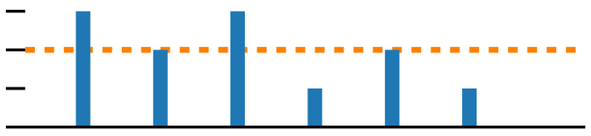
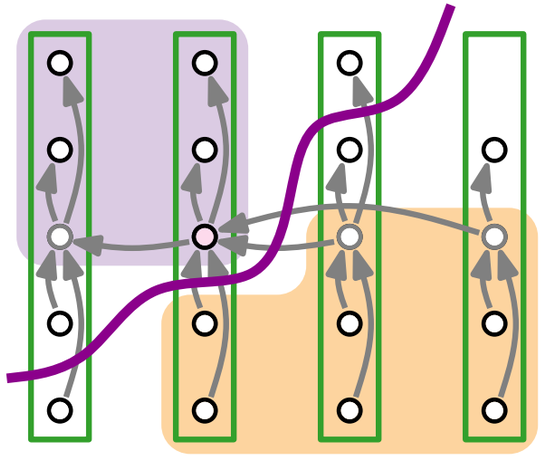
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Algorithmen und Datenstrukturen

Vorlesung 10:
Das Auswahlproblem





Alexander Wolff

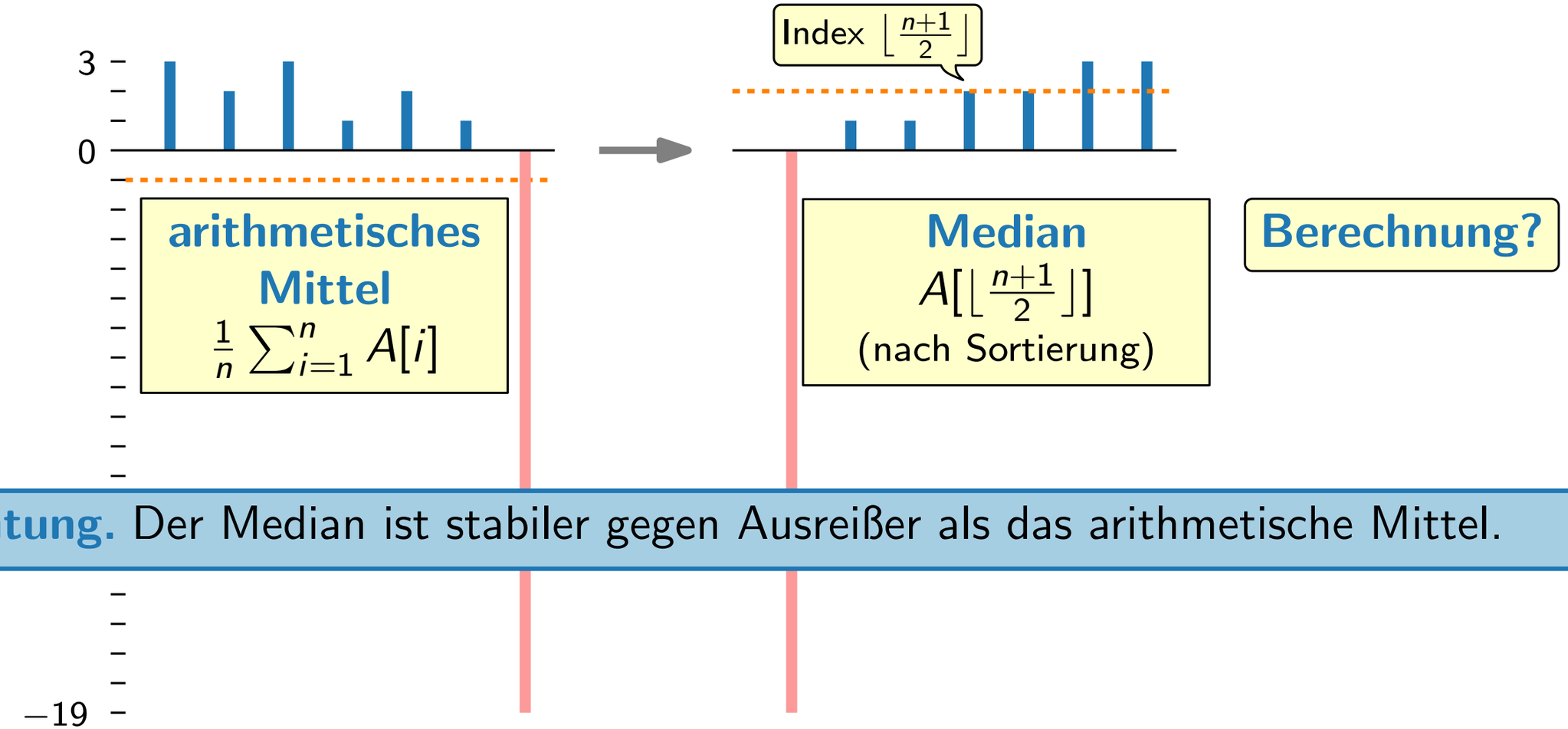
Wintersemester 2024

Analyse von Messreihen

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

Gesucht: Ein „guter“ Mittelwert

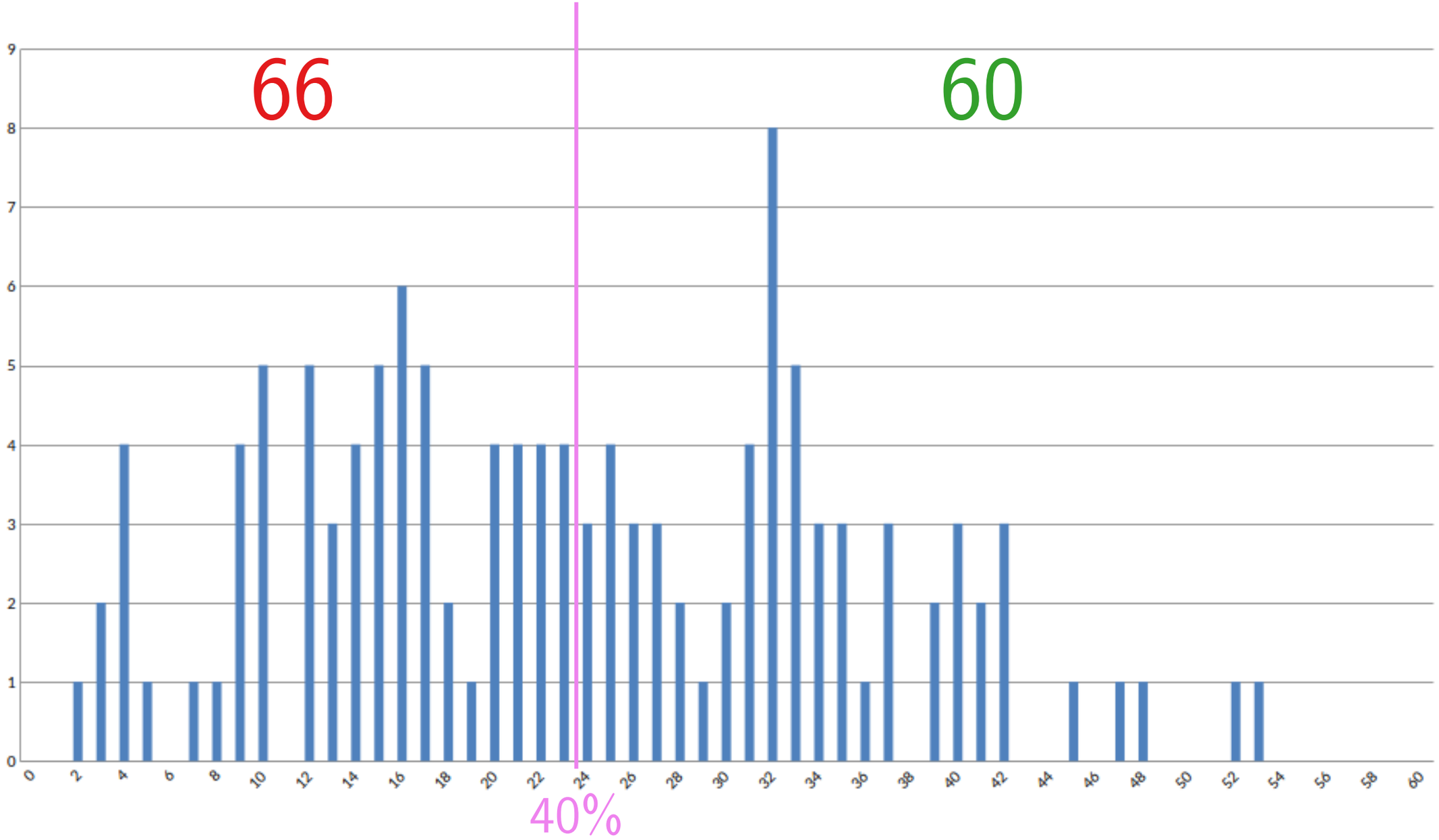
Beispiel.



