Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG
Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität
Institut für Informatik

Algorithmen und Datenstrukturen

Wintersemester 2020/21
10. Vorlesung

Das Auswahlproblem

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

Analyse von Messreihen

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

Analyse von Messreihen

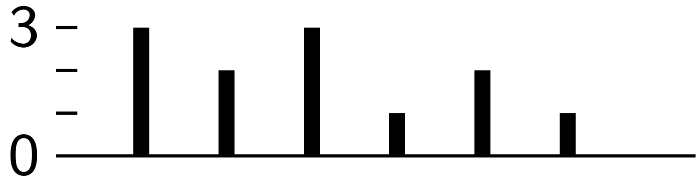
Problem: Gegeben eine Reihe von n Messwerten $A[1..n]$,
finde einen „guten" Mittelwert.

Beispiel:

Analyse von Messreihen

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

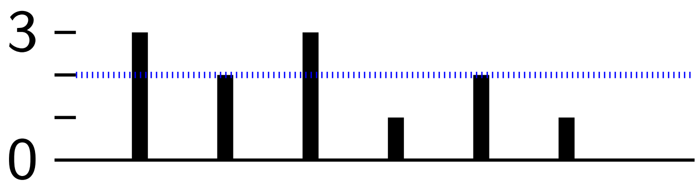
Beispiel:



Analyse von Messreihen

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

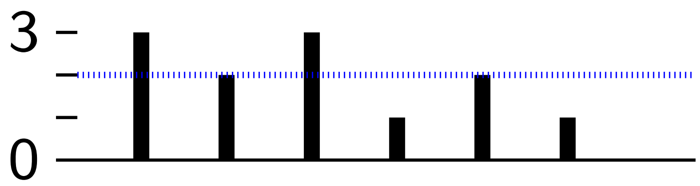
Beispiel:



Analyse von Messreihen

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

Beispiel:



arithmetisches Mittel $\frac{\sum_i A[i]}{n}$

Analyse von Messreihen

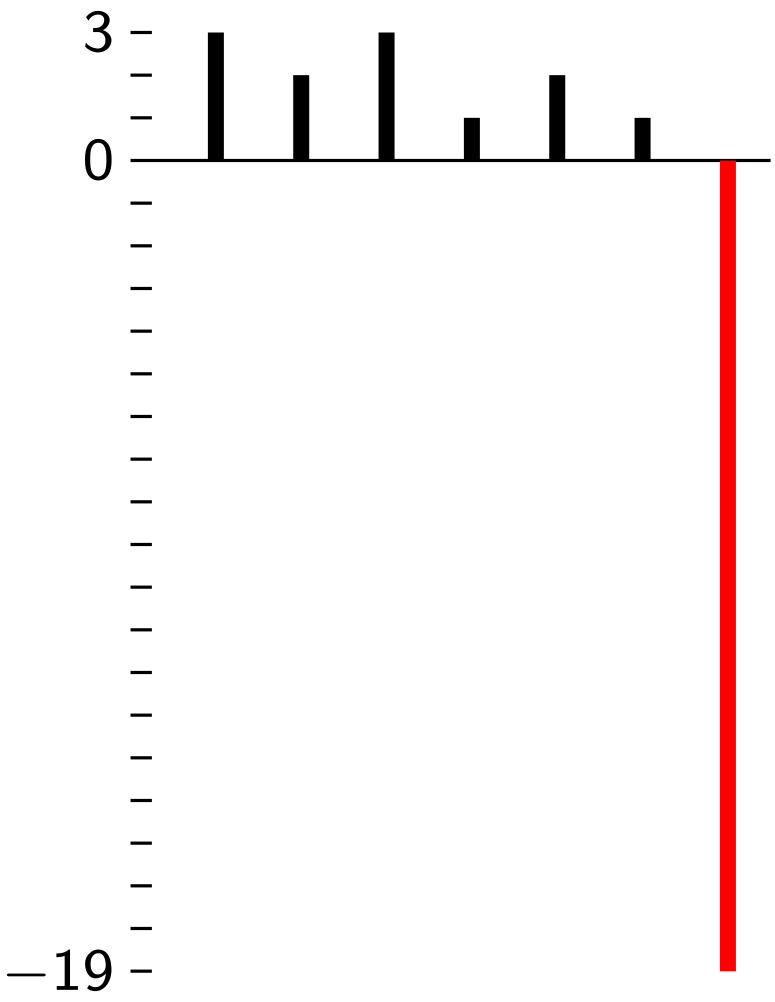
Problem: Gegeben eine Reihe von n Messwerten $A[1..n]$, finde einen „guten“ Mittelwert.

Beispiel:

Analyse von Messreihen

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

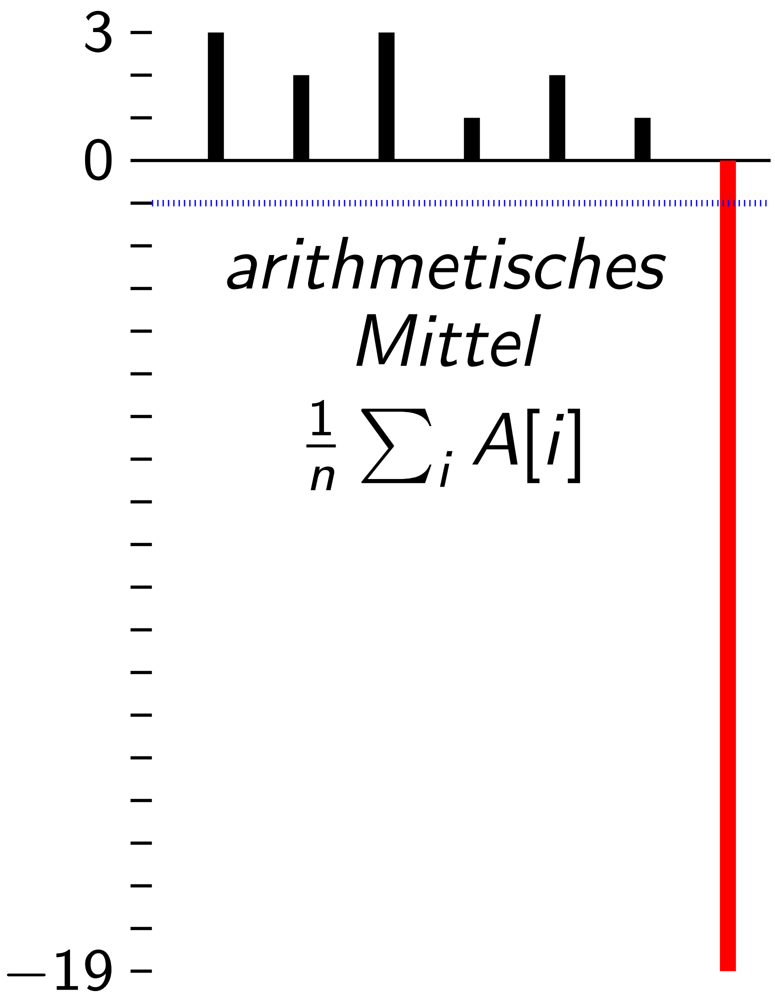
Beispiel:



Analyse von Messreihen

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

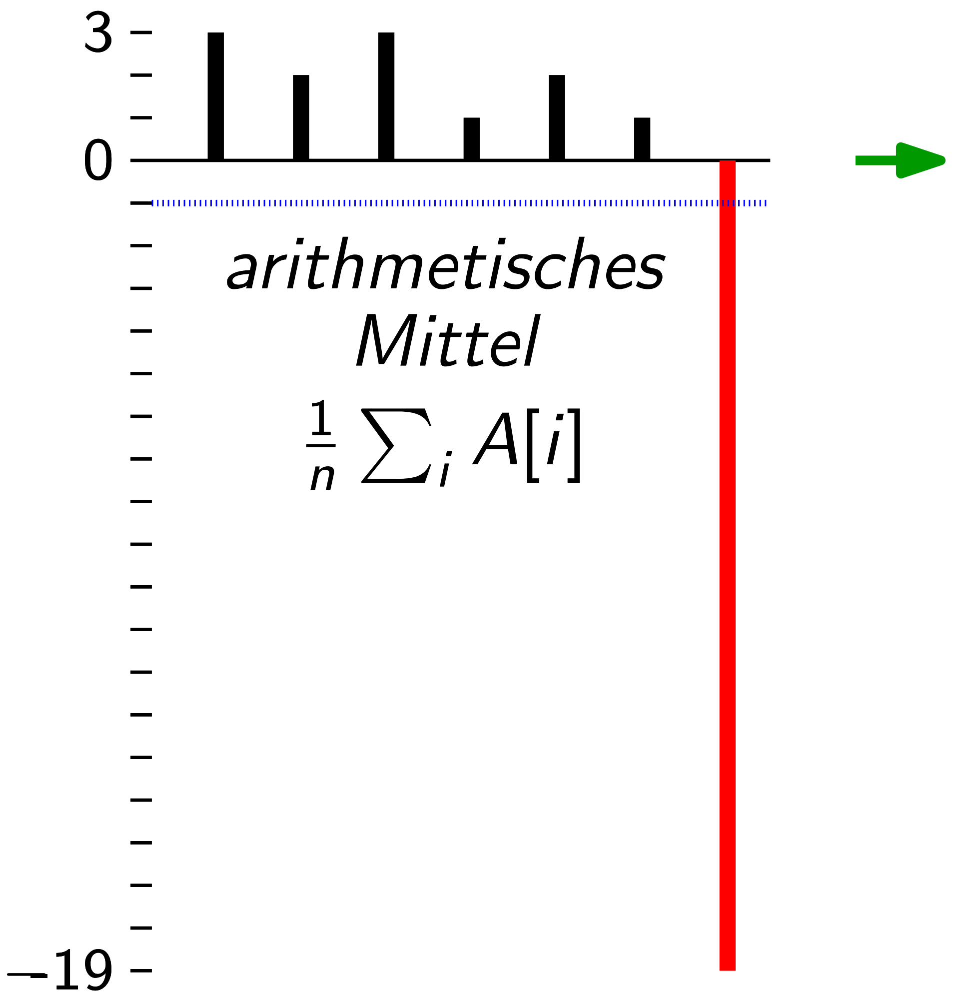
Beispiel:



Analyse von Messreihen

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

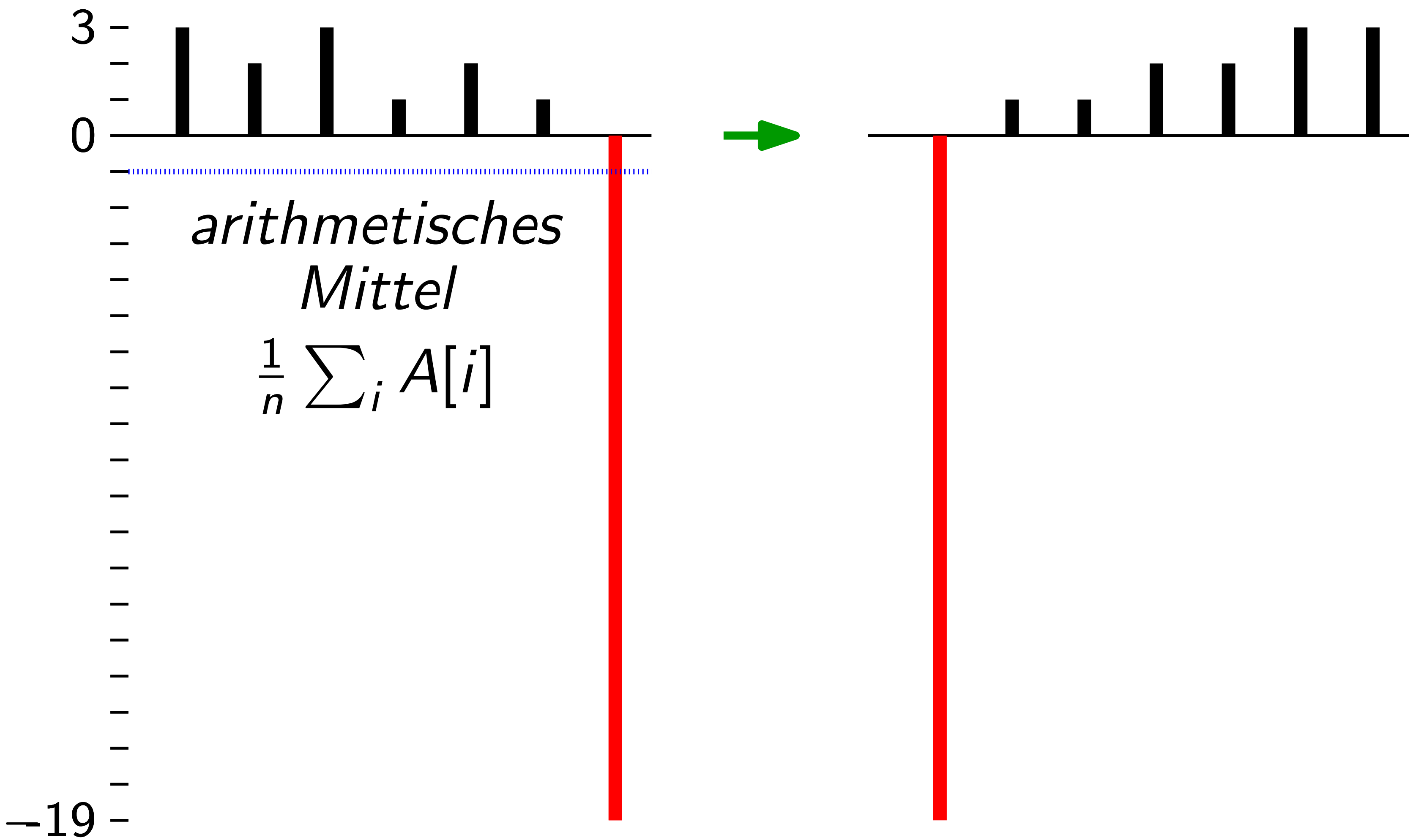
Beispiel:



Analyse von Messreihen

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

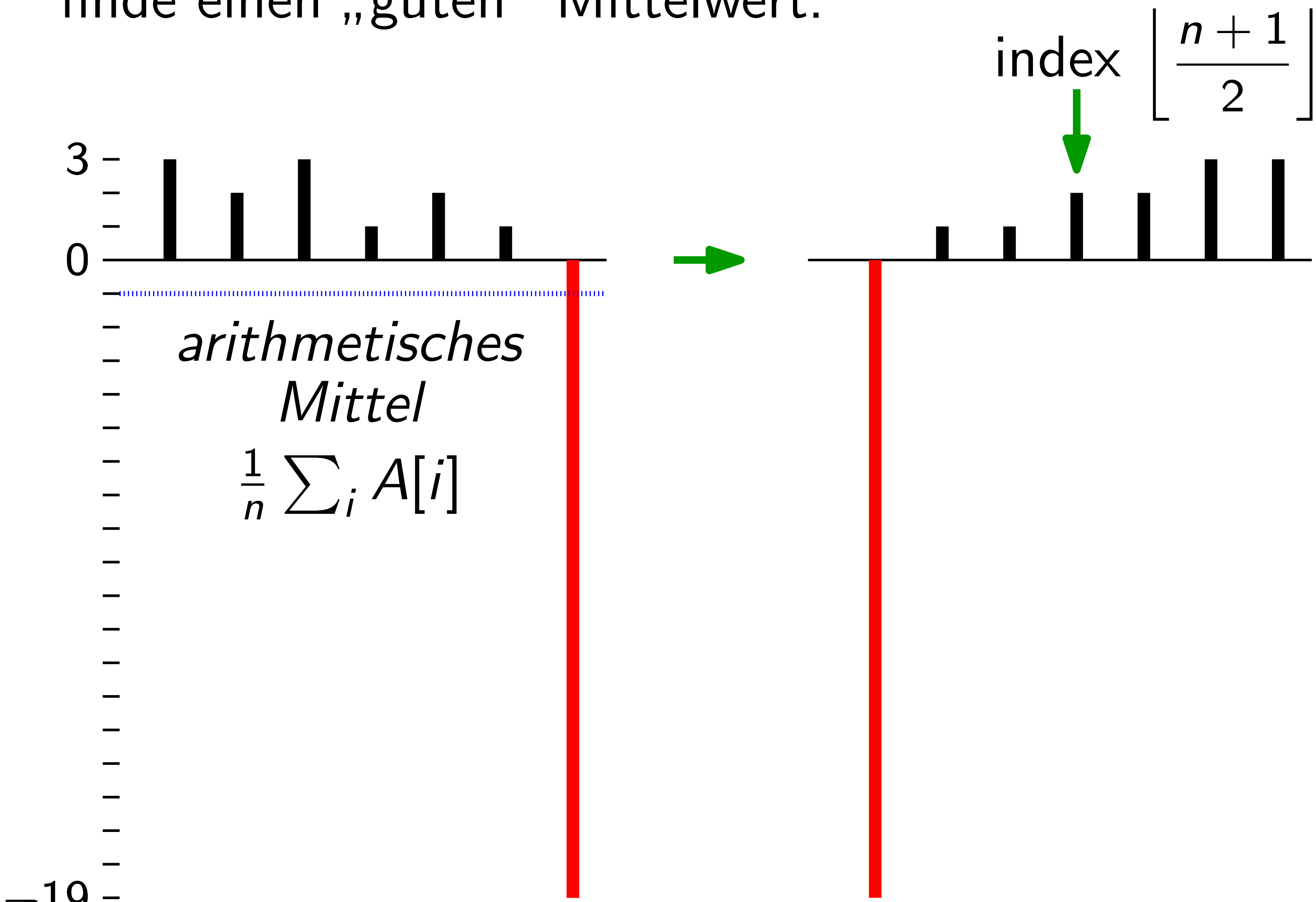
Beispiel:



