

Algorithmen und Datenstrukturen

Wintersemester 2019/20

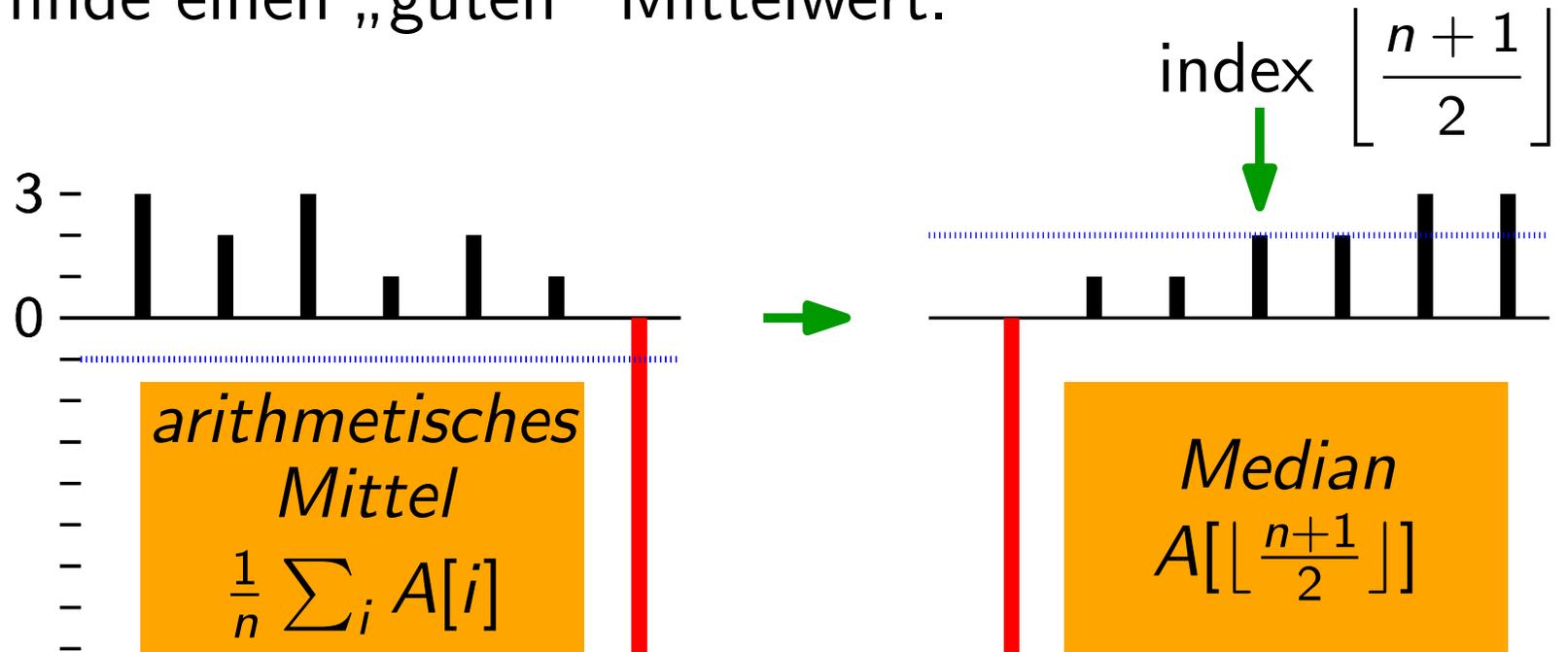
10. Vorlesung

Das Auswahlproblem

Analyse von Messreihen

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

Beispiel:



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

-19-

Berechnung?