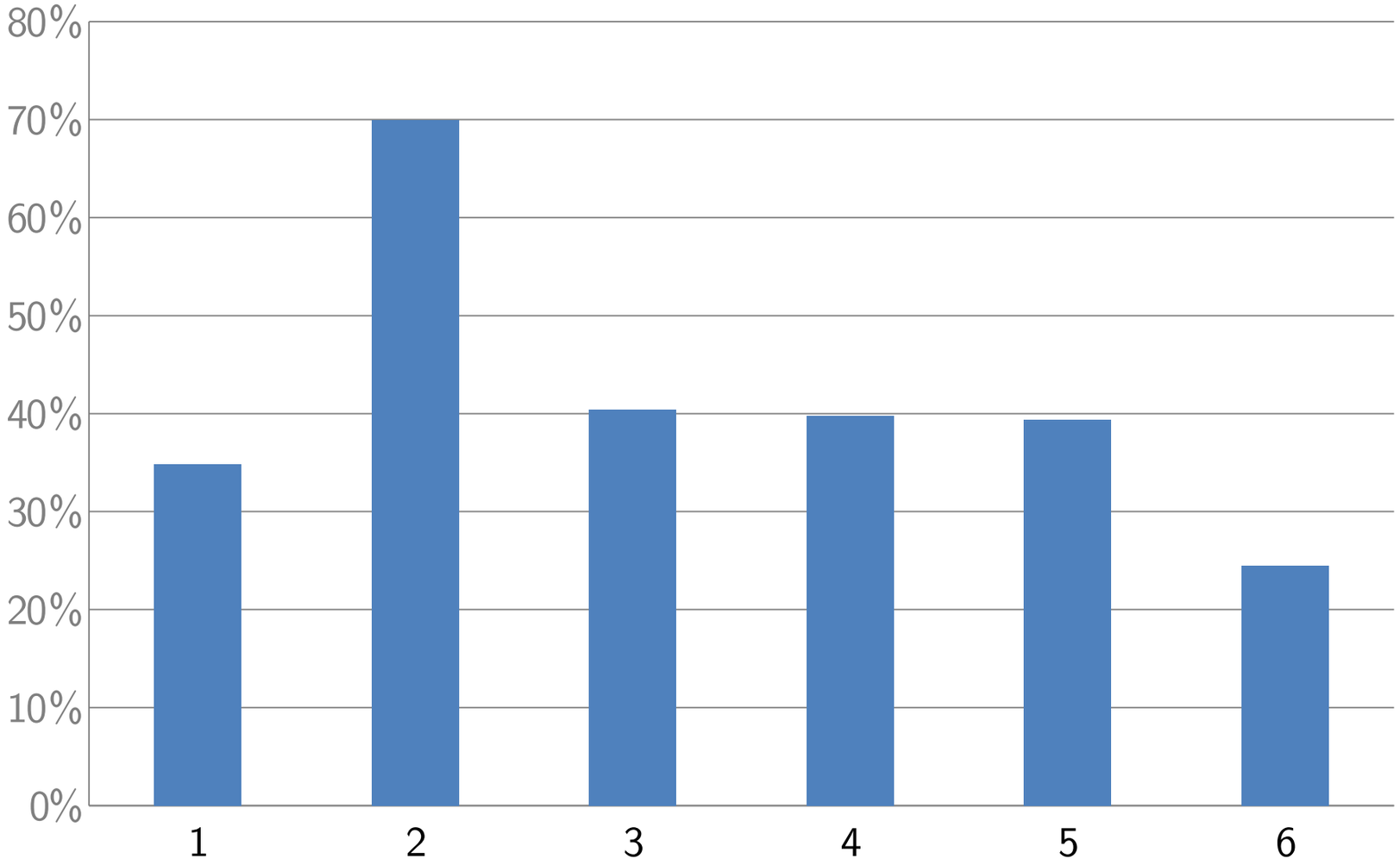
Beobachtung. Der Median ist stabiler gegen Ausreißer als das arithmetische Mittel.

1. Zwischentest

$n = 126$; Durchschnitt $= 23{,}7$; Median $= 21{,}5$.