Analyse von Messreihen

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

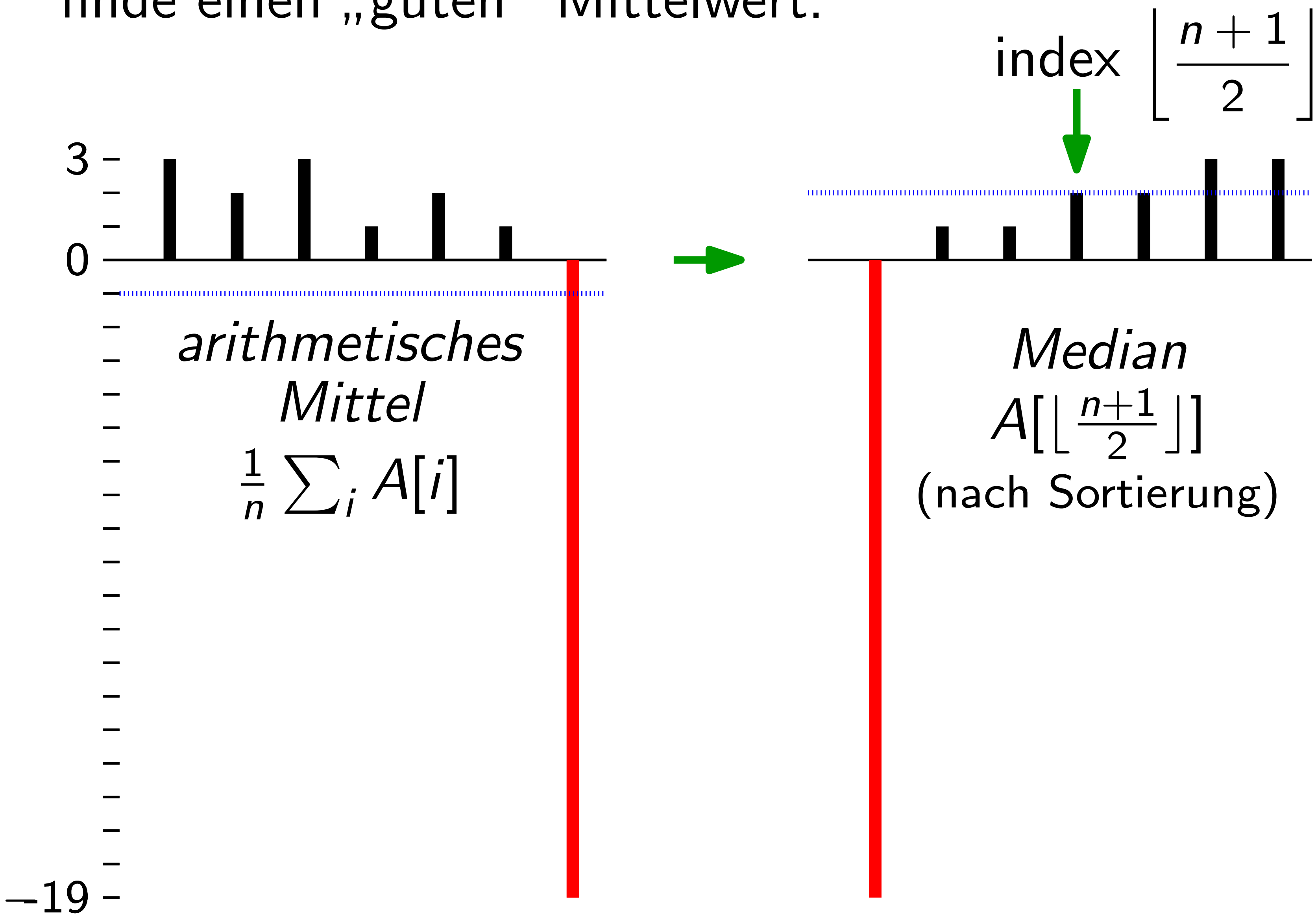
Beispiel:



Analyse von Messreihen

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

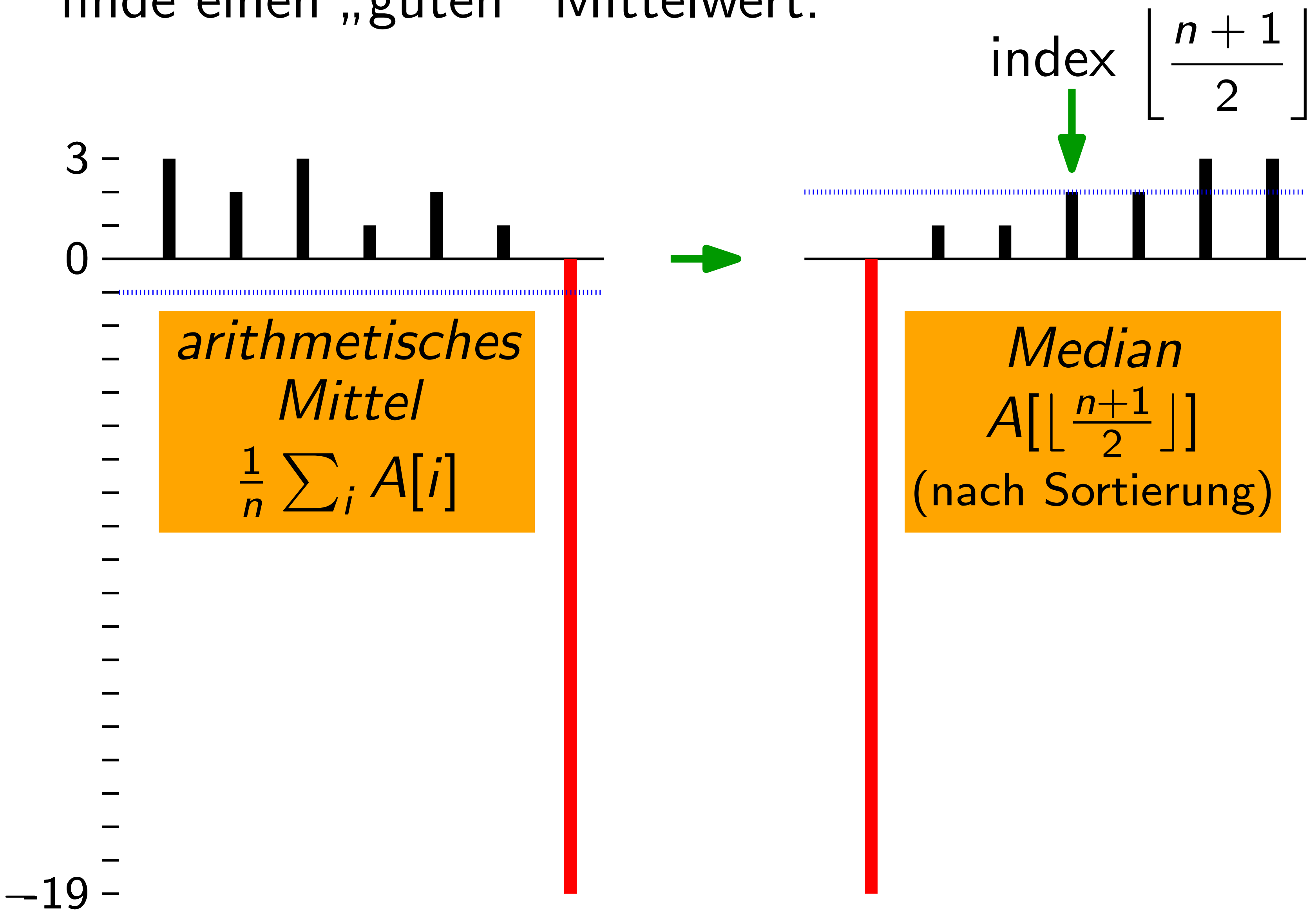
Beispiel:



Analyse von Messreihen

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

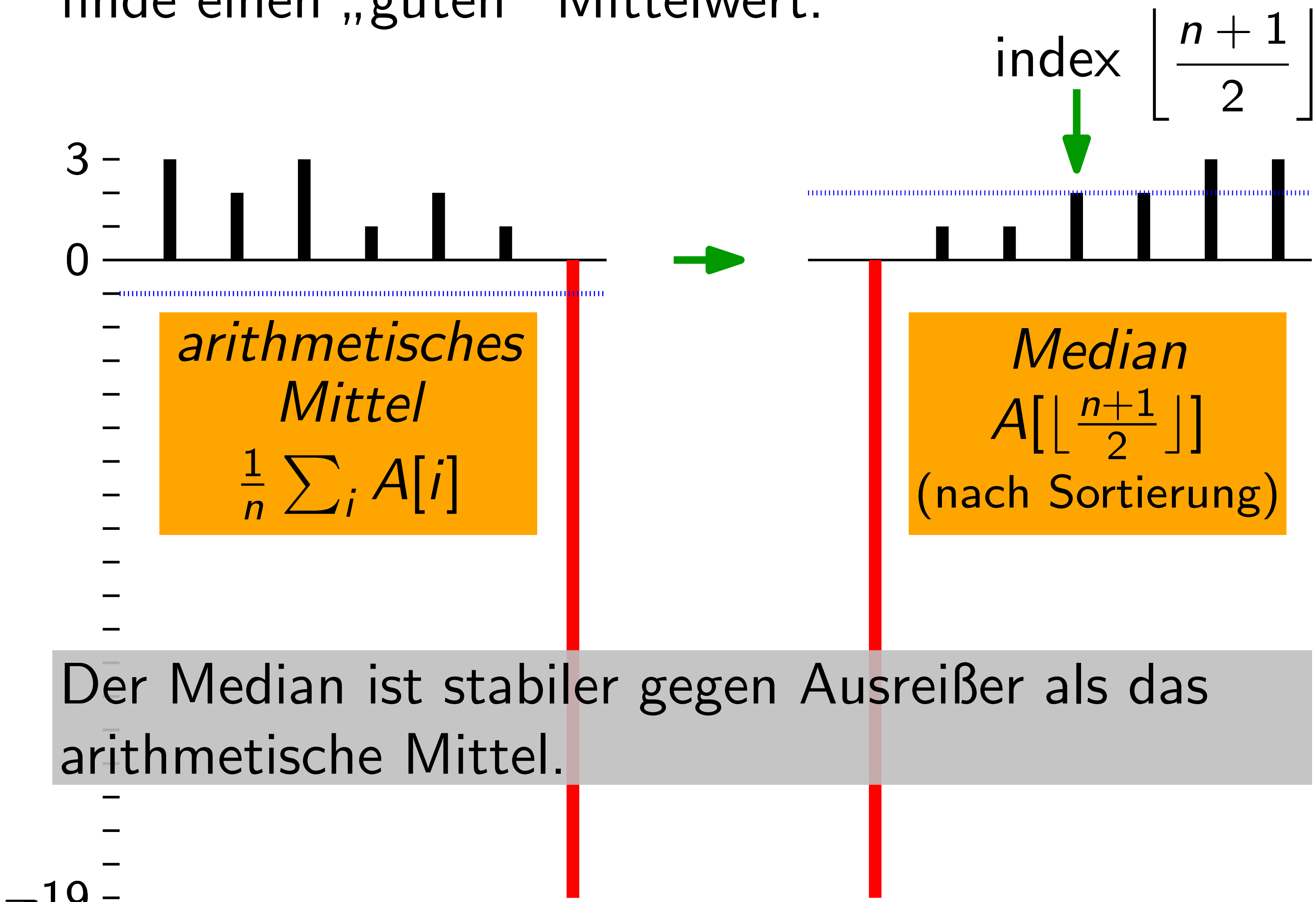
Beispiel:



Analyse von Messreihen

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

Beispiel:

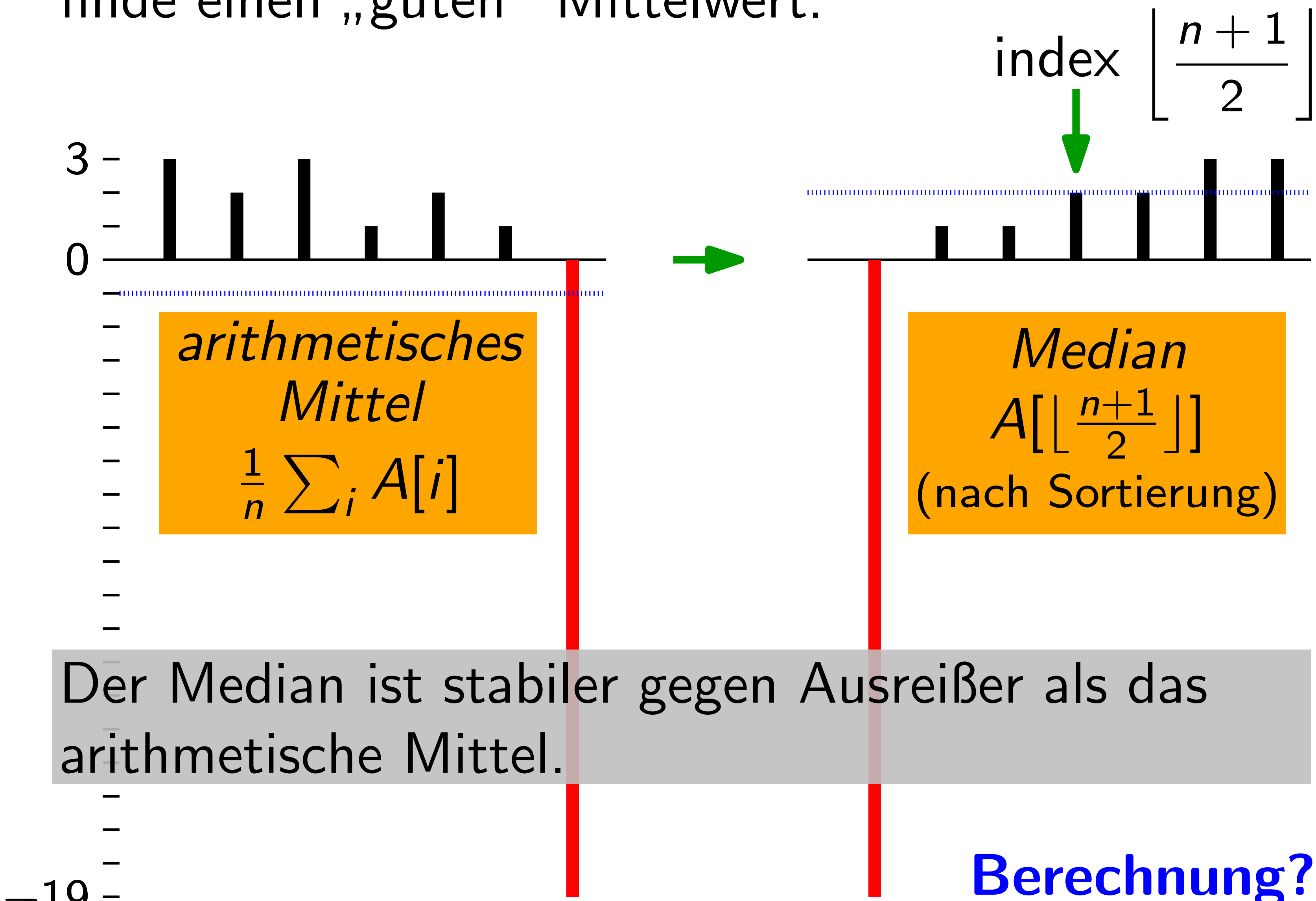


Beob.: Der Median ist stabiler gegen Ausreißer als das arithmetische Mittel.

