

## 4. Suchen

---

Lineare Suche, Binäre Suche, (Interpolationssuche,) Untere Schranken  
[Ottman/Widmayer, Kap. 3.2, Cormen et al, Kap. 2: Problems 2.1-3,2.2-3,2.3-5]

# Das Suchproblem

## Gegeben

- Menge von Datensätzen.

Telefonverzeichnis, Wörterbuch, Symboltabelle

- Jeder Datensatz hat einen Schlüssel  $k$ .
- Schlüssel sind vergleichbar: eindeutige Antwort auf Frage  $k_1 \leq k_2$  für Schlüssel  $k_1, k_2$ .

Aufgabe: finde Datensatz nach Schlüssel  $k$ .

# Suche in Array

## Gegeben

- Array  $A$  mit  $n$  Elementen ( $A[1], \dots, A[n]$ ).
- Schlüssel  $b$

Gesucht: Index  $k$ ,  $1 \leq k \leq n$  mit  $A[k] = b$  oder "nicht gefunden".

22	20	32	10	35	24	42	38	28	41
1	2	3	4	5	6	7	8	9	10

# Lineare Suche

Durchlaufen des Arrays von  $A[1]$  bis  $A[n]$ .

- Bestenfalls 1 Vergleich.
- Schlimmstenfalls  $n$  Vergleiche.
- Annahme: Jede Anordnung der  $n$  Schlüssel ist gleichwahrscheinlich.  
Erwartete Anzahl Vergleiche für die erfolgreiche Suche:

$$\frac{1}{n} \sum_{i=1}^n i = \frac{n+1}{2}.$$

# Suche im sortierten Array

Gegeben

- Sortiertes Array  $A$  mit  $n$  Elementen ( $A[1], \dots, A[n]$ ) mit  $A[1] \leq A[2] \leq \dots \leq A[n]$ .
- Schlüssel  $b$

Gesucht: Index  $k$ ,  $1 \leq k \leq n$  mit  $A[k] = b$  oder "nicht gefunden".

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

# Divide and Conquer!

Suche  $b = 23$ .

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

$b < 28$

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

$b > 20$

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

$b > 22$

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

$b < 24$

10	20	22	24	28	32	35	38	41	42
1	2	3	4	5	6	7	8	9	10

erfolglos

# Binärer Suchalgorithmus    BSearch( $A, l, r, b$ )

**Input:** Sortiertes Array  $A$  von  $n$  Schlüsseln. Schlüssel  $b$ . Bereichsgrenzen  
 $1 \leq l, r \leq n$  mit  $l \leq r$  oder  $l = r + 1$ .

**Output:** Index  $m \in [l, \dots, r + 1]$ , so dass  $A[i] \leq b$  für alle  $l \leq i < m$  und  
 $A[i] \geq b$  für alle  $m < i \leq r$ .

$m \leftarrow \lfloor (l + r) / 2 \rfloor$

**if**  $l > r$  **then** // erfolglose Suche

**return**  $l$

**else if**  $b = A[m]$  **then** // gefunden

**return**  $m$

**else if**  $b < A[m]$  **then** // Element liegt links

**return** BSearch( $A, l, m - 1, b$ )

**else** //  $b > A[m]$ : Element liegt rechts

**return** BSearch( $A, m + 1, r, b$ )

# Analyse (schlechtester Fall)

Rekurrenz ( $n = 2^k$ )

$$T(n) = \begin{cases} d & \text{falls } n = 1, \\ T(n/2) + c & \text{falls } n > 1. \end{cases}$$

Teleskopieren:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + c = T\left(\frac{n}{4}\right) + 2c = \dots \\ &= T\left(\frac{n}{2^i}\right) + i \cdot c \\ &= T\left(\frac{n}{n}\right) + \log_2 n \cdot c = d + c \cdot \log_2 n \in \Theta(\log n) \end{aligned}$$



# Analyse (schlechtester Fall)

$$T(n) = \begin{cases} d & \text{falls } n = 1, \\ T(n/2) + c & \text{falls } n > 1. \end{cases}$$