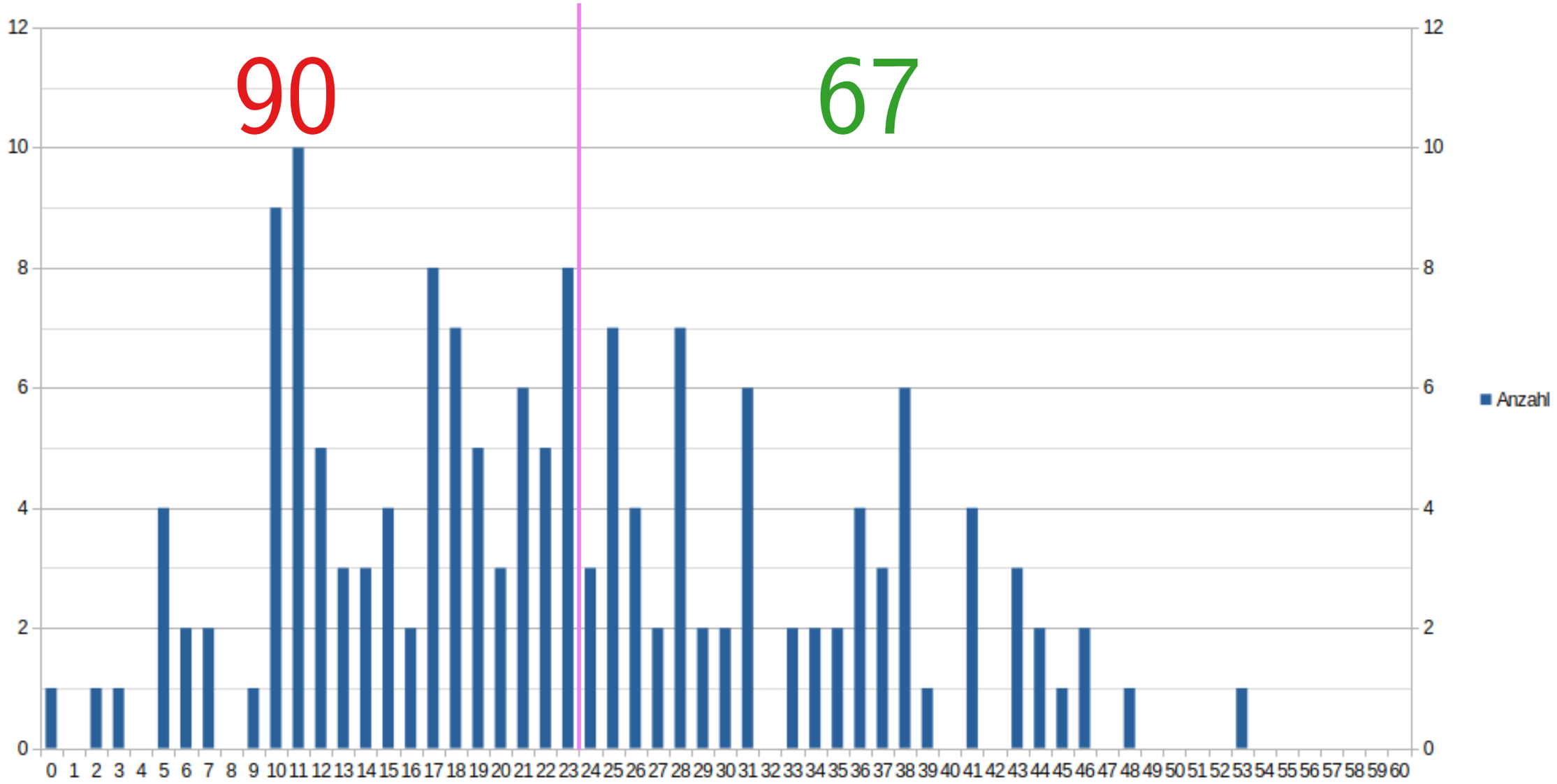
Ergebnisse nach Aufgabe



O-Notation Analyse Pseudocode

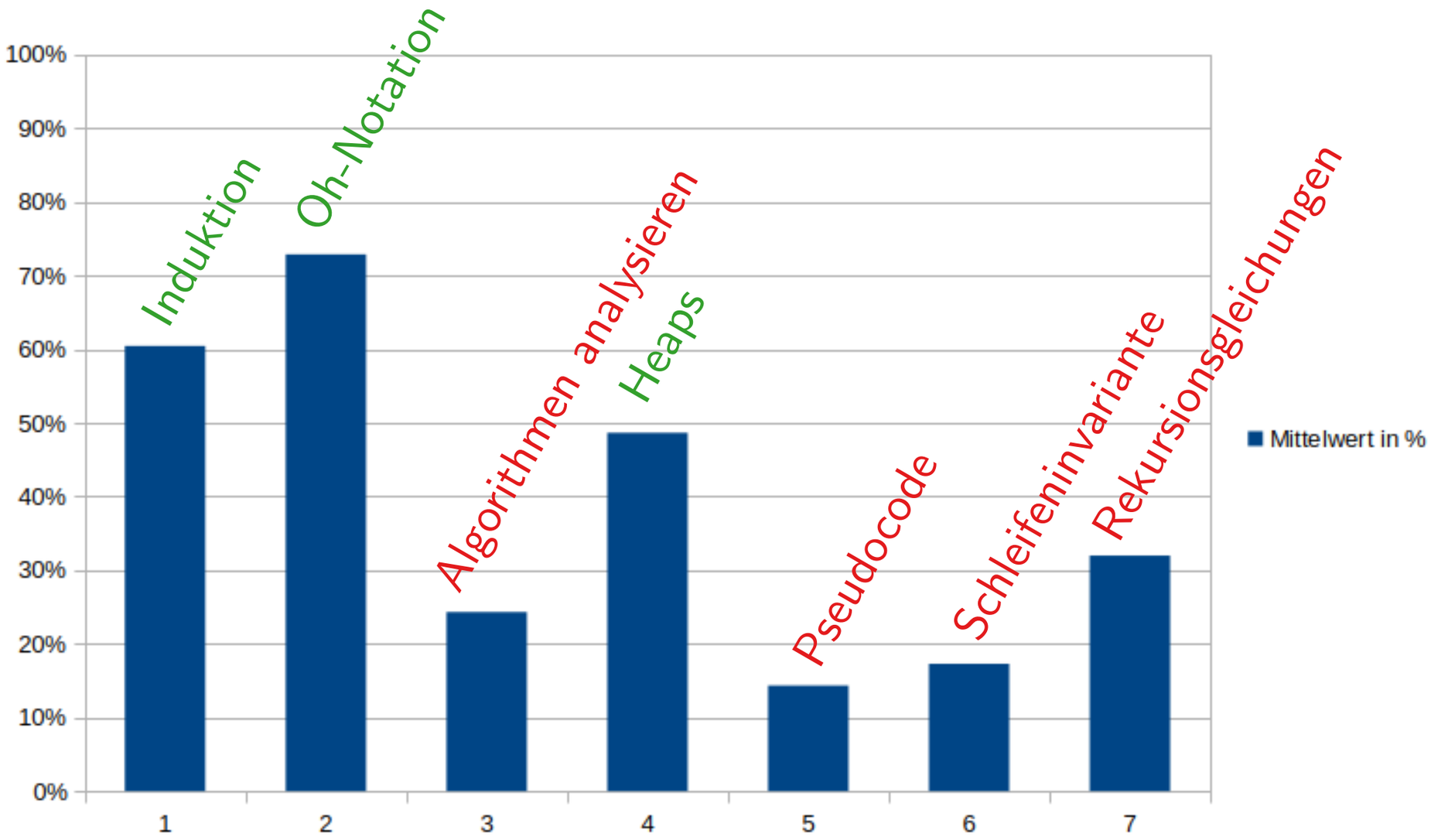
Heaps Invariante Rekursionsgleichungen

1. Zwischentest (letztes Jahr)



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

Ergebnisse nach Aufgabe



Das Auswahlproblem

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

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

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

wenn man nichts über die Verteilung der Zahlen weiß

Geht das besser?

Spezialfälle

Geht das auch in linearer Zeit?

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

$i = 1$: Minimum

$i = n$: Maximum

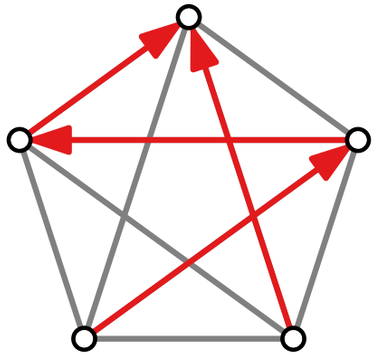
} Laufzeit $\Theta(n)$

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

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

Anzahl Vergleiche $= n - 1$

Ist das optimal?



Betrachte ein K.O.-Turnier.

