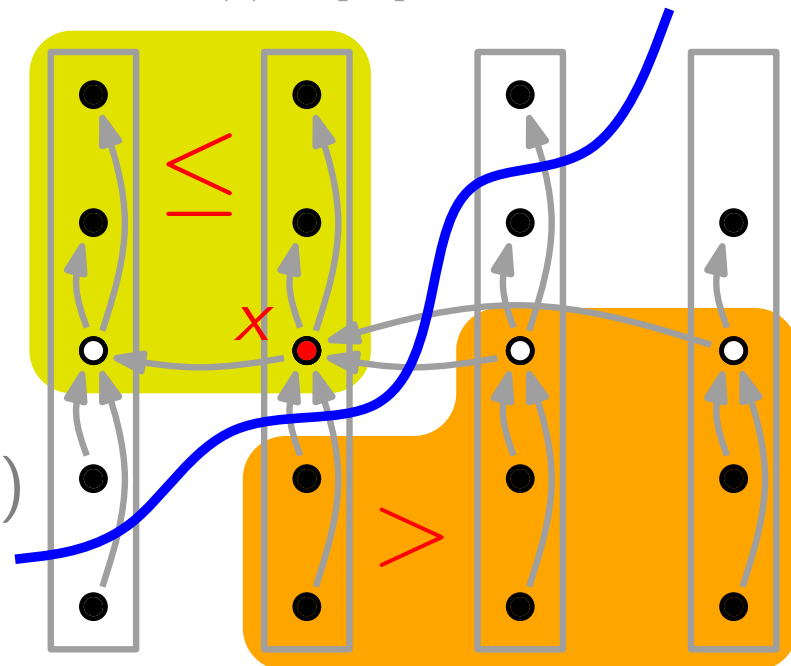
if  $i < k$  then

return Select( $A, \ell, m, i$ )

else

return Select( $A, m + 1, r, i - k$ )

Anzahl()  $\geq 3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6$



# Laufzeit-Analyse

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

# Laufzeit-Analyse

**Beob.** Es genügt wieder, Vergleiche zu zählen!  
Partition':  $\approx 1n$

# Laufzeit-Analyse

**Beob.** Es genügt wieder, Vergleiche zu zählen!  
Partition':  $\approx 1n$ , Sortieren:  $\approx \frac{n}{5} \cdot V_{IS}(5) = 2n$  Vgl.

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$



# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} \overbrace{V'(\lceil n/5 \rceil) + V'(7n/10 + 6)}^{\text{Schritt 3}} + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

Schritt 5

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

# Laufzeit-Analyse

**Beob.** Es genügt wieder, Vergleiche zu zählen!  
 Partition':  $\approx 1n$ , Sortieren:  $\approx \frac{n}{5} \cdot V_{IS}(5) = 2n$  Vgl.

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\Rightarrow V'(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n$$

# Laufzeit-Analyse

**Beob.** Es genügt wieder, Vergleiche zu zählen!  
 Partition':  $\approx 1n$ , Sortieren:  $\approx \frac{n}{5} \cdot V_{IS}(5) = 2n$  Vgl.

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\begin{aligned} \Rightarrow V'(n) &\leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n \\ &= c \cdot (9n/10 + 7) + 3n \end{aligned}$$

# Laufzeit-Analyse

**Beob.** Es genügt wieder, Vergleiche zu zählen!  
 Partition':  $\approx 1n$ , Sortieren:  $\approx \frac{n}{5} \cdot V_{IS}(5) = 2n$  Vgl.

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\begin{aligned} \Rightarrow V'(n) &\leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n \\ &= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n) \end{aligned}$$

# Laufzeit-Analyse

**Beob.** Es genügt wieder, Vergleiche zu zählen!  
 Partition':  $\approx 1n$ , Sortieren:  $\approx \frac{n}{5} \cdot V_{IS}(5) = 2n$  Vgl.