**Vermutung:**  $T(n) = d + c \cdot \log_2 n$

**Beweis durch Induktion:**

- Induktionsanfang:  $T(1) = d$ .
- Hypothese:  $T(n/2) = d + c \cdot \log_2 n/2$
- Schritt ( $n/2 \rightarrow n$ )

$$T(n) = T(n/2) + c = d + c \cdot (\log_2 n - 1) + c = d + c \log_2 n.$$

# Resultat

## *Theorem 8*

*Der Algorithmus zur binären sortierten Suche benötigt im schlechtesten Fall  $\Theta(\log n)$  Elementarschritte.*

# Iterativer binärer Suchalgorithmus

**Input:** Sortiertes Array  $A$  von  $n$  Schlüsseln. Schlüssel  $b$ .

**Output:** Index des gefundenen Elements. 0, wenn erfolglos.

$l \leftarrow 1; r \leftarrow n$

**while**  $l \leq r$  **do**

$m \leftarrow \lfloor (l + r)/2 \rfloor$

**if**  $A[m] = b$  **then**

**return**  $m$

**else if**  $A[m] < b$  **then**

$l \leftarrow m + 1$

**else**

$r \leftarrow m - 1$

**return** *NotFound*;

# Korrektheit

Algorithmus bricht nur ab, falls  $A[l..r]$  leer oder  $b$  gefunden.

**Invariante:** Falls  $b$  in  $A$ , dann im Bereich  $A[l..r]$

## **Beweis durch Induktion**

- Induktionsanfang:  $b \in A[1..n]$  (oder nicht)
- Hypothese: Invariante gilt nach  $i$  Schritten
- Schritt:
  - $b < A[m] \Rightarrow b \in A[l..m - 1]$
  - $b > A[m] \Rightarrow b \in A[m + 1..r]$

# [Geht es noch besser?]

Annahme: Gleichverteilung der *Werte* im Array.

## Beispiel

Name "Becker" würde man im Telefonbuch vorne suchen. "Wawrinka" wohl ziemlich weit hinten.

Binäre Suche vergleicht immer zuerst mit der Mitte.

Binäre Suche setzt immer  $m = \lfloor l + \frac{r-l}{2} \rfloor$ .

# [Interpolationssuche]

Erwartete relative Position von  $b$  im Suchintervall  $[l, r]$

$$\rho = \frac{b - A[l]}{A[r] - A[l]} \in [0, 1].$$

Neue "Mitte":  $l + \rho \cdot (r - l)$

Anzahl Vergleiche im Mittel  $\mathcal{O}(\log \log n)$  (ohne Beweis).

Ist Interpolationssuche also immer zu bevorzugen?

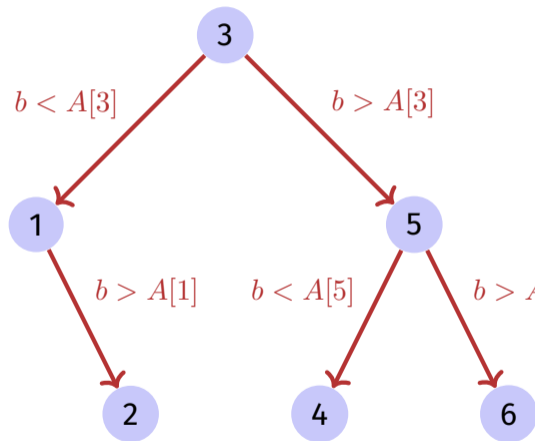
Nein: Anzahl Vergleiche im schlimmsten Fall  $\Omega(n)$ .

# Untere Schranke

Binäre Suche (im schlechtesten Fall):  $\Theta(\log n)$  viele Vergleiche.

Gilt für *jeden* Suchalgorithmus in sortiertem Array (im schlechtesten Fall):  
Anzahl Vergleiche =  $\Omega(\log n)$ ?

# Entscheidungsbaum



- Für jede Eingabe  $b = A[i]$  muss Algorithmus erfolgreich sein  $\Rightarrow$  Baum enthält mindestens  $n$  Knoten.
- Anzahl Vergleiche im schlechtesten Fall = Höhe des Baumes = maximale Anzahl Knoten von Wurzel zu Blatt.