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

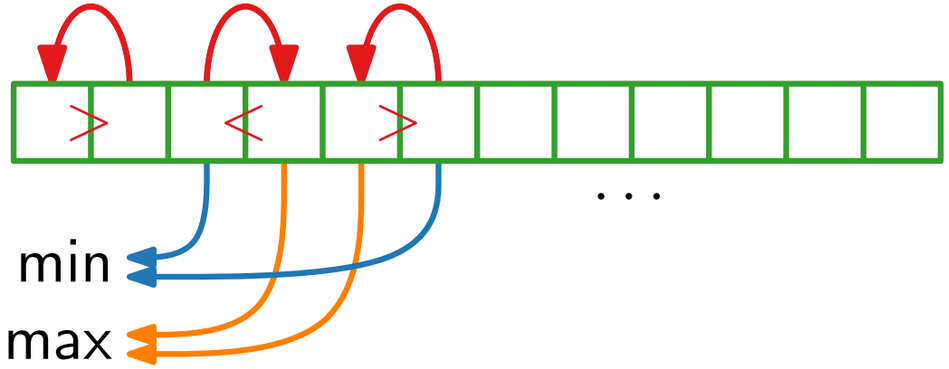
Also sind $n - 1$ Vergleiche optimal.

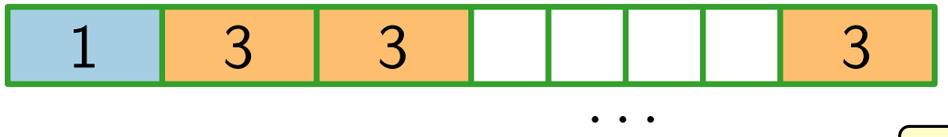
Zuerst zur zweiten Frage

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

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

Frage. Geht es auch mit weniger Vergleichen?





Anzahl der Vergleiche

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

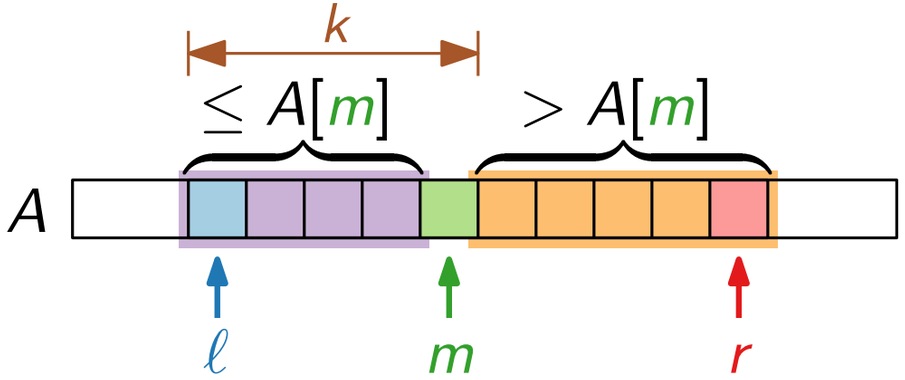
n gerade

Ist das optimal?

Auswahl per Teile & Herrsche

Zur Erinnerung...

```
RandomizedQuickSort(A, ℓ = 1, r = A.length)
  if ℓ < r then
    m = RandomizedPartition(A, ℓ, r)
    QuickSort(A, ℓ, m − 1)
    QuickSort(A, m + 1, r)
```



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

```
RandomizedSelect(int[] A, int ℓ, int r, int i)
  if ℓ == r then return A[ℓ]
  m = RandomizedPartition(A, ℓ, r)
  k = m − ℓ + 1        // Rang von A[m] in A[ℓ ... r]
  if i == k then
    return A[m]
  else
    if i < k then
      return RSelect(A, ℓ, m − 1, i)
    else
      return RSelect(A, m + 1, r, i − k)
```

Laufzeitanalyse

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

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

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

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

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

vorausgesetzt alle Elemente sind verschieden!

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

Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

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

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

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

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

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

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

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

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

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

≤ 0?!

Für jedes $\varepsilon > 0$ gilt:

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

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

Ergebnis und Diskussion

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

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

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

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

Vorbereitung

Wir verwenden wieder Divide & Conquer –

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

d.h. **balanciert:**

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

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

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

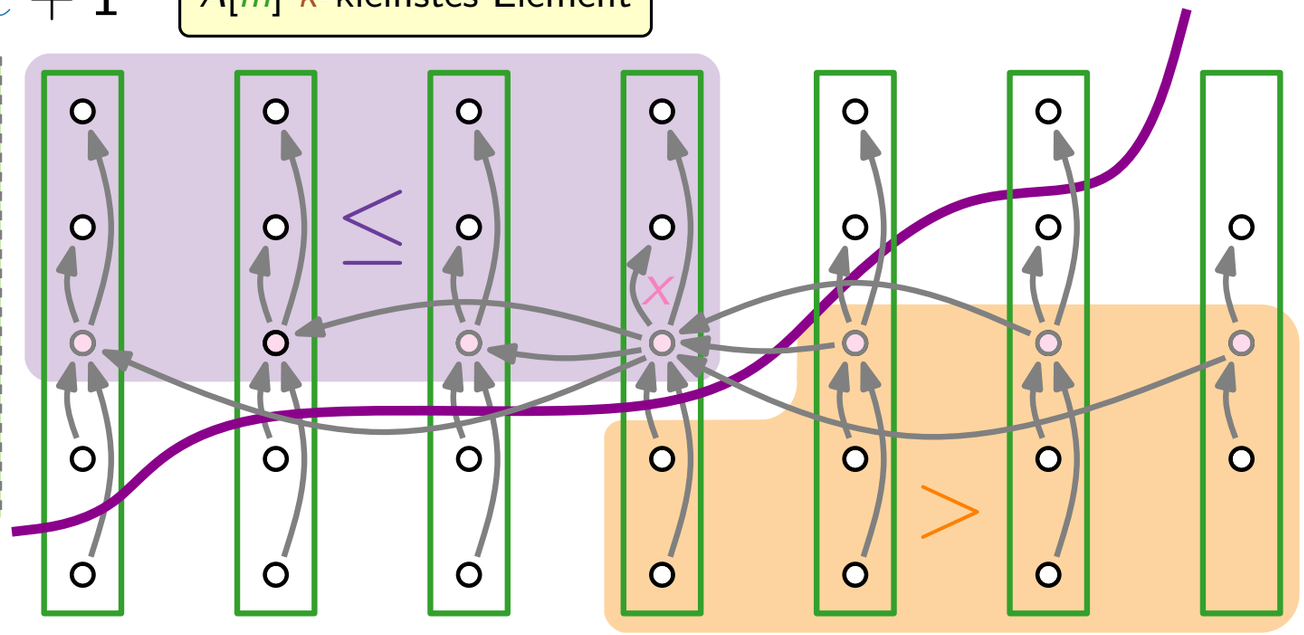
Anzahl der Vergleiche, die PARTITION′ macht: $r - \ell + 1 = n$

SELECT: deterministisch

SELECT(int[] A, int ℓ, int r, int i)