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\begin{aligned} \Rightarrow V'(n) &\leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n \\ &= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n) \end{aligned}$$

# Laufzeit-Analyse

**Beob.** Es genügt wieder, Vergleiche zu zählen!  
 Partition':  $\approx 1n$ , Sortieren:  $\approx \frac{n}{5} \cdot V_{IS}(5) = 2n$  Vgl.

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\begin{aligned} \Rightarrow V'(n) &\leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n && \stackrel{?!}{\geq} 0 \\ &= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n) \end{aligned}$$

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\begin{aligned} \Rightarrow V'(n) &\leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n && \stackrel{?!}{\geq} 0 \\ &= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n) \end{aligned}$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} =$$



# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\begin{aligned} \Rightarrow V'(n) &\leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n && \stackrel{?!}{\geq} 0 \\ &= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n) \end{aligned}$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty}$$

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\begin{aligned} \Rightarrow V'(n) &\leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n && \stackrel{?!}{\geq} 0 \\ &= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n) \end{aligned}$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty} 30^+$$

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\begin{aligned} \Rightarrow V'(n) &\leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n && \stackrel{?!}{\geq} 0 \\ &= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n) \end{aligned}$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty} 30^+ \quad \text{bzw. } n \geq \frac{70c}{c - 30}.$$

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\Rightarrow V'(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n$$

$$= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n)$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty} 30^+ \quad \text{bzw. } n \geq \frac{70c}{c - 30}.$$

$\Rightarrow$  für jedes  $\varepsilon > 0$    gilt:  $V'(n) \leq \underbrace{(30 + \varepsilon)}_c \cdot n$

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\Rightarrow V'(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n$$

$$= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n)$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty} 30^+ \quad \text{bzw. } n \geq \frac{70c}{c - 30}.$$

$$\Rightarrow \text{für jedes } \varepsilon > 0 \text{ und } n \geq \frac{2100}{\varepsilon} + 70 \text{ gilt: } V'(n) \leq \underbrace{(30 + \varepsilon)}_c \cdot n$$

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\Rightarrow V'(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n$$

$$= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n)$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty} 30^+ \quad \text{bzw. } n \geq \frac{70c}{c - 30}.$$

$$\Rightarrow \text{für jedes } \varepsilon > 0 \text{ und } n \geq \frac{2100}{\varepsilon} + 70 \text{ gilt: } V'(n) \leq (30 + \varepsilon) \cdot n$$

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\Rightarrow V'(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n$$

$$= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n)$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty} 30^+ \quad \text{bzw. } n \geq \frac{70c}{c - 30}.$$

$$\Rightarrow \text{für jedes } \varepsilon > 0 \text{ und } n \geq \frac{2100}{\varepsilon} + 70 \text{ gilt: } V'(n) \leq (30 + \varepsilon) \cdot n$$

# Laufzeit-Analyse

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

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

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\Rightarrow V'(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n$$

$$= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n)$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty} 30^+ \quad \text{bzw. } n \geq \frac{70c}{c - 30}.$$

$$\Rightarrow \text{für jedes } \varepsilon > 0 \text{ und } n \geq \frac{2100}{\varepsilon} + 70 \text{ gilt: } V'(n) \leq (30 + \varepsilon) \cdot n$$



# Laufzeit-Analyse

Kann man das verbessern?

**Beob.** Es genügt wieder, Vergleiche zu zählen!  
 Partition':  $\approx 1n$ , Sortieren:  $\approx \frac{n}{5} \cdot V_{IS}(5) = 2n$  Vgl.

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\Rightarrow V'(n) \leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n$$

$$= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n)$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty} 30^+ \quad \text{bzw. } n \geq \frac{70c}{c - 30}.$$

$\Rightarrow$  für jedes  $\varepsilon > 0$  und  $n \geq \frac{2100}{\varepsilon} + 70$  gilt:  $V'(n) \leq (30 + \varepsilon) \cdot n$

# Laufzeit-Analyse

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

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

Kann man das verbessern?  
Hausaufgabe!