# Entscheidungsbaum

Binärer Baum der Höhe  $h$  hat höchstens  $2^0 + 2^1 + \dots + 2^{h-1} = 2^h - 1 < 2^h$  Knoten.

$$2^h > n \Rightarrow h > \log_2 n$$

Entscheidungsbaum mit  $n$  Knoten hat mindestens Höhe  $\log_2 n$ .  
Anzahl Entscheidungen =  $\Omega(\log n)$ .

## *Theorem 9*

*Jeder Algorithmus zur vergleichsbasierten Suche in sortierten Daten der Länge  $n$  benötigt im schlechtesten Fall  $\Omega(\log n)$  Vergleichsschritte.*

# Untere Schranke für Suchen in unsortiertem Array

## *Theorem 10*

*Jeder vergleichsbasierte Algorithmus zur Suche in **un**sortierten Daten der Länge  $n$  benötigt im schlechtesten Fall  $\Omega(n)$  Vergleichsschritte.*

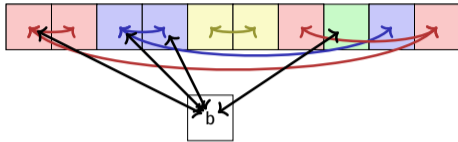
# Versuch

Korrekt?

"Beweis": Um  $b$  in  $A$  zu finden, muss  $b$  mit jedem Element  $A[i]$  ( $1 \leq i \leq n$ ) verglichen werden.

Falsch! Vergleiche zwischen Elementen von  $A$  möglich!

# Besseres Argument



- Unterteilung der Vergleiche: Anzahl Vergleiche mit  $b$ :  $e$  Anzahl Vergleiche untereinander ohne  $b$ :  $i$
- Vergleiche erzeugen  $g$  Gruppen. Initial:  $g = n$ .
- Vereinigen zweier Gruppen benötigt mindestens einen (internen Vergleich):  $n - g \leq i$ .
- Mindestens ein Element pro Gruppe muss mit  $b$  verglichen werden:  $e \geq g$ .
- Anzahl Vergleiche  $i + e \geq n - g + g = n$ . ■

## 5. Auswählen

---

Das Auswahlproblem, Randomisierte Berechnung des Medians, Lineare Worst-Case Auswahl [Ottman/Widmayer, Kap. 3.1, Cormen et al, Kap. 9]

# Das Auswahlproblem

## Eingabe

- Unsortiertes Array  $A = (A_1, \dots, A_n)$  paarweise verschiedener Werte
- Zahl  $1 \leq k \leq n$ .

Ausgabe:  $A[i]$  mit  $|\{j : A[j] < A[i]\}| = k - 1$

## Spezialfälle

$k = 1$ : Minimum: Algorithmus mit  $n$  Vergleichsoperationen trivial.

$k = n$ : Maximum: Algorithmus mit  $n$  Vergleichsoperationen trivial.

$k = \lfloor n/2 \rfloor$ : Median.

# Naiver Algorithmus

Wiederholt das Minimum entfernen / auslesen:  $\Theta(k \cdot n)$ .

→ Median in  $\Theta(n^2)$

# Min und Max

❓ Separates Finden von Minimum und Maximum in  $(A[1], \dots, A[n])$  benötigt insgesamt  $2n$  Vergleiche. (Wie) geht es mit weniger als  $2n$  Vergleichen für beide gemeinsam?

❗ Es geht mit  $\frac{3}{2}n$  Vergleichen: Vergleiche jeweils 2 Elemente und deren kleineres mit Min und grösseres mit Max.<sup>5</sup> Possible with  $\frac{3}{2}n$  comparisons: compare 2 elements each and then the smaller one with min and the greater one with max.<sup>6</sup>

---

<sup>5</sup>Das liefert einen Hinweis darauf, dass der naive Algorithmus verbessert werden kann.

<sup>6</sup>An indication that the naive algorithm can be improved.

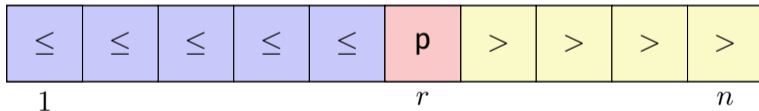


# Bessere Ansätze

- Sortieren (kommt bald):  $\Theta(n \log n)$
- Pivotieren:  $\Theta(n)$  !