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.

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
    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

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

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*?

Spezialfälle

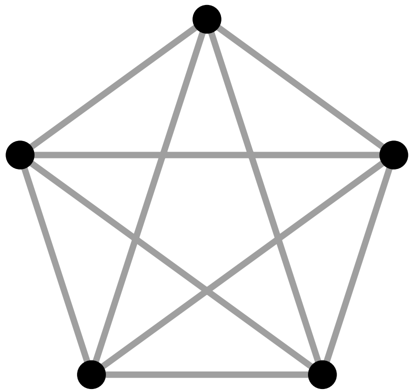
$i = \lfloor \frac{n+1}{2} \rfloor$: Median

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

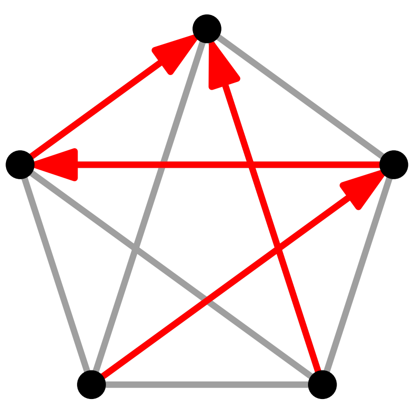
$i = 1$: Minimum
$i = n$: Maximum

$\left.\begin{matrix} i=1: \text{ Minimum} \\ i=n: \text{ Maximum} \end{matrix}\right\}$ 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

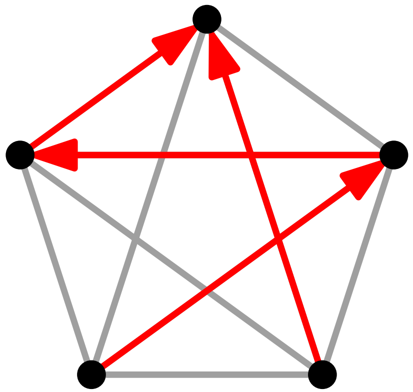
$i = \lfloor \frac{n+1}{2} \rfloor$: Median

$i = 1$: 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

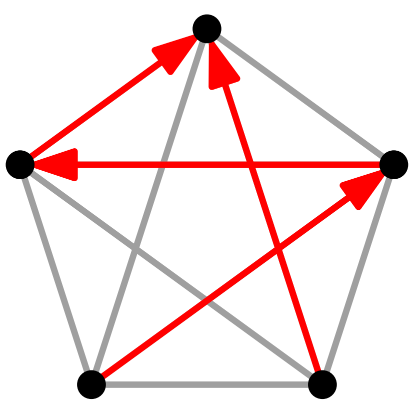
$i = \lfloor \frac{n+1}{2} \rfloor$: Median ← *Geht das auch in linearer Zeit??*

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

Also sind $n - 1$ Vergleiche optimal.

Spezialfälle

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

$i = 1$: Minimum

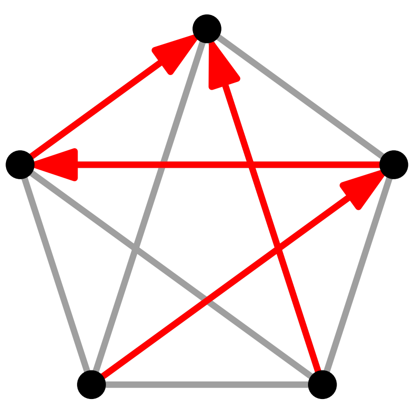
$i = n$: Maximum

$\left.\begin{array}{l}\text{Minimum}\\ \text{Maximum}\end{array}\right\}$ 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

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

Zuerst zur zweiten Frage

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

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

Zuerst zur zweiten Frage

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

Klar: $V_{\mathrm{minmax}}(n) \leq 2 \cdot V_{\mathrm{min}}(n) =$

Zuerst zur zweiten Frage

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

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

Zuerst zur zweiten Frage

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

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

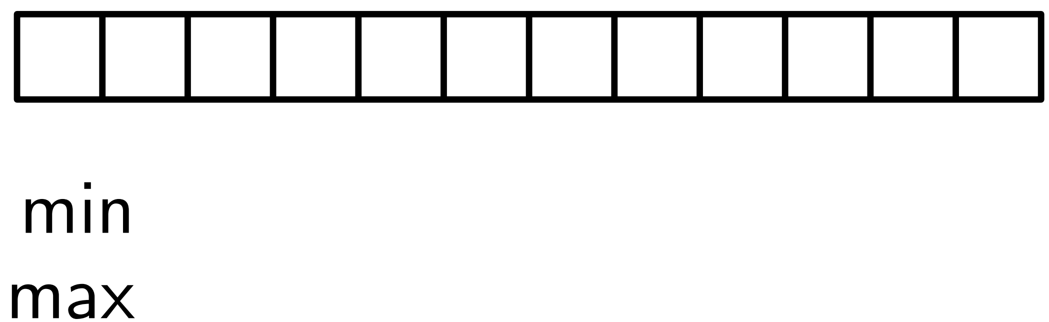
Frage: Geht es auch mit weniger Vergleichen?

Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

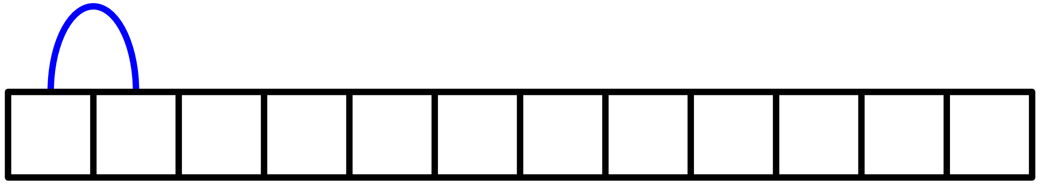


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?



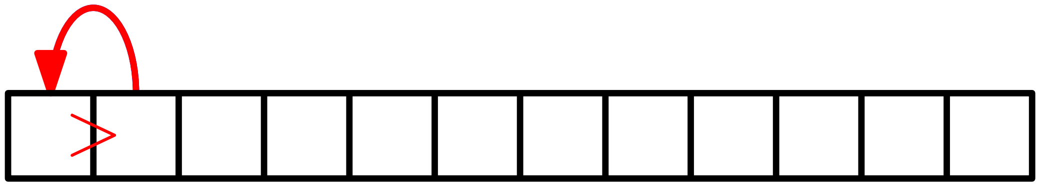
min
max

Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?



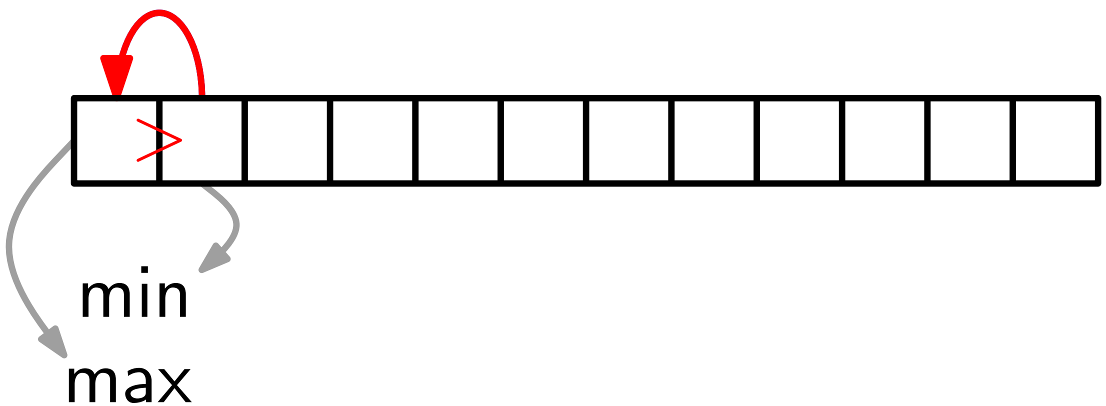
min
max

Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

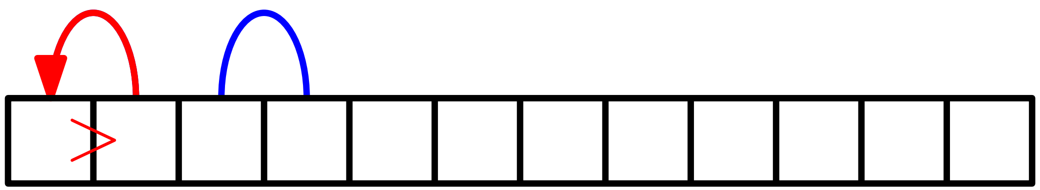


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?



min

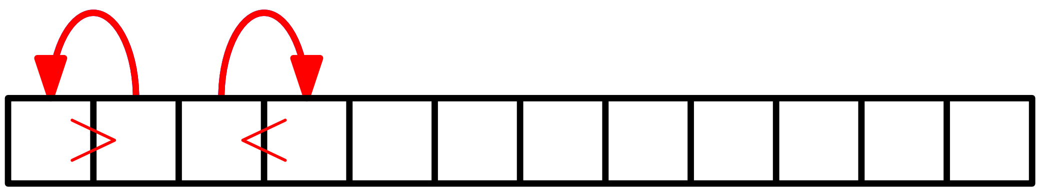
max

Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?