Das Auswahlproblem

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

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

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

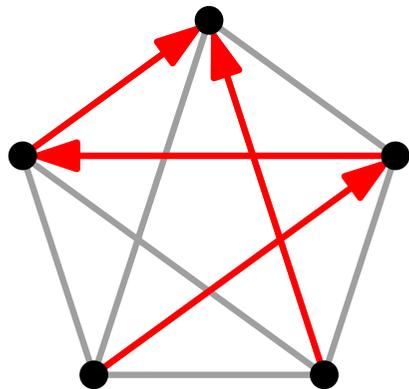
```
    [ if min > A[i] then min = A[i]
```

```
  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.

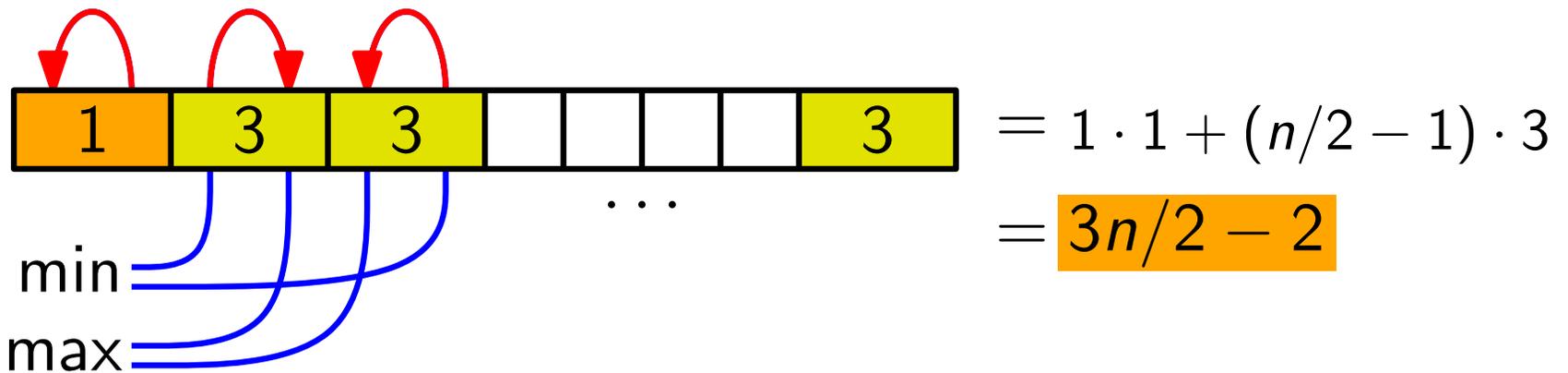
Also sind $n - 1$ Vergleiche optimal.

Eine Randbemerkung...

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

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

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



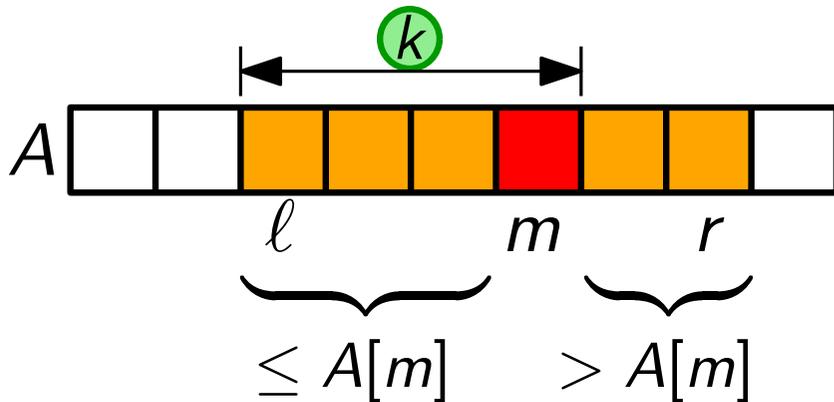
Ist das *optimal*?

Auswahl per Teile & Herrsche

Zur Erinnerung...

```

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



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

```

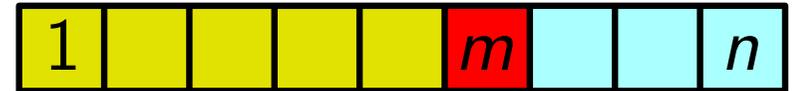
RandomizedSelect(int[] A, int  $\ell, r, i$ )
if  $\ell == r$  then return  $A[\ell]$ 
 $m$  = Randomized Partition( $A, \ell, r$ )
 $k$  =  $m - \ell + 1$  //  $A[m]$  ist  $k$ -kleinstes El.
                       von  $A[\ell..r]$ 
if  $i == k$  then
    | return  $A[m]$ 
else
    | if  $i < k$  then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

Ist Ihnen klar warum?

Laufzeit-Analyse

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

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



⇒ resultierende Zufallsvariable $V(n)$ ist

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

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

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

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

Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

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

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

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

Aufgabe:

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

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

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

Substitutionsmethode

Wir schreiben $f(n)$ für $E[V(n)]$.

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

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

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

Für jedes $\varepsilon > 0$ gilt:

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

Ergebnis und Diskussion

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

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

d.h. *balanciert*:

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

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

```
Partition'(A, l, r, pivot)
```

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