# Pivotieren

1. Wähle ein (beliebiges) Element  $p$  als Pivotelement
2. Teile  $A$  in zwei Teile auf, bestimme dabei den Rang von  $p$ , indem die Anzahl der Indizes  $i$  mit  $A[i] \leq p$  gezählt werden.
3. Rekursion auf dem relevanten Teil. Falls  $k = r$ , dann gefunden.



# Algorithmus Partition( $A, l, r, p$ )

**Input:** Array  $A$ , welches den Pivot  $p$  in  $A[l, \dots, r]$  mindestens einmal enthält.

**Output:** Array  $A$  partitioniert in  $A[l, \dots, r]$  um  $p$ . Rückgabe der Position von  $p$ .

**while**  $l \leq r$  **do**

**while**  $A[l] < p$  **do**

$l \leftarrow l + 1$

**while**  $A[r] > p$  **do**

$r \leftarrow r - 1$

    swap( $A[l], A[r]$ )

**if**  $A[l] = A[r]$  **then**

$l \leftarrow l + 1$

**return**  $l-1$

# Korrektheit: Invariante

Invariante  $I$ :  $A_i \leq p \forall i \in [0, l), A_i \geq p \forall i \in (r, n], \exists k \in [l, r] : A_k = p$ .

**while**  $l \leq r$  **do**

**while**  $A[l] < p$  **do**

$l \leftarrow l + 1$

**while**  $A[r] > p$  **do**

$r \leftarrow r - 1$

$\text{swap}(A[l], A[r])$

**if**  $A[l] = A[r]$  **then**

$l \leftarrow l + 1$

$I$

$I$  und  $A[l] \geq p$

$I$  und  $A[r] \leq p$

$I$  und  $A[l] \leq p \leq A[r]$

$I$

**return**  $l-1$

# Korrektheit: Fortschritt

**while**  $l \leq r$  **do**

**while**  $A[l] < p$  **do**

$l \leftarrow l + 1$

**while**  $A[r] > p$  **do**

$r \leftarrow r - 1$

    swap( $A[l]$ ,  $A[r]$ )

**if**  $A[l] = A[r]$  **then**

$l \leftarrow l + 1$

**return**  $l-1$

Fortschritt wenn  $A[l] < p$

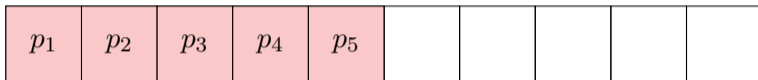
Fortschritt wenn  $A[r] > p$

Fortschritt wenn  $A[l] > p$  oder  $A[r] < p$

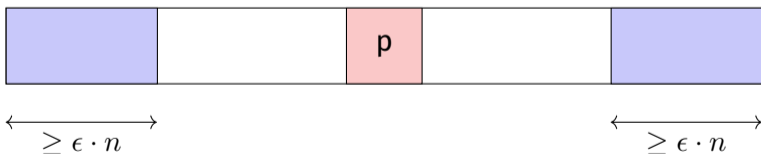
Fortschritt wenn  $A[l] = A[r] = p$

# Wahl des Pivots

Das Minimum ist ein schlechter Pivot: worst Case  $\Theta(n^2)$



Ein guter Pivot hat linear viele Elemente auf beiden Seiten.



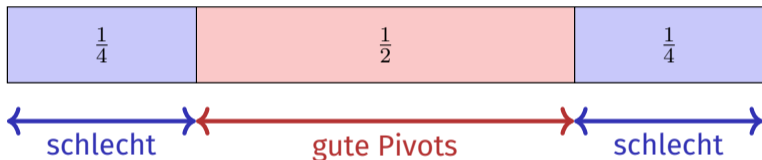
# Analyse

Unterteilung mit Faktor  $q$  ( $0 < q < 1$ ): zwei Gruppen mit  $q \cdot n$  und  $(1 - q) \cdot n$  Elementen (ohne Einschränkung  $q \geq 1 - q$ ).