min

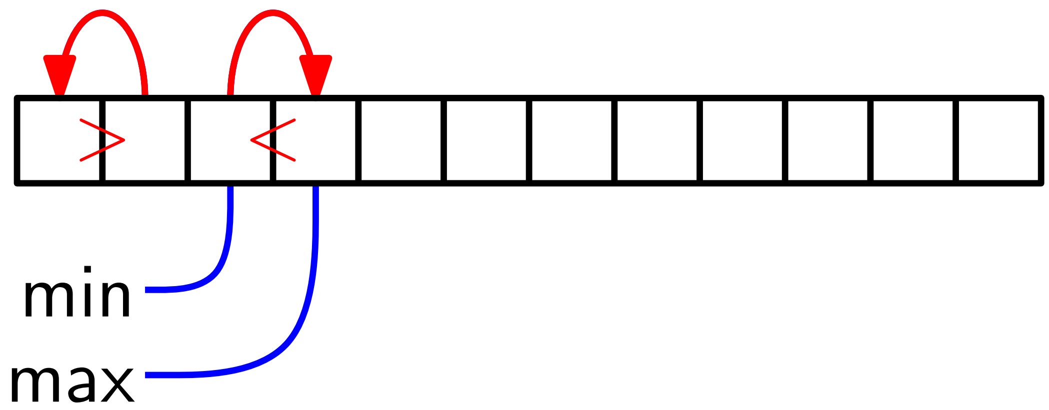
max

Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

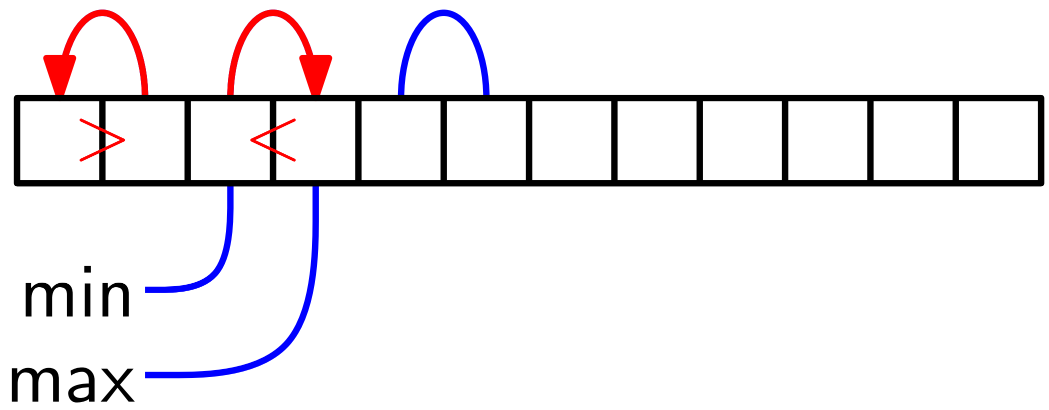


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

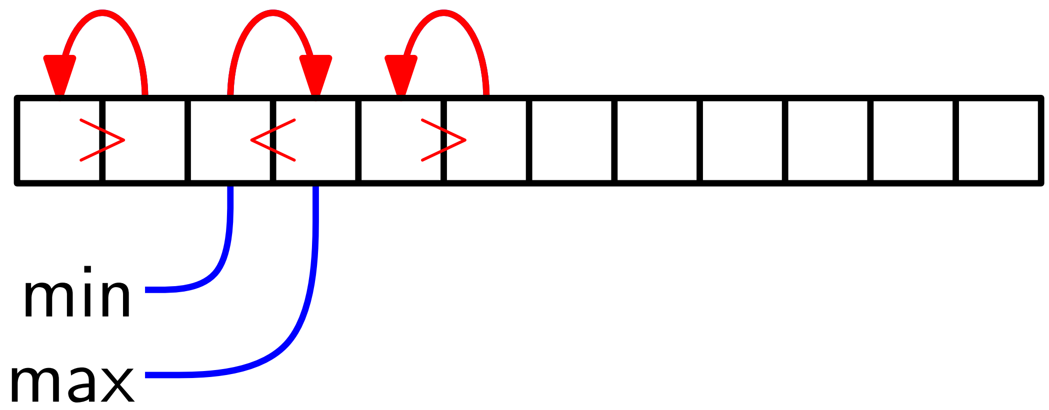


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

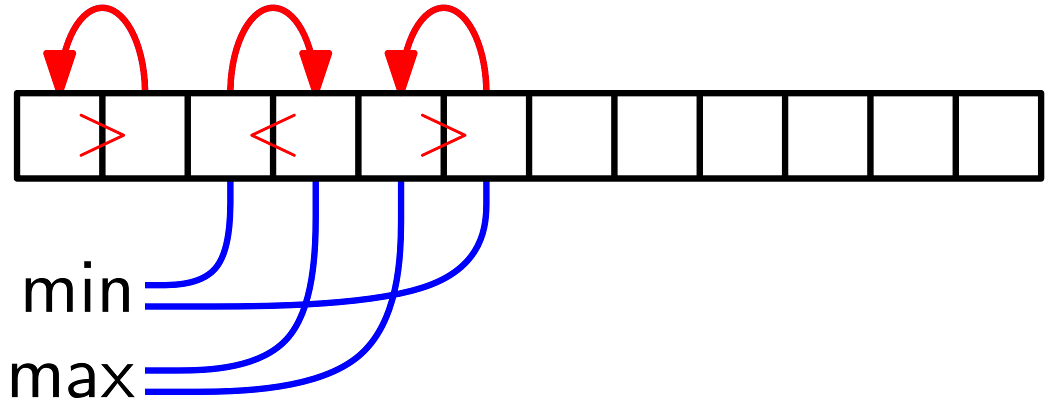


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

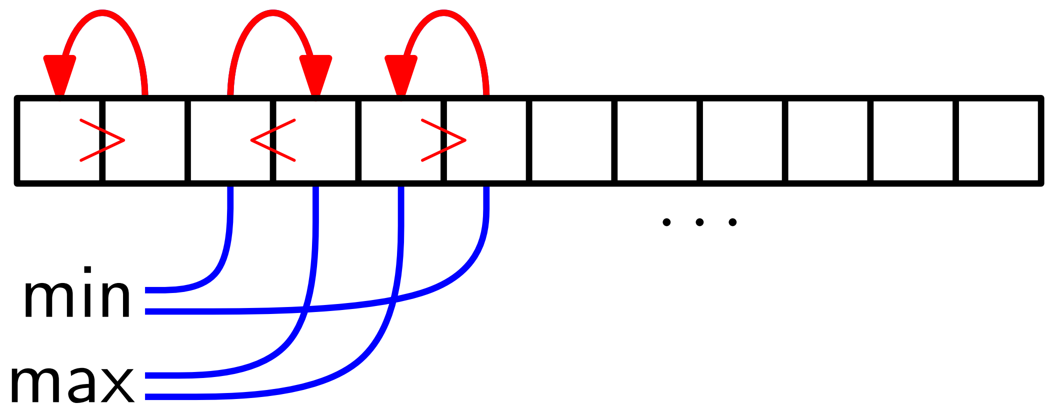


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

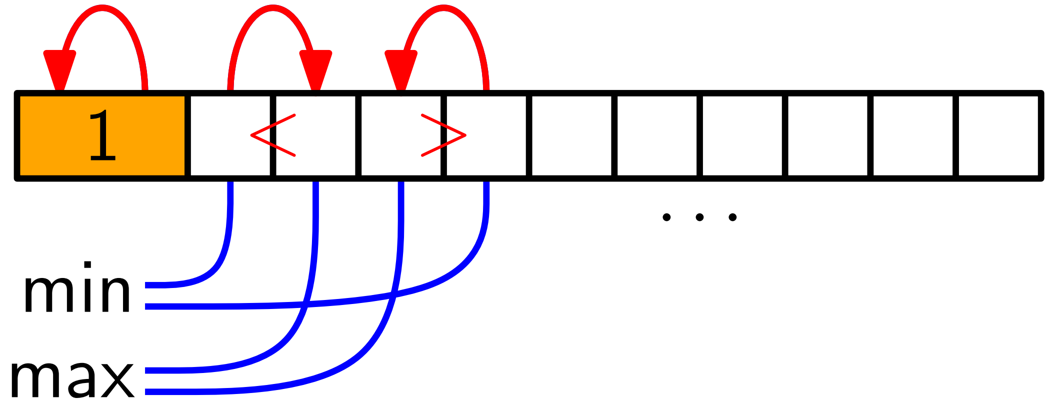


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

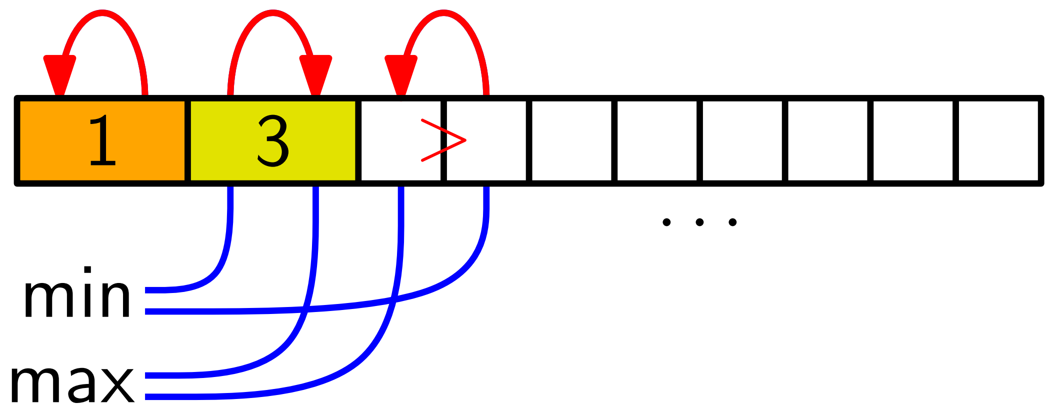


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

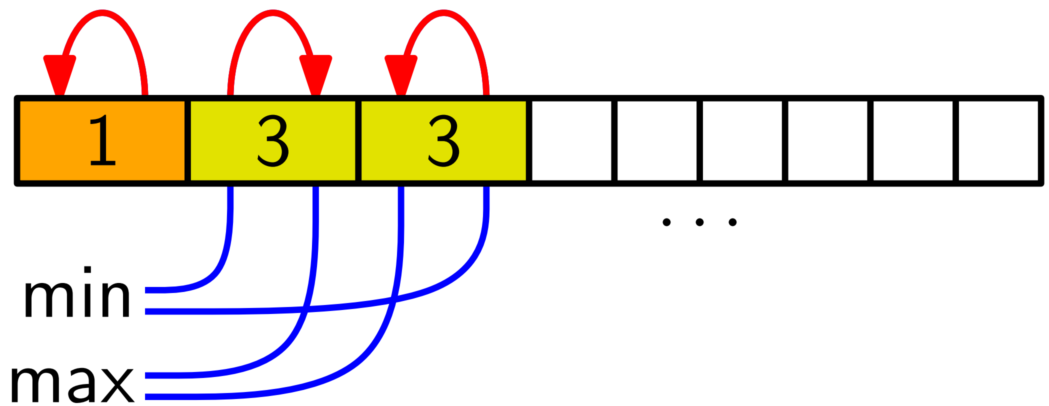


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

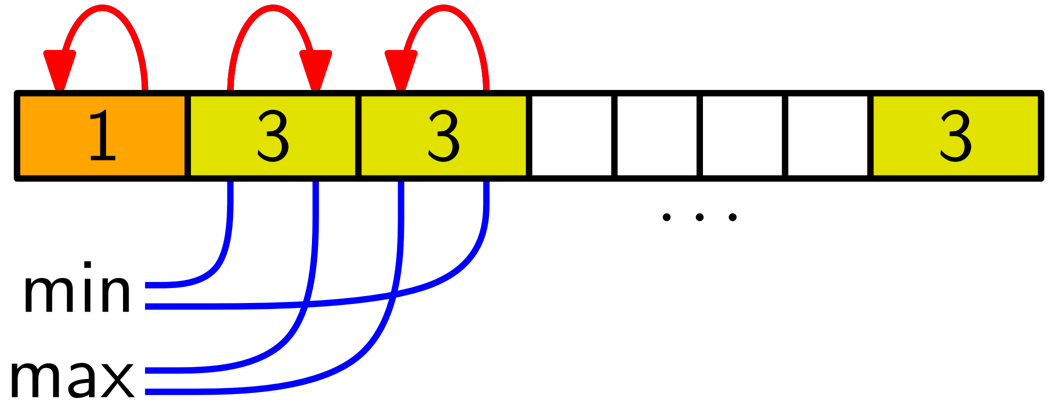


Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen?

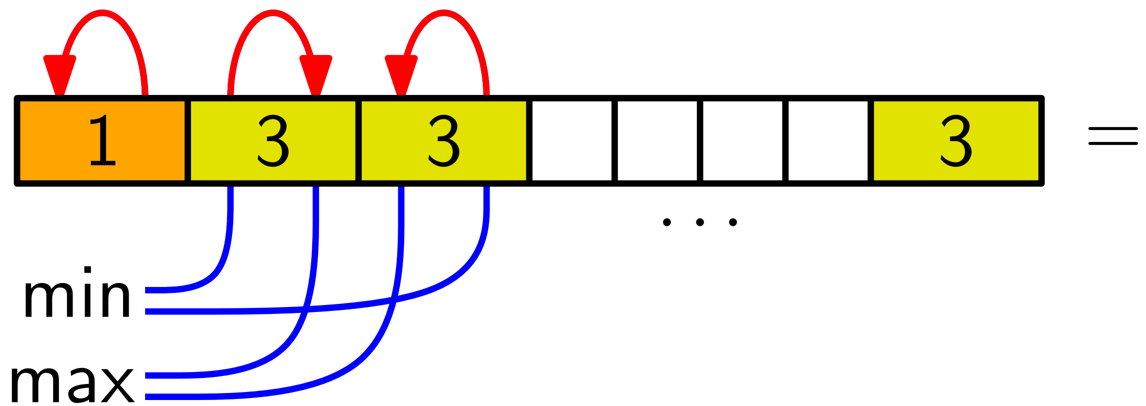


Zuerst zur zweiten Frage

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

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

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

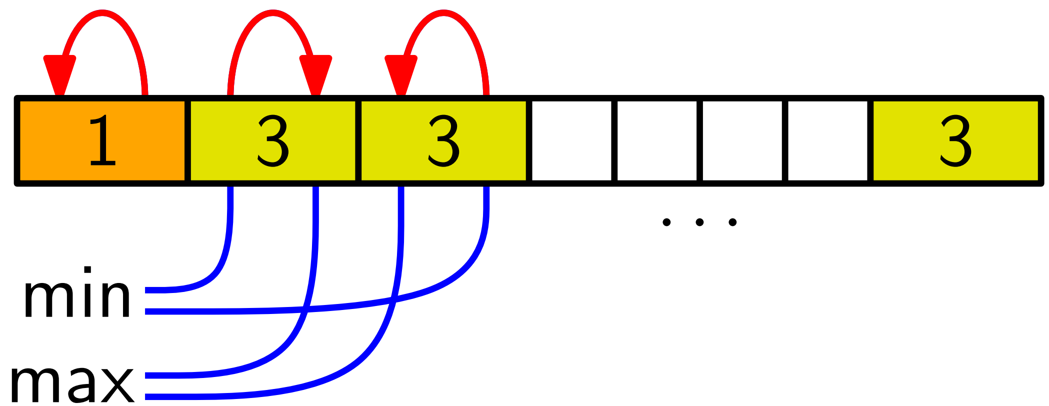


Zuerst zur zweiten Frage

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

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

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



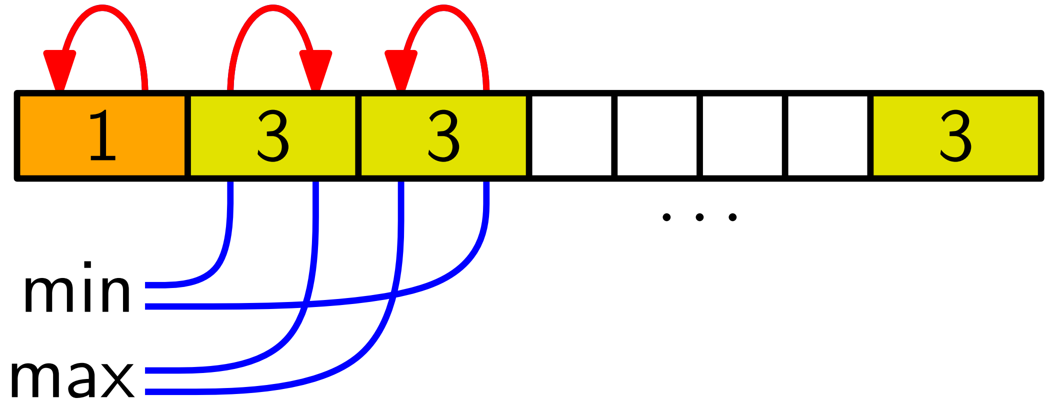
$= 1 \cdot 1 + (n/2 - 1) \cdot 3$

Zuerst zur zweiten Frage

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

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

Frage: Geht es auch mit weniger Vergleichen? $(n \text{ gerade})$



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

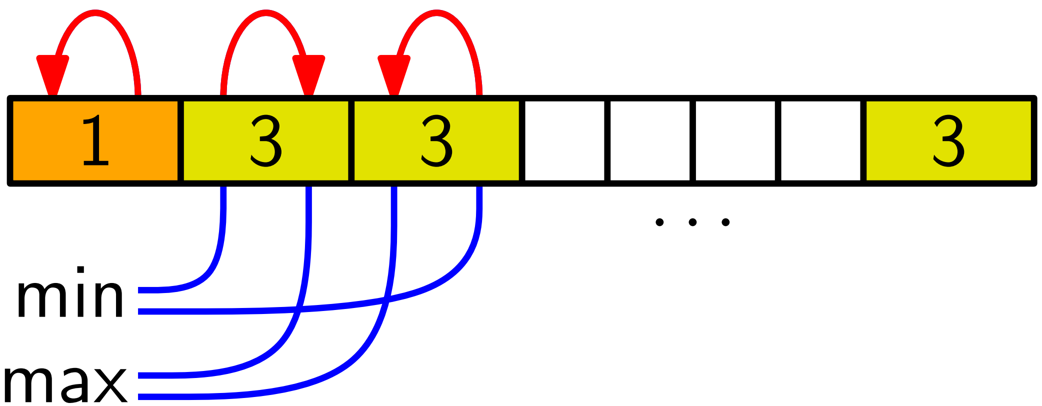
$= 3n/2 - 2$

Zuerst zur zweiten Frage

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

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

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



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

$= \boxed{3n/2 - 2}$

Ist das *optimal*?

Auswahl per Teile & Herrsche

Zur Erinnerung...

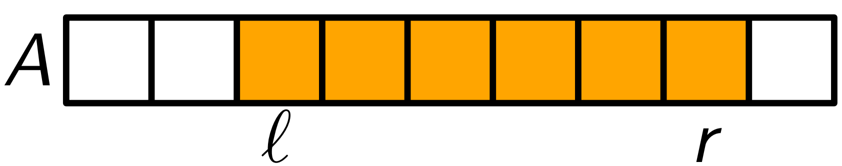
QuickSort(int[] A, int ℓ, r)

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

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

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

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



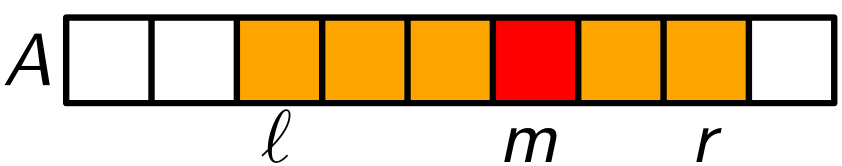
Auswahl per Teile & Herrsche

Zur Erinnerung...

QuickSort(int[] A, int ℓ, 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...

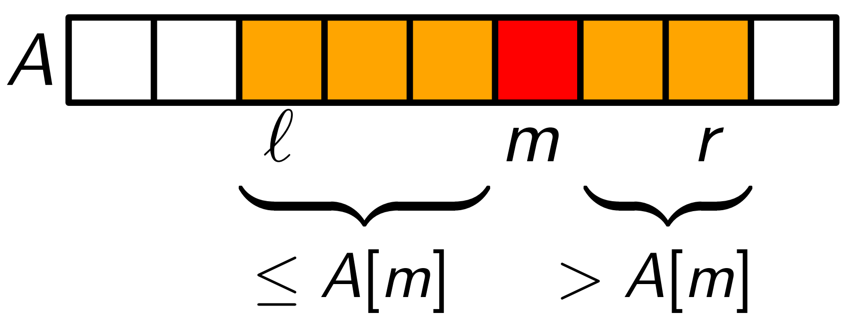
QuickSort(int[] A, int ℓ, r)

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...

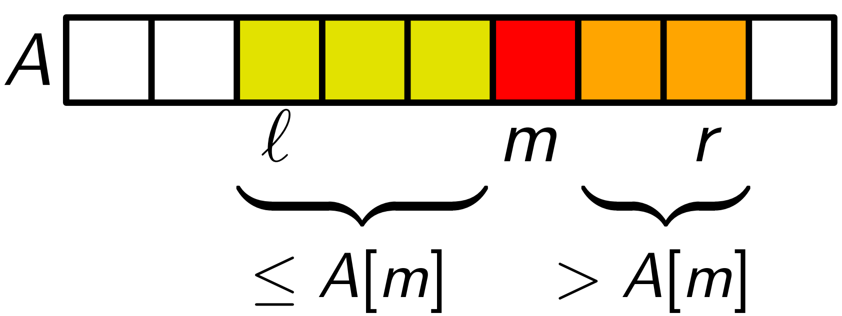
QuickSort(int[] A, int ℓ, r)

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

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

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

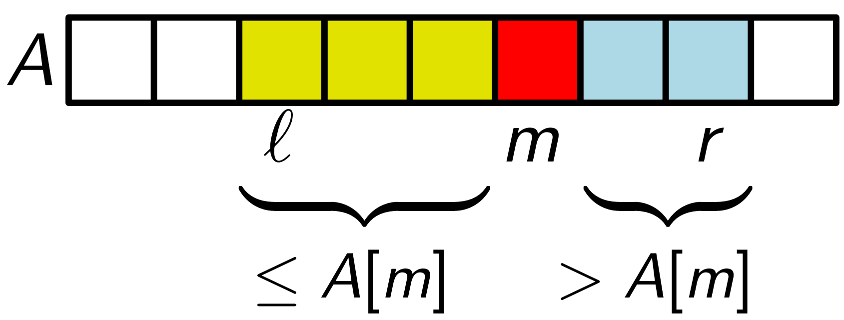
  QuickSort($A, m+1, r$)



Auswahl per Teile & Herrsche

Zur Erinnerung...