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

$A[m]$ k-kleinstes Element

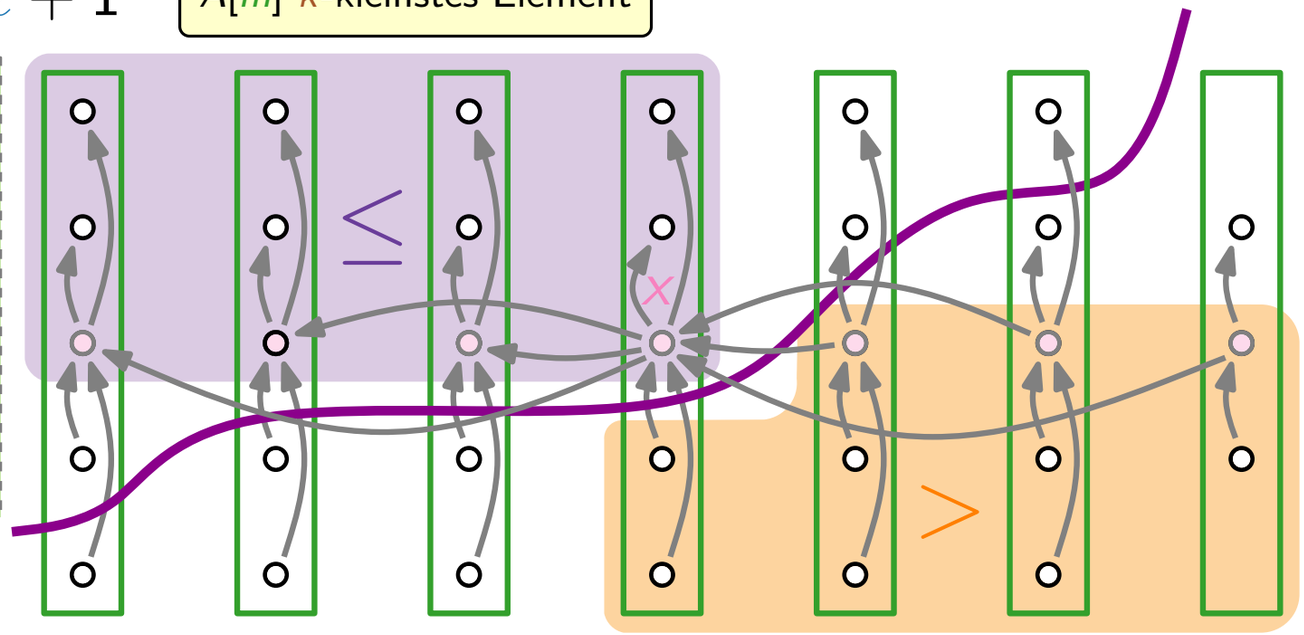


Anzahl(●) $\geq 3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq \frac{3n}{10} - 3$

SELECT: deterministisch

SELECT(int[] A, int ℓ, int r, int i)

1. Teile die n Elemente der Eingabe in $\lfloor n/5 \rfloor$ 5er-Gruppen und eine Gruppe mit den restlichen (n mod 5) Elementen.
2. Sortiere jede der $\lceil n/5 \rceil$ Gruppen und bestimme ihren Median.
3. Bestimme **rekursiv** den Median x der Gruppen-Mediane.
4. $m = \text{PARTITION}'(A, \ell, r, x)$; $k = m - \ell + 1$ — $A[m]$ k-kleinstes Element
5. **if** $i == k$ **then return** $A[m]$ (x)
   else
   if $i < k$ **then**
   return SELECT$(A, \ell, m-1, i)$ ($\leq 7n/10 + 3$ Elemente)
   else
   return SELECT$(A, m+1, r, i-k)$



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

Laufzeitanalyse

Beobachtung. Es genügt wieder, Vergleiche zu zählen!

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

Ansatz.

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

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

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

$$= c \cdot (9n/10 + 4) + 3n \ = cn + \big(3n - c \cdot (n/10 - 4)\big)$$

falls $c \geq \frac{3n}{n/10-4} = \frac{30}{1-40/n} \xrightarrow[n\to\infty]{} 30^+$ bzw. $n \geq \frac{40c}{c-30}$

$\Rightarrow$ für jedes $\varepsilon > 0$ und $n \geq \frac{1200}{\varepsilon} + 40$ gilt: $V(n) \leq \underbrace{(30+\varepsilon)}_{c} \cdot n$

$c \geq \frac{3n}{n/10-4}$
$\Leftrightarrow c(n/10-4) \geq 3n$
$\Leftrightarrow cn/10 - 4c - 3n \geq 0$
$\Leftrightarrow n(c/10-3) \geq 4c$
$\Leftrightarrow n \geq \frac{4c}{c/10-3}$
$\Leftrightarrow n \geq \frac{40c}{c-30}$

Ergebnis und Diskussion

Satz. Das Auswahlproblem kann auch im schlechtesten Fall in linearer Zeit gelöst werden.

Genauer: Für jedes $\varepsilon > 0$ gilt, dass man in einer Folge von $n \geq 1200/\varepsilon + 40$ Zahlen die i-kleinste Zahl mit höchstens $(30 + \varepsilon)n$ Vergleichen finden kann.

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

- Die besten deterministischen Auswahl-Algorithmen (*sehr kompliziert!*) benötigen $3n$ Vergleiche im schlechtesten Fall.
- **Jeder** deterministische Auswahl-Algorithmus benötigt im schlechtesten Fall mindestens $2n$ Vergleiche.

Satz. Durch Suche des Medians in $\mathcal{O}(n)$ Zeit kann QuickSort* im *Worst-Case* in $\mathcal{O}(n \log n)$ Zeit sortieren.

Vergleich Sortieralgorithmen

	Bester Fall	Erw. Fall	Schl. Fall	in-situ	stabil
InsertionSort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✓
SelectionSort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✗
BubbleSort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	✓	✓
MergeSort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✗	✓
HeapSort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✓	✗
QuickSort*	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	✓	✗
CountingSort	$\mathcal{O}(n + k)$	$\mathcal{O}(n + k)$	$\mathcal{O}(n + k)$	✗	✓
RadixSort	$\mathcal{O}(s \cdot (n + b))$	$\mathcal{O}(s \cdot (n + b))$	$\mathcal{O}(s \cdot (n + b))$	✗	✓
BucketSort	$\mathcal{O}(n)$	$\mathcal{O}(n)$ wenn Eingabe zufällig gleichverteilt	$\mathcal{O}(n^2)$	✗	✓