$$\begin{aligned}T(n) &\leq T(q \cdot n) + c \cdot n \\&\leq c \cdot n + q \cdot c \cdot n + T(q^2 \cdot n) \leq \dots = c \cdot n \sum_{i=0}^{\log_q(n)-1} q^i + T(1) \\&\leq c \cdot n \underbrace{\sum_{i=0}^{\infty} q^i}_{\text{geom. Reihe}} + d = c \cdot n \cdot \frac{1}{1 - q} + d = \mathcal{O}(n)\end{aligned}$$

# Wie bekommen wir das hin?

Der Zufall hilft uns (Tony Hoare, 1961). Wähle in jedem Schritt einen zufälligen Pivot.



Wahrscheinlichkeit für guten Pivot nach einem Versuch:  $\frac{1}{2} =: \rho$ .

Wahrscheinlichkeit für guten Pivot nach  $k$  Versuchen:  $(1 - \rho)^{k-1} \cdot \rho$ .

Erwartete Anzahl Versuche:  $1/\rho = 2$  (Erwartungswert der geometrischen Verteilung:)



# Algorithmus Quickselect ( $A, l, r, k$ )

**Input:** Array  $A$  der Länge  $n$ . Indizes  $1 \leq l \leq k \leq r \leq n$ , so dass für alle  $x \in A[l..r]$  :  $|\{j|A[j] \leq x\}| \geq l$  und  $|\{j|A[j] \leq x\}| \leq r$ .

**Output:** Wert  $x \in A[l..r]$  mit  $|\{j|A[j] \leq x\}| \geq k$  und  $|\{j|x \leq A[j]\}| \geq n - k + 1$

**if**  $l=r$  **then**

  | return  $A[l]$ ;

$x \leftarrow$  RandomPivot( $A, l, r$ )

$m \leftarrow$  Partition( $A, l, r, x$ )

**if**  $k < m$  **then**

  | return QuickSelect( $A, l, m - 1, k$ )

**else if**  $k > m$  **then**

  | return QuickSelect( $A, m + 1, r, k$ )

**else**

  | return  $A[k]$

# Algorithmus RandomPivot ( $A, l, r$ )

**Input:** Array  $A$  der Länge  $n$ . Indizes  $1 \leq l \leq r \leq n$

**Output:** Zufälliger "guter" Pivot  $x \in A[l, \dots, r]$

**repeat**

    wähle zufälligen Pivot  $x \in A[l..r]$

$p \leftarrow l$

**for**  $j = l$  **to**  $r$  **do**

**if**  $A[j] \leq x$  **then**  $p \leftarrow p + 1$

**until**  $\lfloor \frac{3l+r}{4} \rfloor \leq p \leq \lceil \frac{l+3r}{4} \rceil$

**return**  $x$

*Dieser Algorithmus ist nur von theoretischem Interesse und liefert im Erwartungswert nach 2 Durchläufen einen guten Pivot. Praktisch kann man im Algorithmus Quickselect direkt einen zufälligen Pivot uniformverteilt ziehen oder einen deterministischen Pivot wählen, z.B. den Median von drei Elementen.*

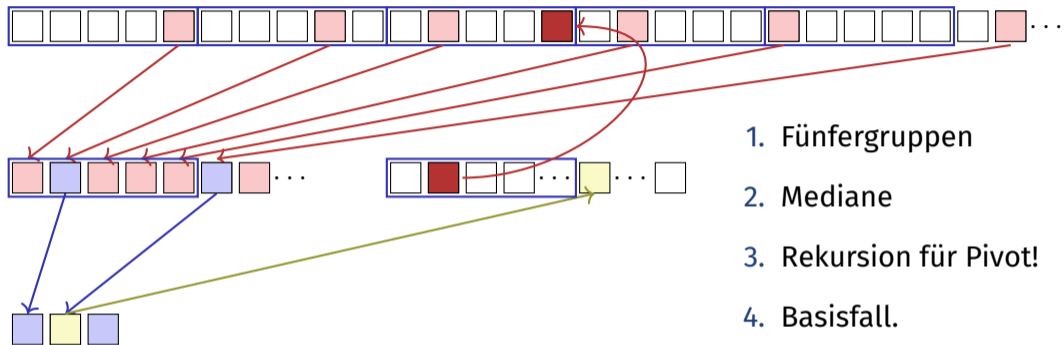
# Median der Mediane

Ziel: Finde einen Algorithmus, welcher im schlechtesten Fall nur linear viele Schritte benötigt.