QuickSort(int[] A, int ℓ, 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...

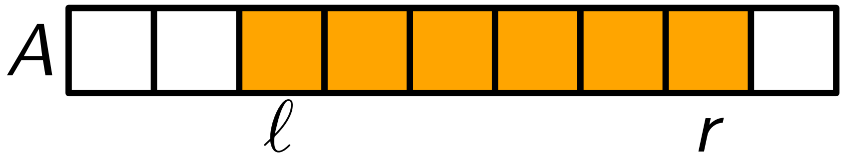
QuickSort(int[] A, int ℓ, r)

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

  $m =$ Partition(A, ℓ, r)

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

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



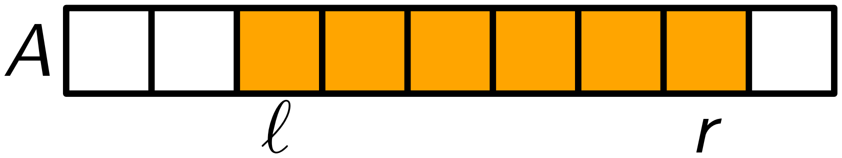
Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int ℓ, r)

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

Randomized
$m =$ Partition(A, ℓ, r)
QuickSort$(A, \ell, m-1)$
QuickSort$(A, m+1, r)$



Auswahl per Teile & Herrsche

Zur Erinnerung...

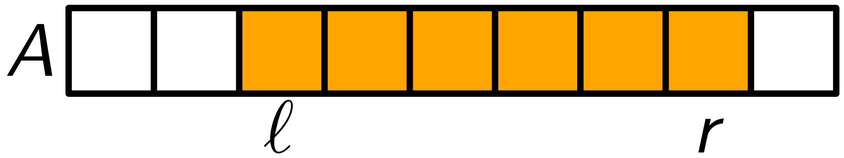
Randomized
QuickSort(int[] A, int ℓ, r)

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

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

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

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



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

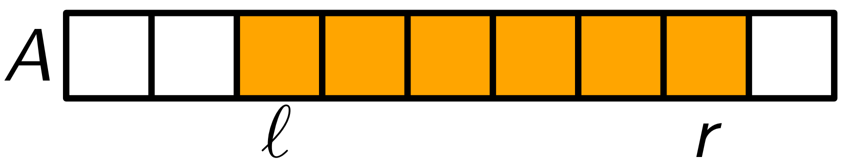
Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int $\boxed{\ell, r}$)

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

          Randomized
    $m =$ Partition(A, ℓ, r)
    QuickSort($A, \ell, m - 1$)
    QuickSort($A, m + 1, r$)



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

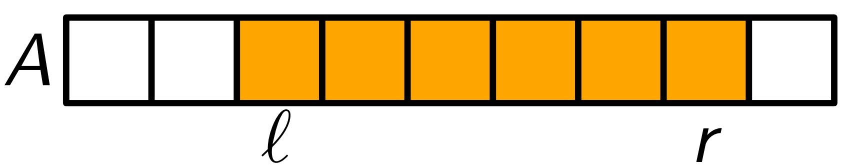
RandomizedSelect(int[] A, int ℓ, r, i)

Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int ℓ, r)

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



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

RandomizedSelect(int[] A, int ℓ, r, i)

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

Auswahl per Teile & Herrsche

Zur Erinnerung...

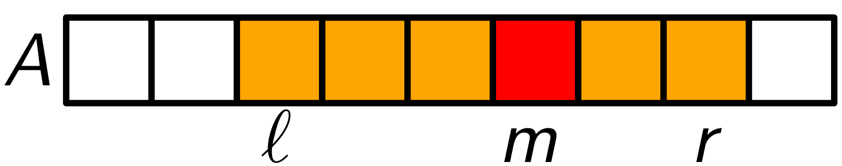
Randomized
QuickSort(int[] A, int ℓ, r)

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

  m = Randomized Partition(A, ℓ, r)

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

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



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

RandomizedSelect(int[] A, int ℓ, r, i)

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

m = RandomizedPartition(A, ℓ, r)

$k = m - \ell + 1$

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

  return

else

  if $i < k$ **then**

    return

  else

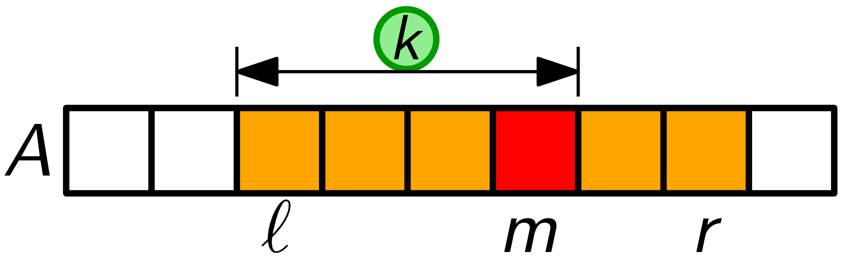
    return

Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int ℓ, r)

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



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

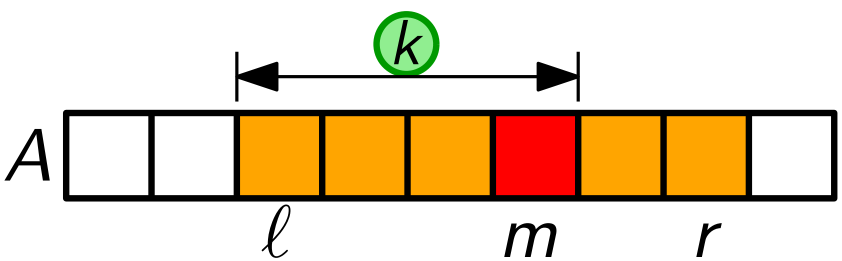
RandomizedSelect(int[] A, int ℓ, r, i)

if $\ell == r$ **then return** $A[\ell]$
m = RandomizedPartition(A, ℓ, 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 ℓ, r)
if $\ell < r$ **then**
  m = Randomized Partition(A, ℓ, r)
  QuickSort($A, \ell, m - 1$)
  QuickSort($A, m + 1, r$)



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

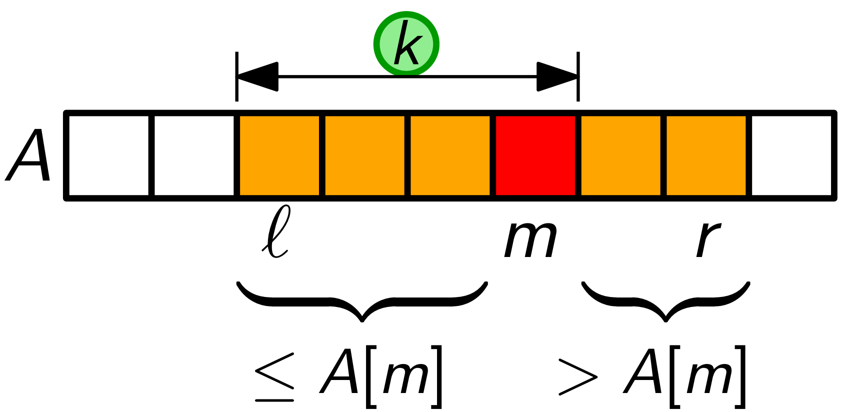
RandomizedSelect(int[] A, int ℓ, r, i)
if $\ell == r$ **then return** $A[\ell]$
m = RandomizedPartition(A, ℓ, r)
$k = m - \ell + 1$ *// Rang v. $A[m]$ in $A[\ell..r]$*
if $i == k$ **then**
  return
else
  if $i < k$ **then**
    return
  else
    return

Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int ℓ, r)

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



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

RandomizedSelect(int[] A, int ℓ, r, i)

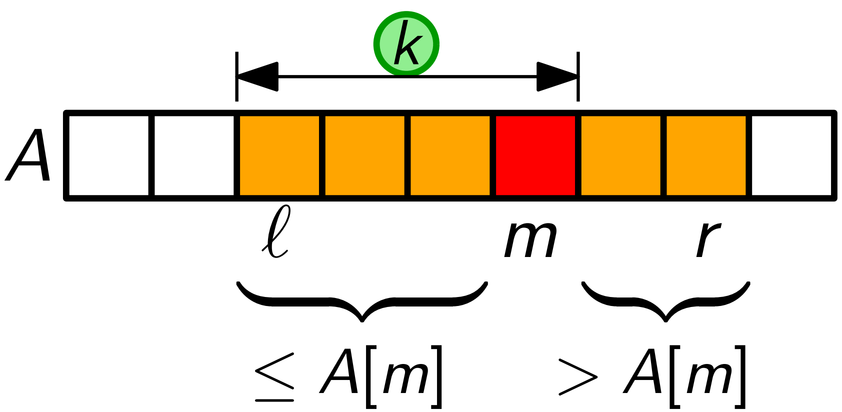
if $\ell == r$ **then return** $A[\ell]$
$m =$ RandomizedPartition(A, ℓ, r)
$k = m - \ell + 1$ *// Rang v. $A[m]$ in $A[\ell..r]$*
if $i == k$ **then**
  return
else
  if $i < k$ **then**
    return
  else
    return

Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int ℓ, r)

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



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

RandomizedSelect(int[] A, int ℓ, r, i)

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

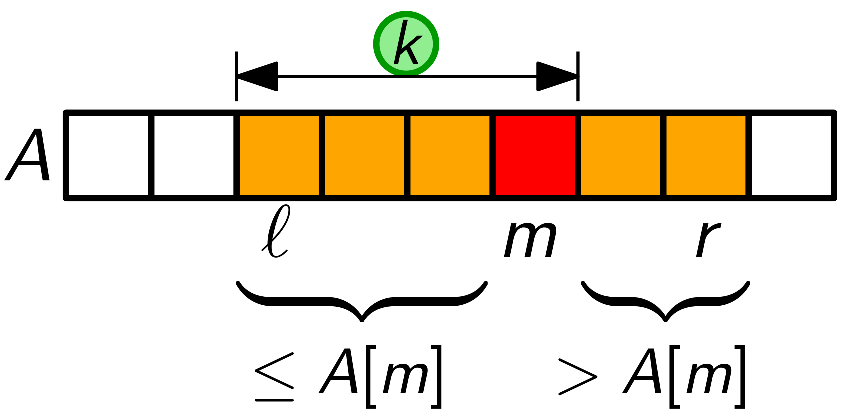
else
- **if** $i < k$ **then**
  - **return**
- **else**
  - **return**

Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int ℓ, r)

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



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

RandomizedSelect(int[] A, int ℓ, r, i)

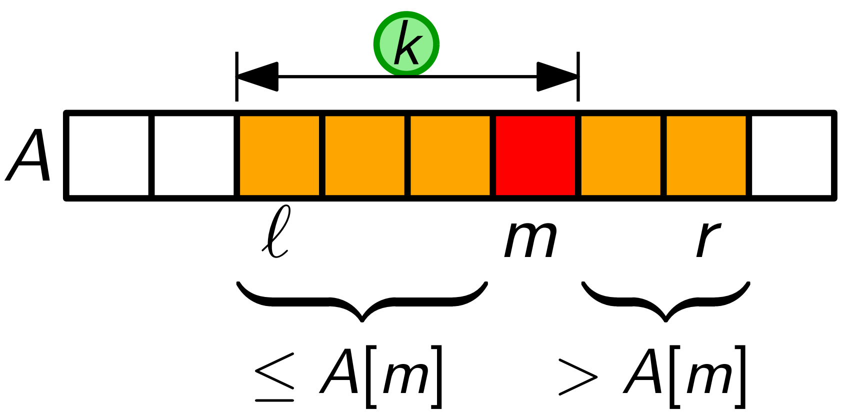
if $\ell == r$ **then return** $A[\ell]$
m = RandomizedPartition(A, ℓ, r)
$k = m - \ell + 1$ *// Rang v. $A[m]$ in $A[\ell..r]$*
if $i == k$ **then**
  return $A[m]$
else
  if $i < k$ **then**
    return RSelect($A, \ell, m-1, i$)
  else
    return

Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int ℓ, r)

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



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

RandomizedSelect(int[] A, int ℓ, r, i)

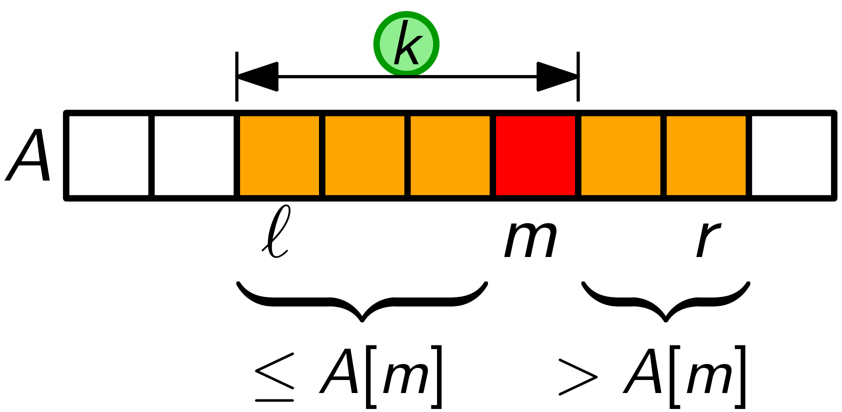
if $\ell == r$ **then return** $A[\ell]$
m = RandomizedPartition(A, ℓ, r)
$k = m - \ell + 1$ // *Rang* v. $A[m]$ in $A[\ell..r]$
if $i == k$ **then**
  return $A[m]$
else
  if $i < k$ **then**
    return RSelect($A, \ell, m-1, i$)
  else
    return RSelect($A, m+1, r, i-k$)

Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int ℓ, r)

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



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

RandomizedSelect(int[] A, int ℓ, r, i)

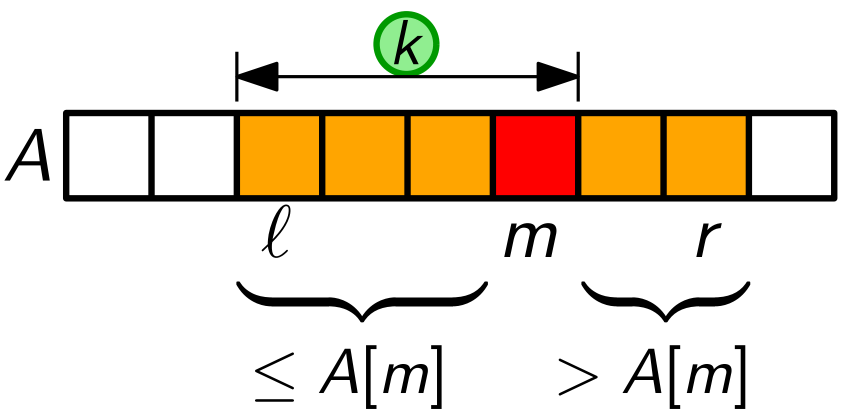
if $\ell == r$ **then return** $A[\ell]$
m = RandomizedPartition(A, ℓ, r)
$k = m - \ell + 1$ // *Rang* v. $A[m]$ in $A[\ell..r]$
if $i == k$ **then**
  return $A[m]$
else
  if $i < k$ **then**
    return RSelect($A, \ell, m-1, i$)
  else
    return RSelect($A, m+1, r, i-k$)

Auswahl per Teile & Herrsche

Zur Erinnerung...

Randomized
QuickSort(int[] A, int ℓ, r)

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



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

RandomizedSelect(int[] A, int ℓ, r, i)

if $\ell == r$ **then return** $A[\ell]$
m = RandomizedPartition(A, ℓ, 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($A, \ell, m-1, \boxed{i}$)
- **else**
  - **return** RSelect($A, m+1, r, \boxed{i-k}$)

Ist Ihnen klar warum?