**Ansatz:**

$$V'(n) \leq \begin{cases} V'(\lceil n/5 \rceil) + V'(7n/10 + 6) + 3n & \text{falls } n \geq n_0, \\ O(1) & \text{sonst.} \end{cases}$$

**Behauptung:**

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

$$\begin{aligned} \Rightarrow V'(n) &\leq c \cdot (n/5 + 1) + c \cdot (7n/10 + 6) + 3n \\ &= c \cdot (9n/10 + 7) + 3n = cn - (c \cdot (n/10 - 7) - 3n) \end{aligned}$$

$$\text{falls } c \geq \frac{3n}{n/10 - 7} = \frac{30}{1 - 70/n} \xrightarrow{n \rightarrow \infty} 30^+ \quad \text{bzw. } n \geq \frac{70c}{c - 30}.$$

$\Rightarrow$  für jedes  $\varepsilon > 0$  und  $n \geq \frac{2100}{\varepsilon} + 70$  gilt:  $V'(n) \leq (30 + \varepsilon) \cdot n$

# 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  $n \geq 2100/\varepsilon + 70$  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  $n \geq 2100/\varepsilon + 70$  Zahlen die  $i$ -kleinste Zahl mit höchstens  $(30 + \varepsilon)n$  Vergleichen finden kann.

**Literatur:** *Randomized Algorithms* [Motwani+Raghavan, Cambridge U Press, '95]  
*Algorithmen und Zufall* [Vorlesungsskript, Jochen Geiger, Uni KL]

# 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  $n \geq 2100/\varepsilon + 70$  Zahlen die  $i$ -kleinste Zahl mit höchstens  $(30 + \varepsilon)n$  Vergleichen finden kann.

**Literatur:** *Randomized Algorithms* [Motwani+Raghavan, Cambridge U Press, '95]  
*Algorithmen und Zufall* [Vorlesungsskript, Jochen Geiger, Uni KL]

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit WK  $1 - O(1/\sqrt[4]{n})$  mit  $\frac{3}{2}n + o(n)$  Vgl.

# 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  $n \geq 2100/\varepsilon + 70$  Zahlen die  $i$ -kleinste Zahl mit höchstens  $(30 + \varepsilon)n$  Vergleichen finden kann.

**Literatur:** *Randomized Algorithms* [Motwani+Raghavan, Cambridge U Press, '95]  
*Algorithmen und Zufall* [Vorlesungsskript, Jochen Geiger, Uni KL]

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit WK  $1 - O(1/\sqrt[4]{n})$  mit  $\frac{3}{2}n + o(n)$  Vgl.
- 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  $n \geq 2100/\varepsilon + 70$  Zahlen die  $i$ -kleinste Zahl mit höchstens  $(30 + \varepsilon)n$  Vergleichen finden kann.

**Literatur:** *Randomized Algorithms* [Motwani+Raghavan, Cambridge U Press, '95]  
*Algorithmen und Zufall* [Vorlesungsskript, Jochen Geiger, Uni KL]

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit WK  $1 - O(1/\sqrt[4]{n})$  mit  $\frac{3}{2}n + o(n)$  Vgl.
- Die besten deterministischen Auswahl-Algorithmen (*sehr kompliziert!*) benötigen  $3n$  Vergleiche im schlechtesten Fall.
- Jeder deterministische Auswahl-Alg. 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  $n \geq 2100/\varepsilon + 70$  Zahlen die  $i$ -kleinste Zahl mit höchstens  $(30 + \varepsilon)n$  Vergleichen finden kann.

**Literatur:** *Randomized Algorithms* [Motwani+Raghavan, Cambridge U Press, '95]  
*Algorithmen und Zufall* [Vorlesungsskript, Jochen Geiger, Uni KL]

- Der Algorithmus LazySelect [Floyd & Rivest, 1975] löst das Auswahlproblem mit WK  $1 - O(1/\sqrt[4]{n})$  mit  $\frac{3}{2}n + o(n)$  Vgl.
- Die besten deterministischen Auswahl-Algorithmen (*sehr kompliziert!*) benötigen  $3n$  Vergleiche im schlechtesten Fall.
- Jeder deterministische Auswahl-Alg. benötigt im schlechtesten Fall mindestens  $2n$  Vergleiche.