Algorithmus Select ( $k$ -smallest)

- Fünfergruppen bilden.
- Median jeder Gruppe bilden (naiv).
- Select rekursiv auf den Gruppenmedianen.
- Partitioniere das Array um den gefundenen Median der Mediane.  
Resultat:  $i$
- Wenn  $i = k$ , Resultat. Sonst: Select rekursiv auf der richtigen Seite.

# Median der Mediane



1. Fünfergruppen
2. Mediane
3. Rekursion für Pivot!
4. Basisfall.
5. Pivot (Level 1)
6. Partition (Level 1)
7. Median = Pivot Level 0
8. 2. Rekursion startet

# Algorithmus MMSelect( $A, l, r, k$ )

**Input:** Array  $A$  der Länge  $n$  mit paarweise verschiedenen Einträgen.

$$1 \leq l \leq k \leq r \leq n, A[i] < A[k] \forall 1 \leq i < l, A[i] > A[k] \forall r < i \leq n$$

**Output:** Wert  $x \in A$  mit  $|\{j | A[j] \leq x\}| = k$

$m \leftarrow$  MMChoose( $A, l, r$ )

$i \leftarrow$  Partition( $A, l, r, m$ )

**if**  $k < i$  **then**

    | **return** MMSelect( $A, l, i - 1, k$ )

**else if**  $k > i$  **then**

    | **return** MMSelect( $A, i + 1, r, k$ )

**else**

    | **return**  $A[i]$

# Algorithmus MMChoose( $A, l, r$ )

**Input:** Array  $A$  der Länge  $n$  mit paarweise verschiedenen Einträgen.

$$1 \leq l \leq r \leq n.$$

**Output:** Median  $m$  der Mediane

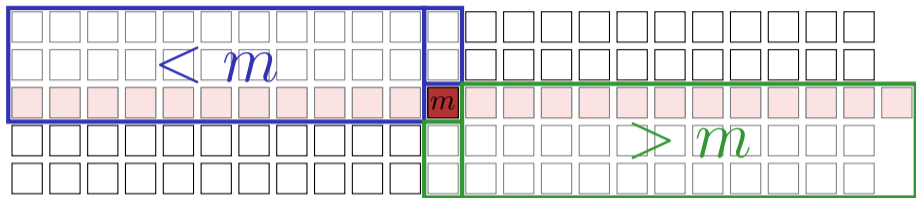
**if**  $r - l \leq 5$  **then**

| return MedianOf5( $A[l, \dots, r]$ )

**else**

|  $A' \leftarrow$  MedianOf5Array( $A[l, \dots, r]$ )  
| **return** MMSelect( $A', 1, |A'|, \lfloor \frac{|A'|}{2} \rfloor$ )

# Was bringt das?



- Anzahl Fünfergruppen:  $\lceil \frac{n}{5} \rceil$ , ohne Mediengruppe:  $\lceil \frac{n}{5} \rceil - 1$
- Minimale Anzahl Gruppen links / rechts von Mediengruppe  $\lfloor \frac{1}{2} (\lceil \frac{n}{5} \rceil - 1) \rfloor$
- Minimale Anzahl Punkte kleiner / grösser als  $m$

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

(Fülle Restgruppe konzeptuell mit Punkten aus Mediengruppe auf)

⇒ Rekursiver Aufruf mit maximal  $\lceil \frac{7n}{10} + 6 \rceil$  Elementen.

# Analyse

Rekursionsungleichung:

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\left\lceil \frac{7n}{10} + 6 \right\rceil\right) + d \cdot n.$$

mit einer Konstanten  $d$ .