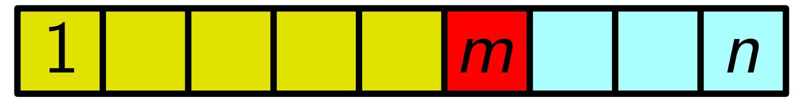
Laufzeit-Analyse

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.

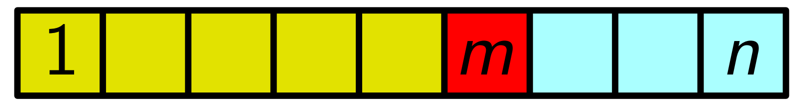
Trick: Geh davon aus, dass das gesuchte i. Element immer im *größeren* Teilfeld liegt.



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.



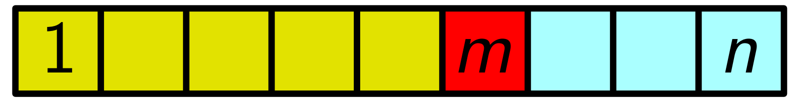
$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist

- 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.



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

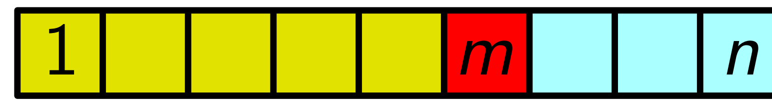
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von i

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

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.



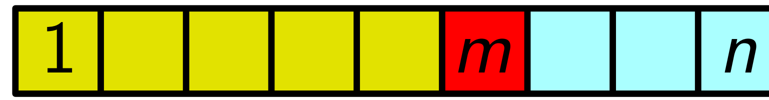
$\Rightarrow$ resultierende Zufallsvariable $V(n)$ ist
– obere Schranke für tatsächliche Anzahl von Vergleichen
– unabhängig von i

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

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.



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

– obere Schranke für tatsächliche Anzahl von Vergleichen

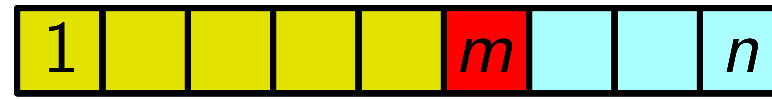
– unabhängig von i

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

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.



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

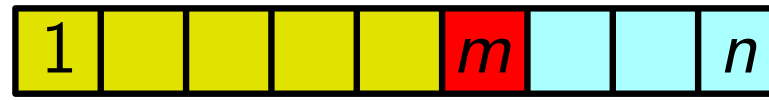
- obere Schranke für tatsächliche Anzahl von Vergleichen
- unabhängig von i

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

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.



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

– obere Schranke für tatsächliche Anzahl von Vergleichen

– unabhängig von i

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

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.

1					m			n

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

– obere Schranke für tatsächliche Anzahl von Vergleichen

– unabhängig von i

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

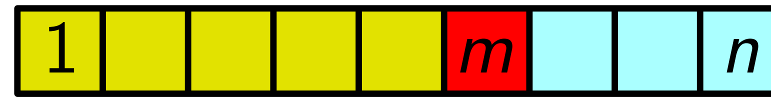
Alle Fälle gleich wahrscheinlich!

[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.



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

– obere Schranke für tatsächliche Anzahl von Vergleichen

– unabhängig von i

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

Alle Fälle gleich wahrscheinlich!

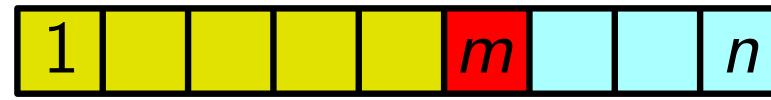
[vorausgesetzt alle Elem. sind verschieden!]

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

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.



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

– obere Schranke für tatsächliche Anzahl von Vergleichen
– unabhängig von i

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

Alle Fälle gleich wahrscheinlich!

[vorausgesetzt alle Elem. sind verschieden!]

$$\Rightarrow \boxed{E[V(n)] \leq n-1 + 2 \cdot \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[V(k)]} \overset{?}{\leq} c \cdot n \quad \binom{\text{für ein}}{c > 0}$$

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

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

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 $\lfloor \ldots \rfloor$.)

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 $f(n) \leq cn$.

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

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \le cn$.

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

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

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 $f(n) \leq cn$.

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

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

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 $f(n) \leq cn$.

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

Für jedes $\varepsilon > 0$ gilt:

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

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.

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 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, 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 = 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$'(A, \ell, r\cancel{)}, \mathit{pivot})$

 $\mathit{pivot} = A[r]$
 $i = \ell - 1$
 for $j = \ell$ **to** $r - 1$ **do**
  if $A[j] \leq \mathit{pivot}$ **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$'(A, \ell, r\cancel{)}, pivot)$

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

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$'(A, \ell, r\cancel{)}, pivot)$

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

$i = \ell - 1$

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

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

    $i = i + 1$

    Swap(A, i, j)

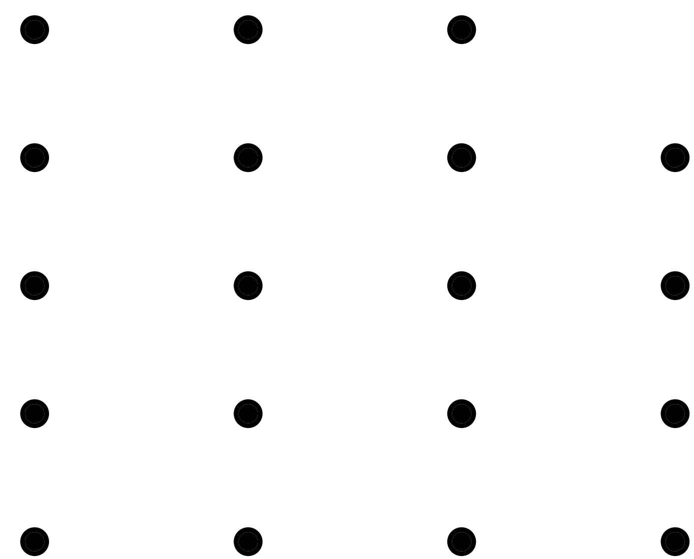
$\cancel{\text{Swap}(A, i + 1, r)}$

return $i \cancel{+ 1}$

Select: deterministisch

Select(A, ℓ, r, i)

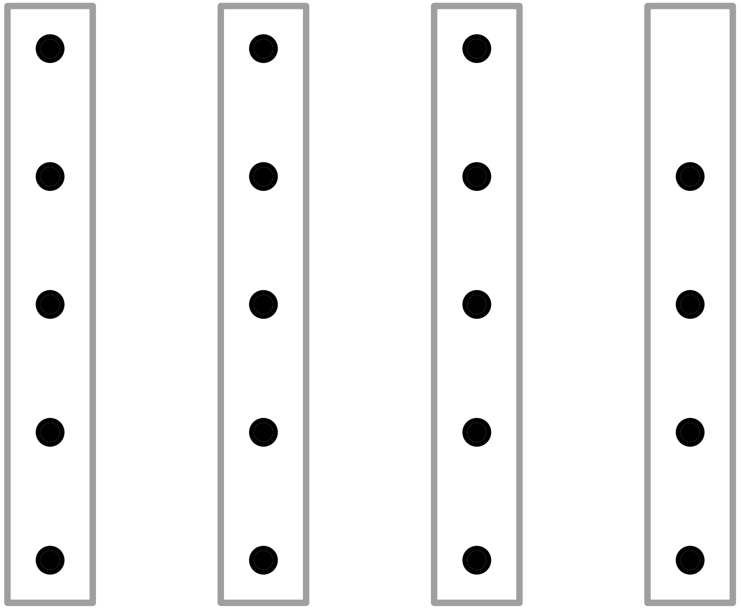
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.



Select: deterministisch

Select(A, ℓ, r, i)

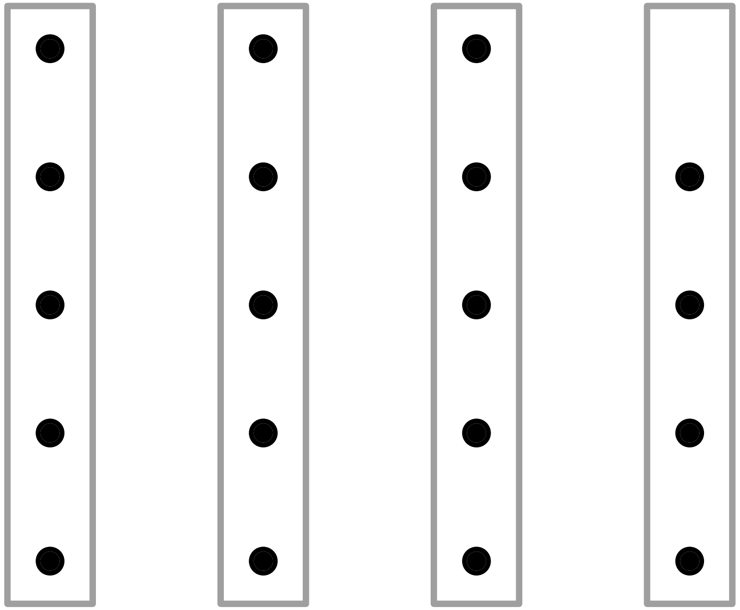
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.



Select: deterministisch

Select(A, ℓ, r, i)

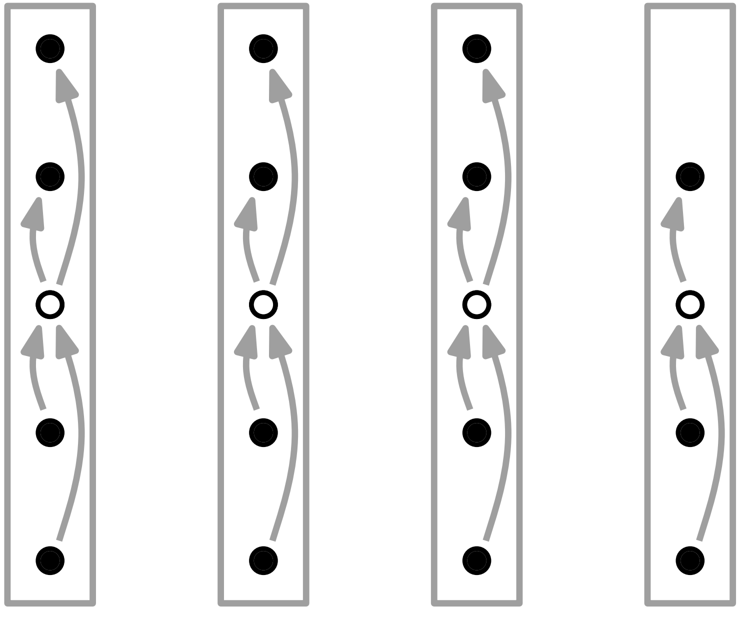
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n \bmod 5$) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.



Select: deterministisch

Select(A, ℓ, r, i)

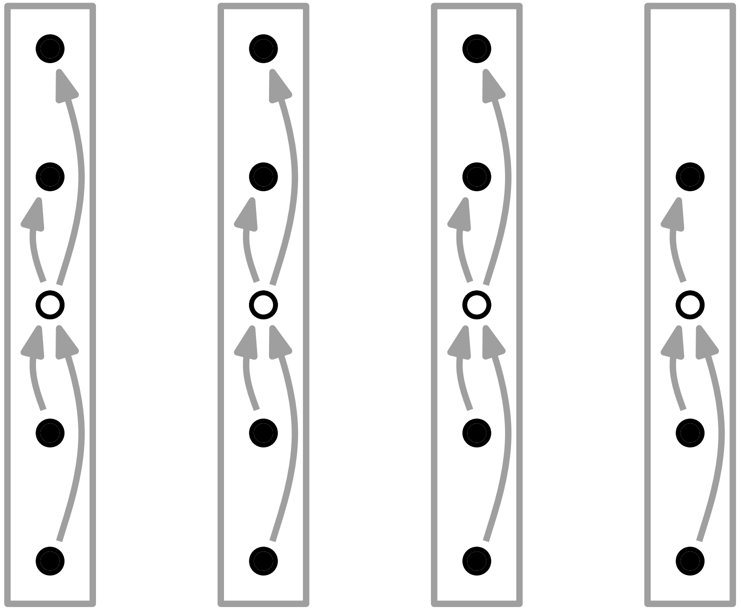
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen ($n \bmod 5$) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.



Select: deterministisch

Select(A, ℓ, r, i)

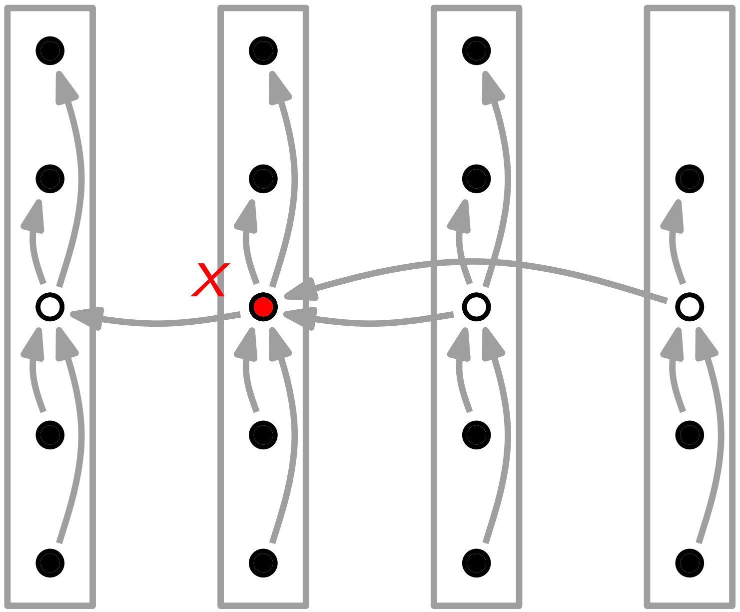
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.



Select: deterministisch

Select(A, ℓ, r, i)

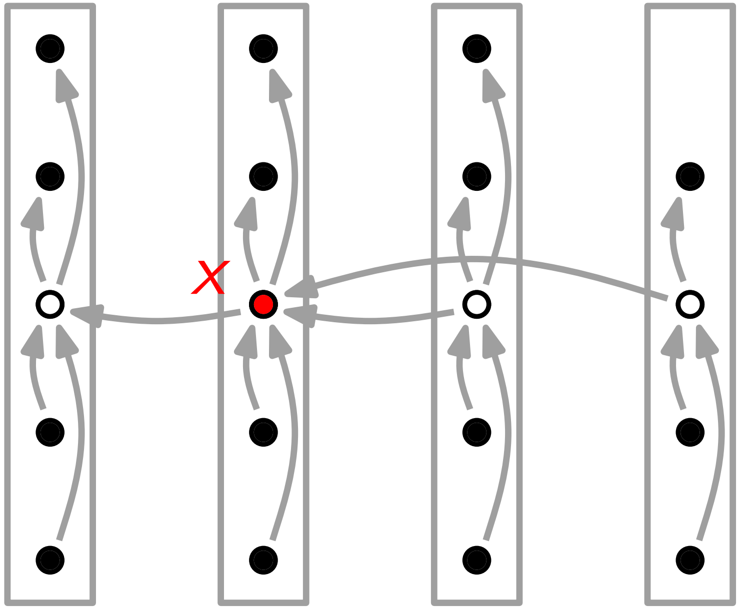
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.