```
i = l - 1
```

```
for j = l to r >= 1 do
```

```
    if A[j] ≤ pivot
```

```
    then
```

```
        i = i + 1
```

```
        Swap(A, i, j)
```

```
Swap(A, i + 1, r)
```

```
return i + 1
```

Select: deterministisch

Select(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.
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

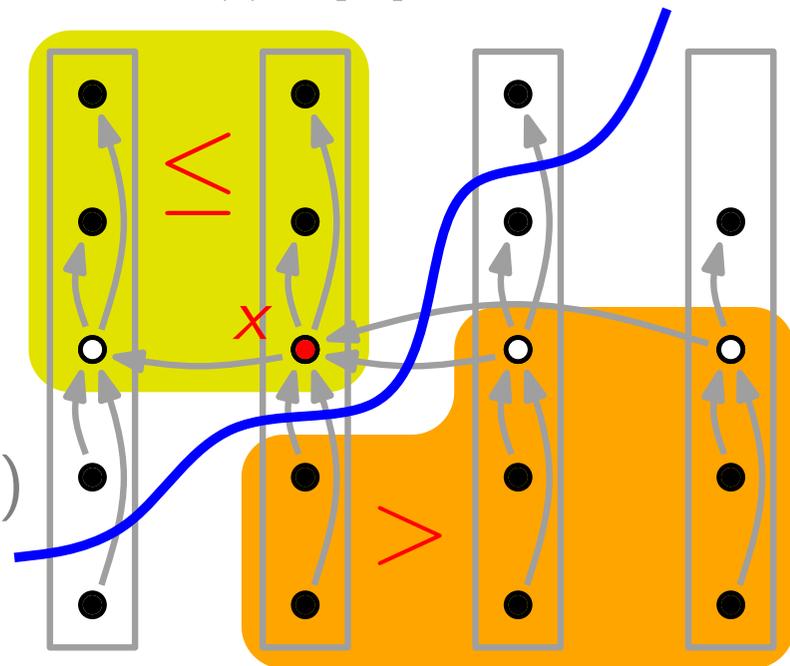
if $i < k$ then

return Select($A, \ell, m - 1, i$)

else

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

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



Select: deterministisch

Select(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.
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

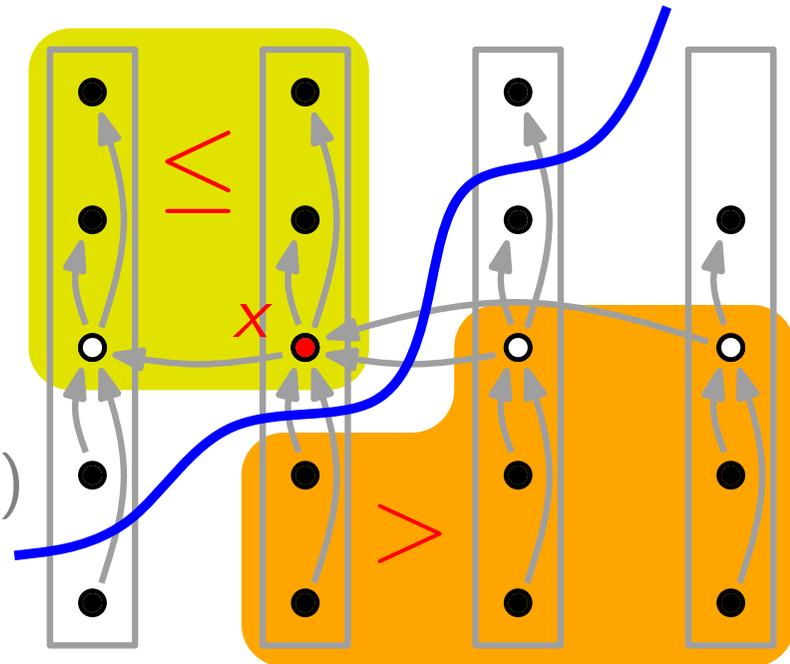
if $i < k$ then

return Select($A, \ell, m - 1, i$)

else

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

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



Laufzeit-Analyse

Beob. Es genügt wieder, Vergleiche zu zählen!

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

Ansatz:

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

Schritt 5

Laufzeit-Analyse

Beob. Es genügt wieder, Vergleiche zu zählen!

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

Ansatz:

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

Behauptung:

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

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

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

Laufzeit-Analyse

Beob. Es genügt wieder, Vergleiche zu zählen!

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

Ansatz:

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

Behauptung:

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

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

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

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

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

Laufzeit-Analyse

Beob. Es genügt wieder, Vergleiche zu zählen!

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

Ansatz:

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

Behauptung:

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

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

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

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

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

Kann man das verbessern?

Hausaufgabe!

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.