Behauptung:

$$T(n) = \mathcal{O}(n).$$



# Beweis

Induktionsanfang:<sup>7</sup> Wähle  $c$  so gross, dass

$$T(n) \leq c \cdot n \text{ für alle } n \leq n_0.$$

Induktionsannahme :  $H(n)$

$$T(i) \leq c \cdot i \text{ für alle } i < n.$$

Induktionsschritt:  $H(k)_{k < n} \rightarrow H(n)$

$$\begin{aligned} T(n) &\leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\left\lceil \frac{7n}{10} + 6 \right\rceil\right) + d \cdot n \\ &\leq c \cdot \left\lceil \frac{n}{5} \right\rceil + c \cdot \left\lceil \frac{7n}{10} + 6 \right\rceil + d \cdot n \quad (\text{für } n > 20). \end{aligned}$$

---

<sup>7</sup>Es wird sich im Induktionsschritt herausstellen, dass der Basisfall für alle  $n \leq n_0$  und ein bestimmtes (aber festes)  $n_0 > 0$  betrachtet werden muss. Da  $c$  beliebig gross gewählt werden darf und eine beschränkte Anzahl von Termen eingeht, ist das eine einfache Erweiterung des Basisfalles  $n = 1$

# Beweis

Induktionsschritt:

$$\begin{aligned} T(n) &\stackrel{n > 20}{\leq} c \cdot \left\lceil \frac{n}{5} \right\rceil + c \cdot \left\lceil \frac{7n}{10} + 6 \right\rceil + d \cdot n \\ &\leq c \cdot \frac{n}{5} + c + c \cdot \frac{7n}{10} + 6c + c + d \cdot n = \frac{9}{10} \cdot c \cdot n + 8c + d \cdot n. \end{aligned}$$

Zu zeigen

$$\exists n_0, \exists c \quad \left| \quad \frac{9}{10} \cdot c \cdot n + 8c + d \cdot n \leq cn \quad \forall n \geq n_0 \right.$$

Also

$$8c + d \cdot n \leq \frac{1}{10}cn \quad \Leftrightarrow \quad n \geq \frac{80c}{c - 10d}$$

Setze z.B.  $c = 90d, n_0 = 91 \quad \Rightarrow \quad T(n) \leq cn \quad \forall n \geq n_0$



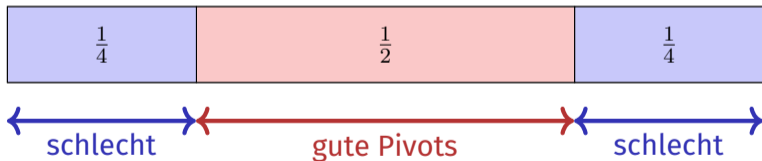
# Resultat

## *Theorem 11*

*Das  $i$ -te Element einer Folge von  $n$  Elementen kann im schlechtesten Fall in  $\Theta(n)$  Schritten gefunden werden.*

# Überblick

- |                                     |                                      |
|-------------------------------------|--------------------------------------|
| 1. Wiederholt Minimum finden        | $\mathcal{O}(n^2)$                   |
| 2. Sortieren und $A[i]$ ausgeben    | $\mathcal{O}(n \log n)$              |
| 3. Quickselect mit zufälligem Pivot | $\mathcal{O}(n)$ im Mittel           |
| 4. Median of Medians (Blum)         | $\mathcal{O}(n)$ im schlimmsten Fall |



## 5.1 Anhang

---

Herleitung einiger mathematischen Formeln

# [Erwartungswert der geometrischen Verteilung]

Zufallsvariable  $X \in \mathbb{N}^+$  mit  $\mathbb{P}(X = k) = (1 - p)^{k-1} \cdot p$ .

Erwartungswert

$$\begin{aligned}\mathbb{E}(X) &= \sum_{k=1}^{\infty} k \cdot (1 - p)^{k-1} \cdot p = \sum_{k=1}^{\infty} k \cdot q^{k-1} \cdot (1 - q) \\ &= \sum_{k=1}^{\infty} k \cdot q^{k-1} - k \cdot q^k = \sum_{k=0}^{\infty} (k + 1) \cdot q^k - k \cdot q^k \\ &= \sum_{k=0}^{\infty} q^k = \frac{1}{1 - q} = \frac{1}{p}.\end{aligned}$$