Select: deterministisch

Select(A, ℓ, r, i)

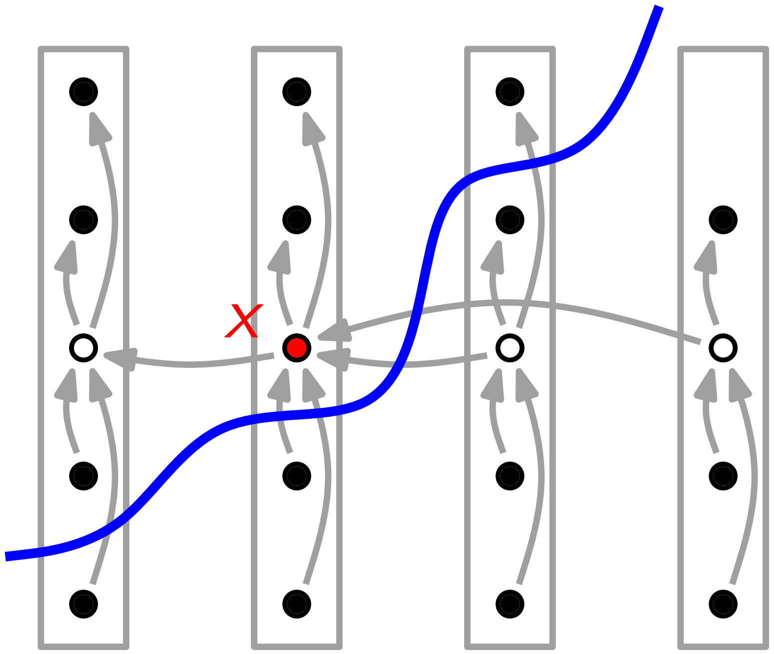
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.
4. $m =$ Partition$'(A, \ell, r, x)$



Select: deterministisch

Select(A, ℓ, r, i)

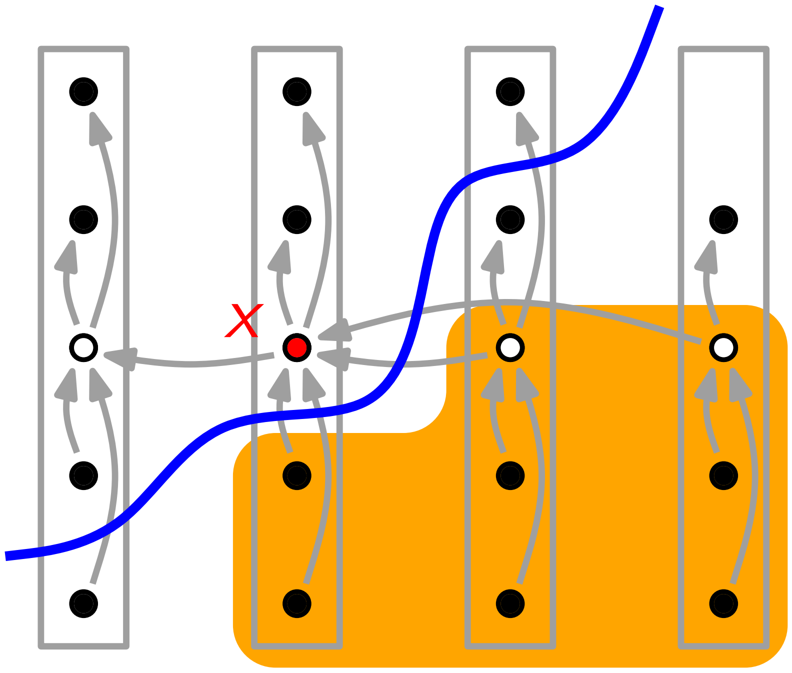
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.
4. $m =$ Partition$'(A, \ell, r, x)$



Select: deterministisch

Select(A, ℓ, r, i)

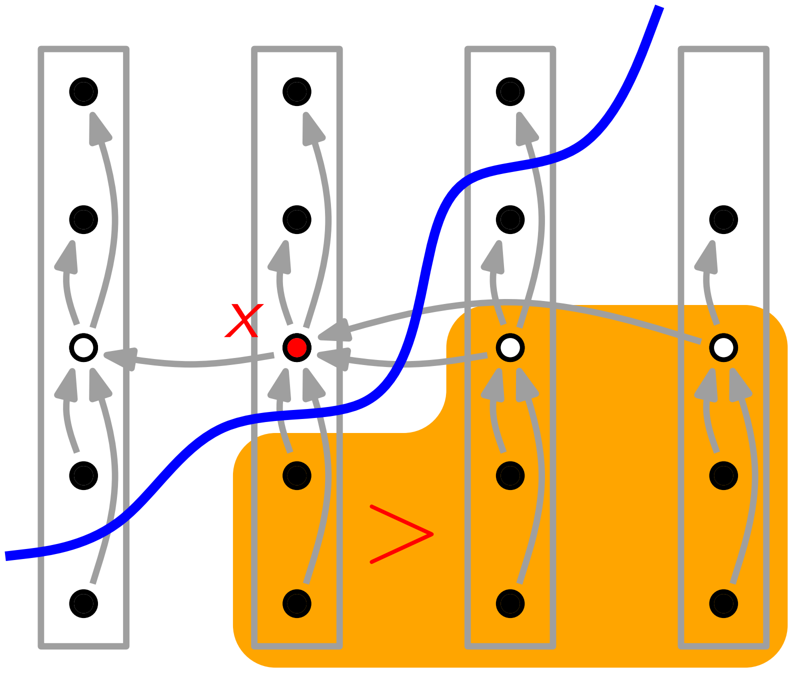
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.
4. $m = \text{Partition}'(A, \ell, r, x)$



Select: deterministisch

Select(A, ℓ, r, i)

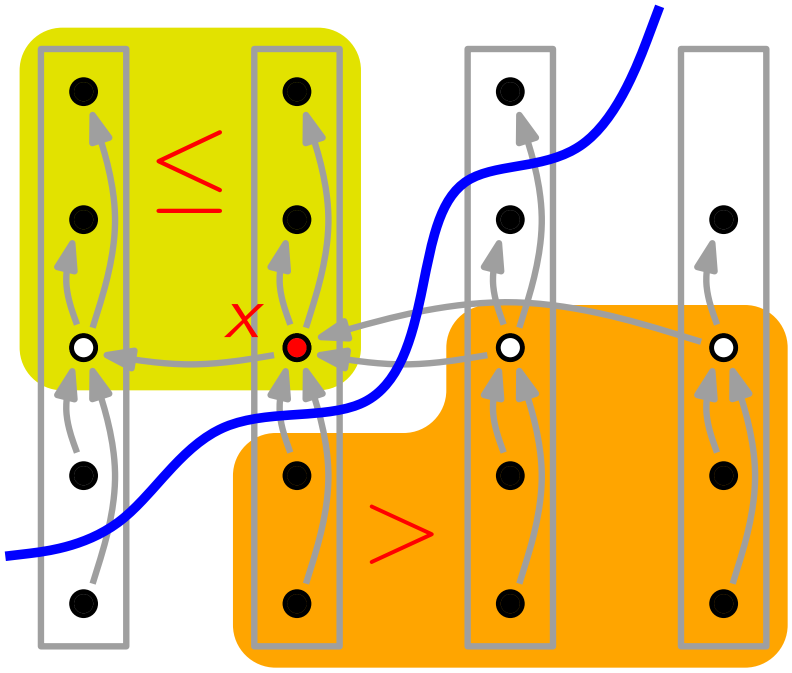
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.
4. $m =$ Partition$'(A, \ell, r, x)$



Select: deterministisch

Select(A, ℓ, r, i)

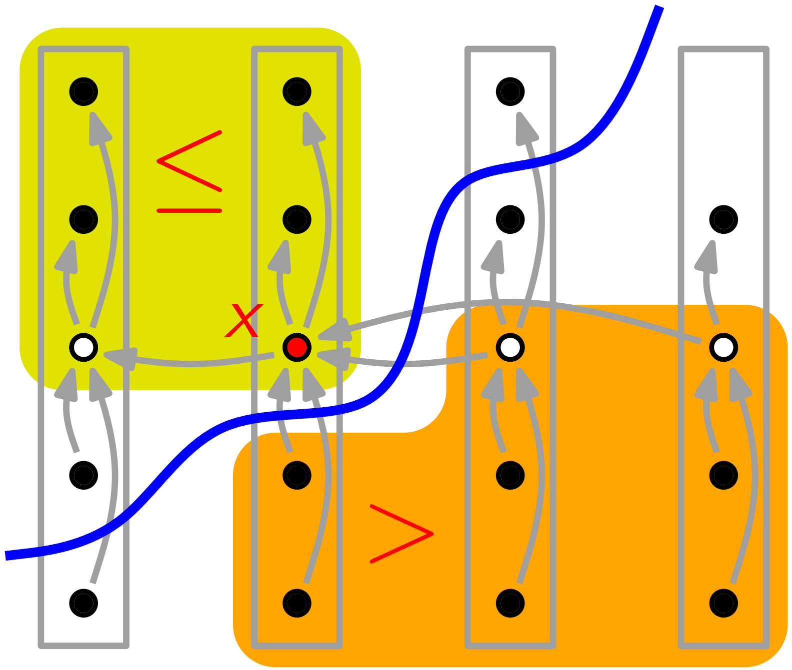
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.
4. $m =$ Partition$'(A, \ell, r, x)$



Select: deterministisch

Select(A, ℓ, r, i)

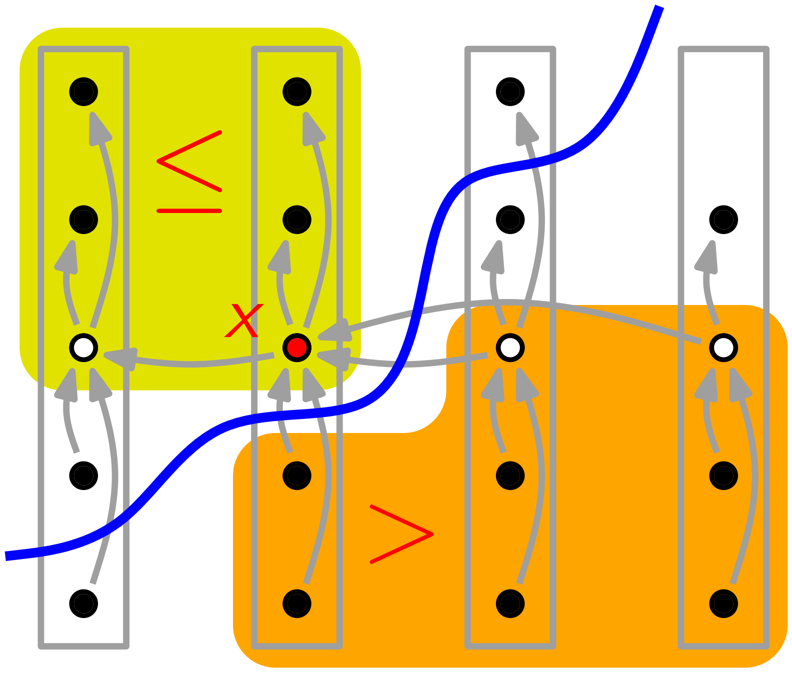
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.
4. $m =$ Partition$'(A, \ell, r, x)$; $k = m -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



Select: deterministisch

Select(A, ℓ, r, i)

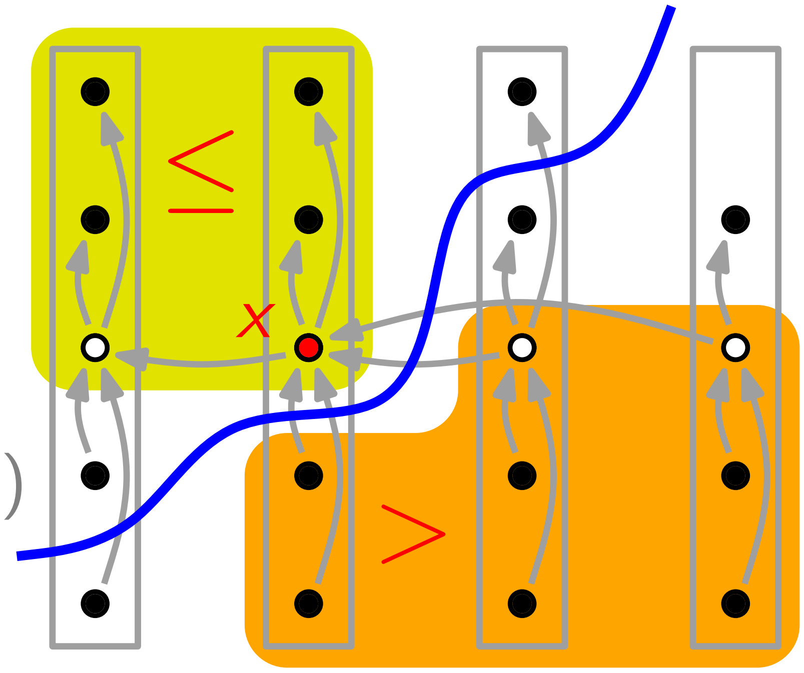
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.
4. $m =$ Partition$'(A, \ell, r, x)$; $k = m -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



Select: deterministisch

Select(A, ℓ, r, i)

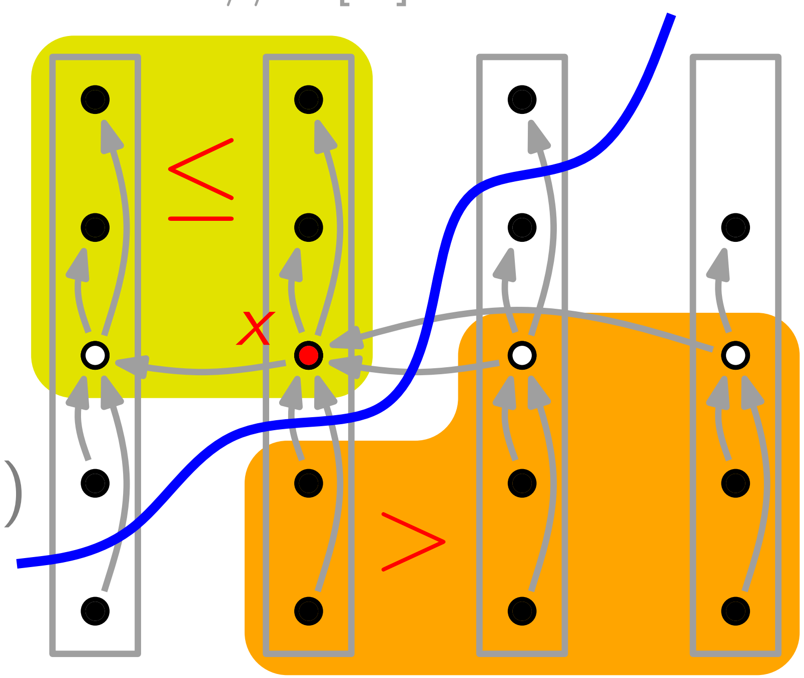
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.
4. $m = \text{Partition}'(A, \ell, r, x)$; $k =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



Select: deterministisch

Select(A, ℓ, r, i)

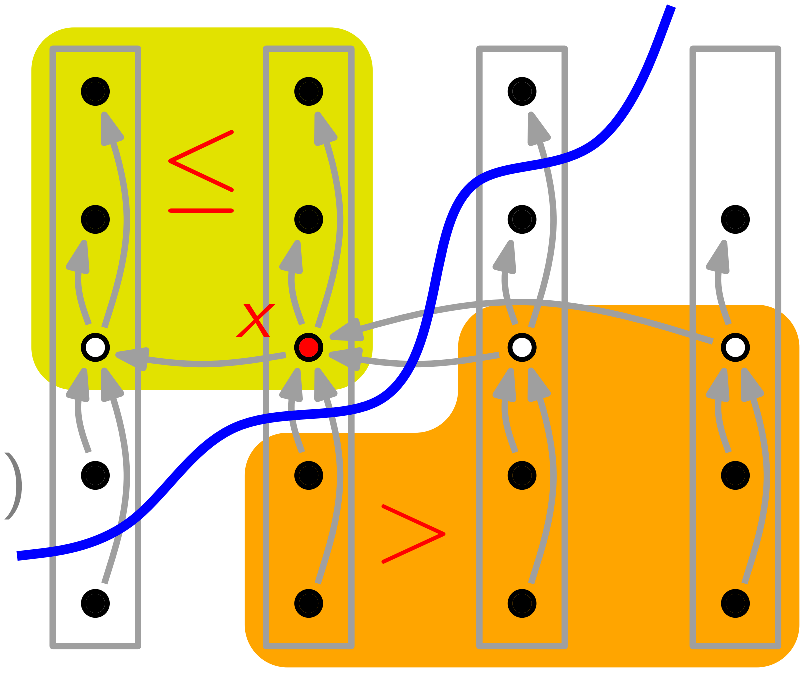
1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elem.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme *rekursiv* den Median x der Gruppen-Mediane.
4. $m =$ Partition$'(A, \ell, r, x)$; $k = m -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



Select: deterministisch

Select(A, ℓ, r, i)

1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 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** $\underbrace{A[m]}_{x}$
   else
   if $i < k$ **then**
   return Select$(A, \ell, m-1, i)$
   else
   return Select$(A, m+1, r, i-k)$

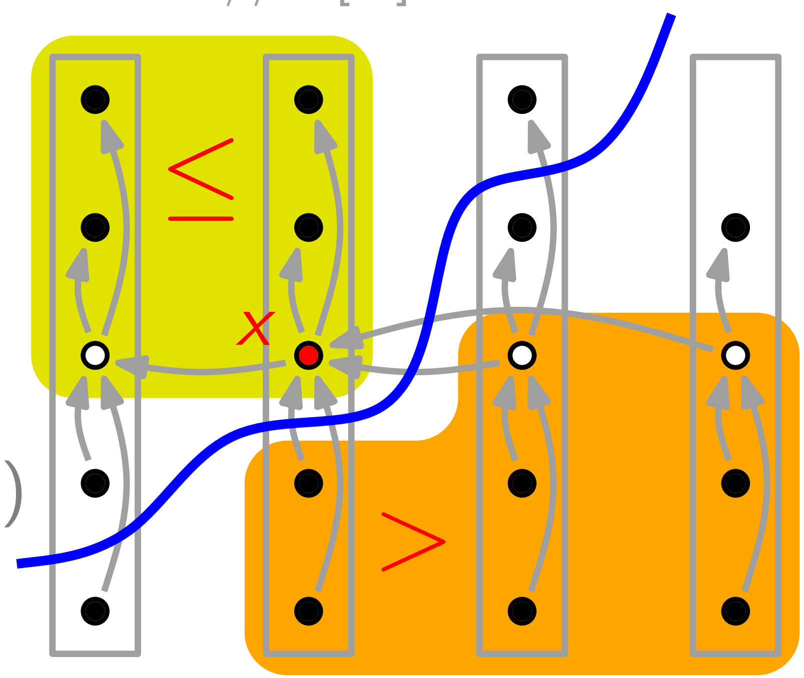


Anzahl(•) ≥

Select: deterministisch

Select(A, ℓ, r, i)

1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 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. $m =$ Partition$'(A, \ell, r, x)$; $k = m - \ell + 1$ // $A[m]$ k.-kleinstes El.
5. **if** $i == k$ **then return** $\underbrace{A[m]}_{x}$
   else
   if $i < k$ **then**
   return Select$(A, \ell, m-1, i)$
   else
   return Select$(A, m+1, r, i-k)$

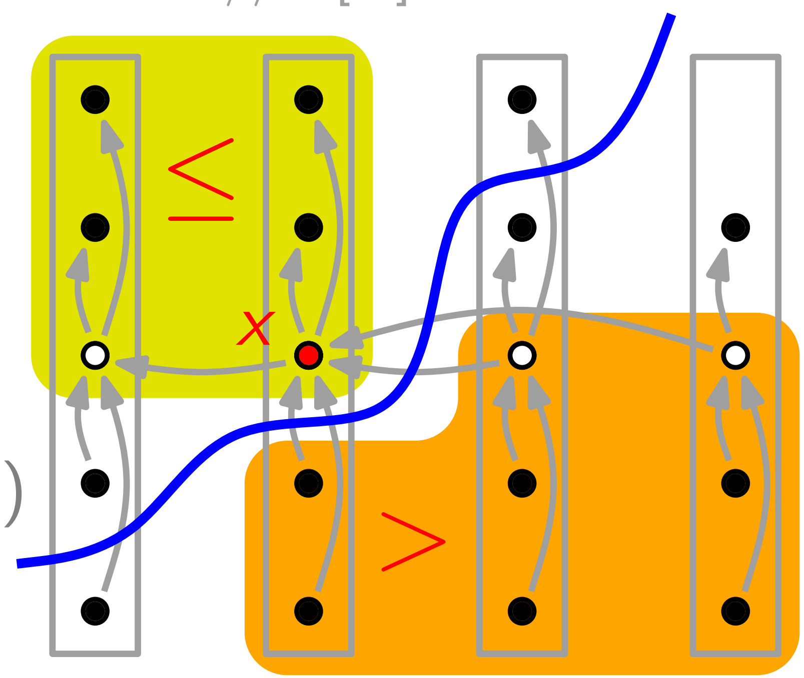


Anzahl(●) $\geq 3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5}\right\rceil\right\rceil - 2\right)$

Select: deterministisch

Select(A, ℓ, r, i)

1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 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. $m =$ Partition$'(A, \ell, r, x)$; $k = m - \ell + 1$ // $A[m]$ k.-kleinstes El.
5. **if** $i == k$ **then return** $\underbrace{A[m]}_{x}$
   else
   if $i < k$ **then**
   return Select$(A, \ell, m-1, i)$
   else
   return Select$(A, m+1, r, i-k)$

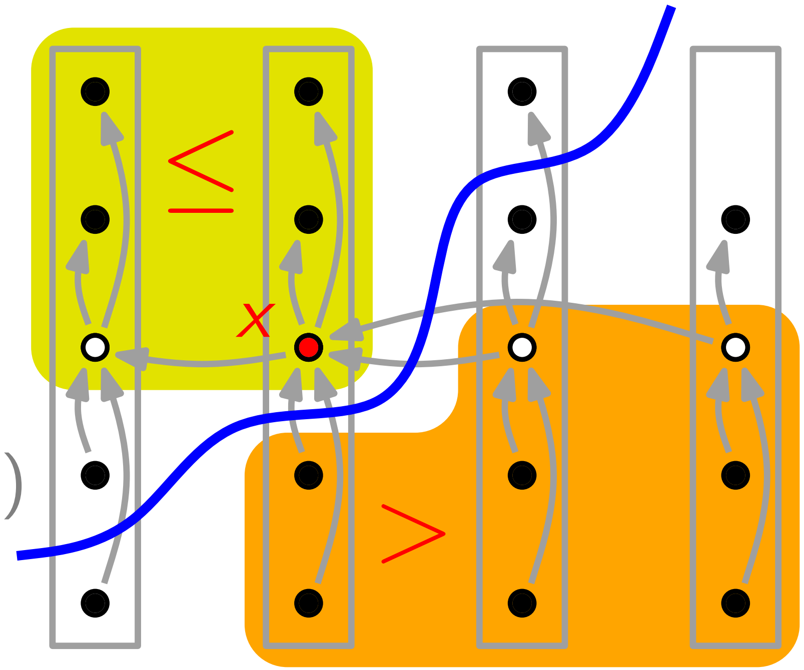


Anzahl(•) $\geq 3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \boxed{\frac{3n}{10} - 6}$

Select: deterministisch

Select(A, ℓ, r, i)

1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 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** $\underbrace{A[m]}_{x}$
   else
   if $i < k$ **then**
   return Select$(A, \ell, m-1, i)$
   else
   return Select$(A, m+1, r, i-k)$

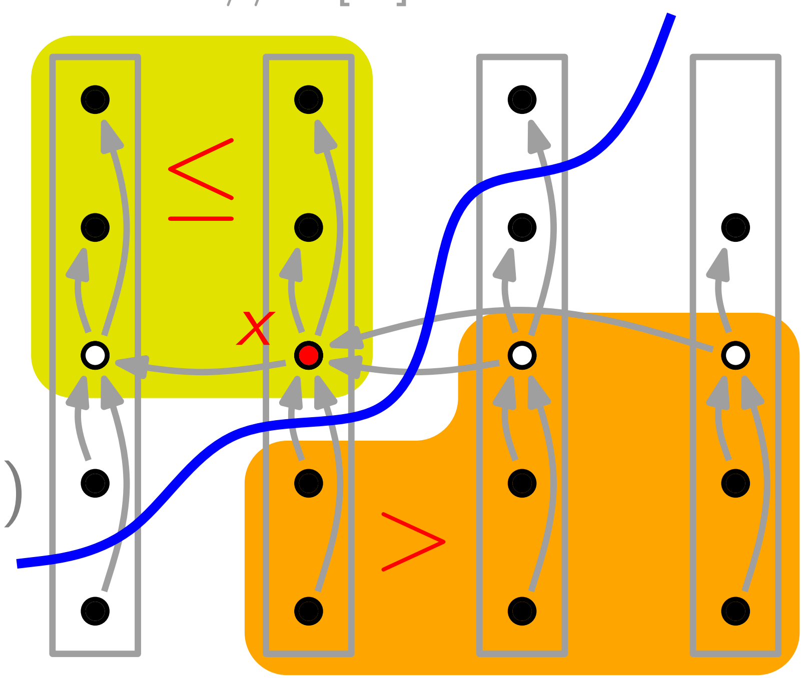


Anzahl(●) $\geq 3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \boxed{\frac{3n}{10} - 6}$

Select: deterministisch

Select(A, ℓ, r, i)

1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 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** $\underbrace{A[m]}_{x}$
   else
   if $i < k$ **then**
   return Select$(A, \underbrace{\ell, m-1,} i)$
   else
   return Select$(A, \overbrace{m+1, r,} i - k)$

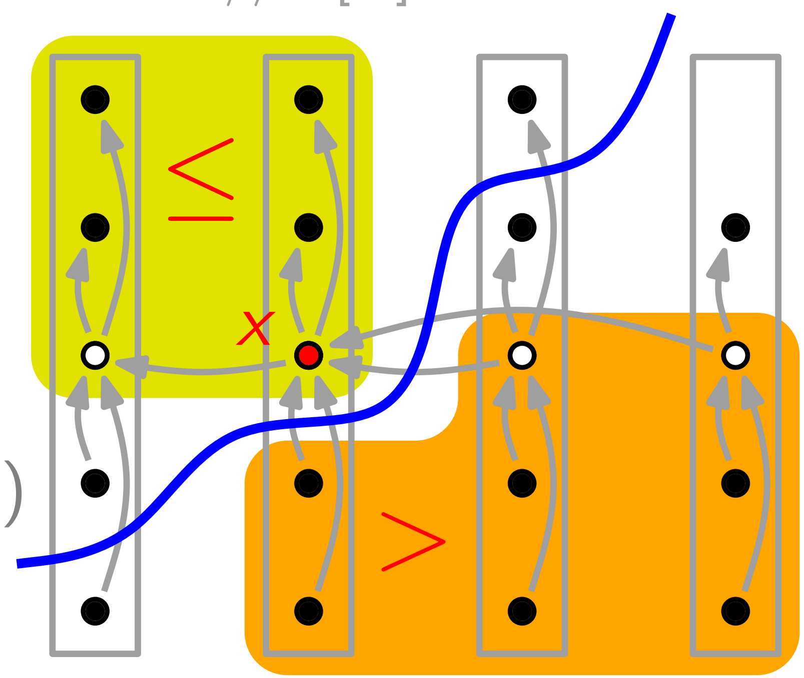


Anzahl$(\bullet) \geq 3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \boxed{\frac{3n}{10} - 6}$

Select: deterministisch

Select(A, ℓ, r, i)

1. Teile die n Elem. der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 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. $m =$ Partition$'(A, \ell, r, x)$; $k = m - \ell + 1$ // $A[m]$ k.-kleinstes El.
5. **if** $i == k$ **then return** $\underbrace{A[m]}_{x}$
   else
   if $i < k$ **then**
   return Select$(A, \underbrace{\ell, m-1}_{\le 7n/10 + 6 \text{ Elem.}}, i)$
   else
   return Select$(A, \overbrace{m+1, r}^{\le 7n/10 + 6 \text{ Elem.}}, i-k)$



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

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: $\approx \frac{n}{5} \cdot V_{\mathsf{IS}}(5) = 2n$ Vgl.

Laufzeit-Analyse

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

Ansatz:

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

Laufzeit-Analyse

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

Ansatz:

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

Behauptung:

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

Laufzeit-Analyse

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

Ansatz:

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

Behauptung:

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

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

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

Laufzeit-Analyse

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

Ansatz:

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

Behauptung:

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

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

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

Laufzeit-Analyse

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

Ansatz:

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

Behauptung:

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

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

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

Laufzeit-Analyse

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

Ansatz:

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

Behauptung:

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

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

$$= c \cdot (9n/10 + 7) + 3n = cn - \Big(\underbrace{c \cdot (n/10 - 7) - 3n}_{\overset{?!}{\ge} 0}\Big)$$

Laufzeit-Analyse

Beob. Es genügt wieder, Vergleiche zu zählen!
Partition′: $\approx 1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ 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: $V(n) \leq cn$.

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

$$= c \cdot (9n/10 + 7) + 3n = cn - \left(c \cdot (n/10 - 7) - 3n\right) \quad \overset{?!}{\geq} 0$$

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

Laufzeit-Analyse

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

Ansatz:

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

Behauptung:

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

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

$$= c \cdot (9n/10 + 7) + 3n = cn - \Big(\underbrace{c \cdot (n/10 - 7) - 3n}_{}\Big) \quad \overset{?!}{\ge} 0$$

falls $c \ge \frac{3n}{n/10-7} = \frac{30}{1-70/n} \xrightarrow[n \to \infty]{}$

Laufzeit-Analyse

Beob. Es genügt wieder, Vergleiche zu zählen!
Partition′: $\approx 1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ 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: $V(n) \leq cn$.

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

$$= c \cdot (9n/10 + 7) + 3n = cn - \big(\underbrace{c \cdot (n/10 - 7) - 3n}\big) \quad \overset{?!}{\geq} 0$$

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

Laufzeit-Analyse

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

Ansatz:

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

Behauptung:

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

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

$$= c \cdot (9n/10 + 7) + 3n = cn - \big(\underbrace{c \cdot (n/10 - 7) - 3n}_{\overset{?!}{\ge} 0}\big)$$

falls $c \ge \frac{3n}{n/10-7} = \frac{30}{1-70/n} \xrightarrow[n \to \infty]{} 30^+$ bzw. $n \ge \frac{70c}{c-30}$.

Laufzeit-Analyse

Beob. Es genügt wieder, Vergleiche zu zählen!
Partition′: $\approx 1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ 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: $V(n) \leq cn$.

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

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

falls $c \geq \frac{3n}{n/10-7} = \frac{30}{1-70/n} \xrightarrow[n \to \infty]{} 30^+$ 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: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ 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: $V(n) \leq cn$.

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

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

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

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

Laufzeit-Analyse

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

Ansatz:

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

Behauptung:

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

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

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

falls $c \ge \frac{3n}{n/10-7} = \frac{30}{1-70/n} \xrightarrow[n \to \infty]{} 30^+$ bzw. $n \ge \frac{70c}{c-30}$.

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

Laufzeit-Analyse

Beob. Es genügt wieder, Vergleiche zu zählen!
Partition′: $\approx 1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ 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: $V(n) \leq cn$.

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

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

falls $c \geq \frac{3n}{n/10-7} = \frac{30}{1-70/n} \xrightarrow[n \to \infty]{} 30^+$ 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′: $\approx \boxed{1}n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = \boxed{2}n$ Vgl.

Ansatz:

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

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

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

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

falls $c \ge \frac{\boxed{3}n}{n/10-7} = \frac{30}{1-70/n} \xrightarrow[n \to \infty]{} 30^+$ bzw. $n \ge \frac{70c}{c-30}$.

$\Rightarrow$ für jedes $\varepsilon > 0$ und $n \ge \frac{2100}{\varepsilon} + 70$ gilt: $V(n) \le (\boxed{30} + \varepsilon) \cdot n$

Laufzeit-Analyse

Kann man das verbessern?

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

Ansatz:

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

Behauptung:

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

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

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

falls $c \geq \frac{\boxed{3}n}{n/10-7} = \frac{30}{1-70/n} \xrightarrow[n \to \infty]{} 30^+$ bzw. $n \geq \frac{70c}{c-30}$.

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

Laufzeit-Analyse

Kann man das verbessern?

Hausaufgabe!

Beob. Es genügt wieder, Vergleiche zu zählen!

Partition′: $\approx 1n$, Sortieren: $\approx \frac{n}{5} \cdot V_{\mathrm{IS}}(5) = 2n$ 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: $V(n) \leq cn$.

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

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

falls $c \geq \frac{3n}{n/10-7} = \frac{30}{1-70/n} \xrightarrow[n \to \infty]{} 30^+$ 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.