2.4 Suchen

2.4.1 Selektion

2.4.2 Hashing

2.4 Suchen

k-Selektion

- Selektions-Problem
  - gegeben:
    - n (verschiedene) Zahlen $A[1..n]$
    - eine Zahl k mit $1 \leqslant k \leqslant n$
  - gesucht: k-kleinstes Element von A, d.h. dasjenige Element $x \in A$ mit $\#\{ i : A[i] < x \} = k-1$

Man will nicht nach bestimmten Schlüsseln suchen, sondern Fragen beantworten wie „was ist das größte/kleinste Element“, etc.

k-Selektion

- Man nennt das k-kleinste Element
  - für $k=1$ das Minimum
  - für $k=\lfloor (n+1)/2 \rfloor$, $k=\lceil (n+1)/2 \rceil$ den unteren, oberen Median
  - für $k=n$ das Maximum

Suche nach Median ist in naiver Implementierung in O(n^2). Hierfür werden wir eine intelligentere Implementierung finden.

k-Selektion: Beispiel

A[1..10] = 41, 76, 32, 62, 21, 52, 19, 83, 0, 91

Minimum: 0

Maximum: 91

Unterer Median: 41, denn

#{ i : A[i] < 41 } = {3, 5, 7, 9} = 4 = ⌊(10+1)/2⌋ − 1

Oberer Median: 52, denn

#{ i : A[i] < 52 } = {1, 3, 5, 7, 9} = 5 = ⌈(10+1)/2⌉ − 1

Selektion durch Sortieren

- SortSelect(A,k)

  sort(A)

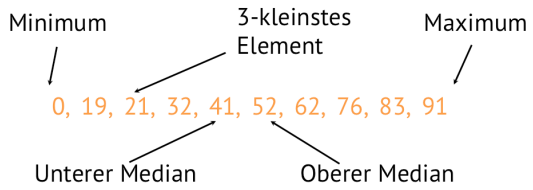
  return A[k]
- Aufwand = Aufwand für Sortierung

  T(n) = O(n×log n)

k-Selektion: Beispiel

$A[1..10] = 41, 76, 32, 62, 21, 52, 19, 83, 0, 91$

Sortieren liefert:



2.4 Suchen

Selektion durch Partitionierung

- PartitionSelect(A[1..n], l, r, k)
  - if l = r then
    - return A[l]
  - m ⟵ Partition(A, l, r)
  - i ⟵ m − l + 1
  - if k = i then
    - return A[m]
  - elseif k < i then
    - return PartitionSelect(A, l, m−1, k)
  - else
    - return PartitionSelect(A,m+1,r,k−i)

Berechne das k-kleinste Element von A[l..r]

Precondition: $1 \leqslant k \leqslant r - l + 1$

Initialer Aufruf mit l=1, r=n

Selektion durch Partitionierung

- Aufwand
  - Worst case (vgl. Quick-Sort)
    $T(n) = O(n^2)$
  - Average case (ohne Beweis)
    $T(n) = O(n)$
  - anschaulich: $n/2 + n/4 + \ldots \leqslant n$

Avg. Case: nach jedem Schritt muss nur noch etwa die Hälfte der Elemente angeschaut werden

Beispiel

1	2	3	4	5	6	7	8	9
2	9	8	3	7	5	1	6	4

k = 6

Beispiel

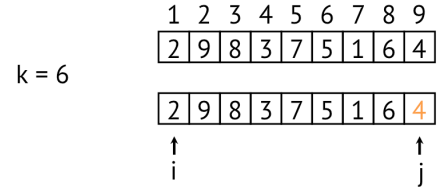
1	2	3	4	5	6	7	8	9
2	9	8	3	7	5	1	6	4

Pivot = 4

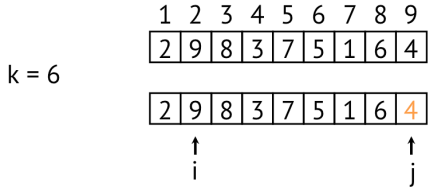
k = 6

2	9	8	3	7	5	1	6	4
↑								↑
i								j

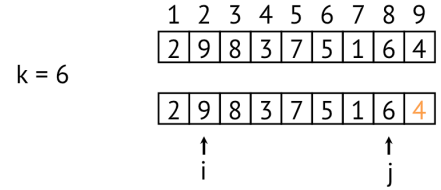
Beispiel



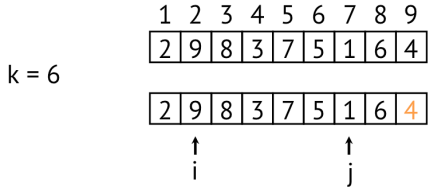
Beispiel



Beispiel



Beispiel



Beispiel

1	2	3	4	5	6	7	8	9
2	9	8	3	7	5	1	6	4

k = 6

2	1	8	3	7	5	9	6	4

i j

Beispiel

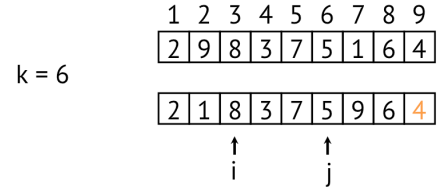
1	2	3	4	5	6	7	8	9
2	9	8	3	7	5	1	6	4

k = 6

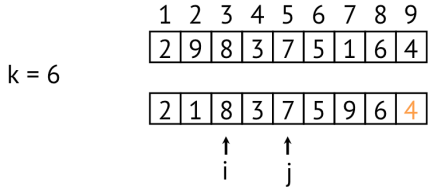
2	1	8	3	7	5	9	6	4

i j

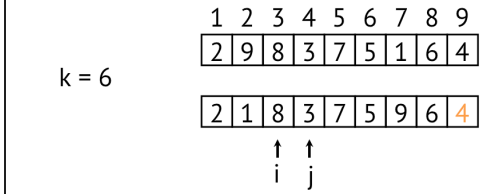
Beispiel



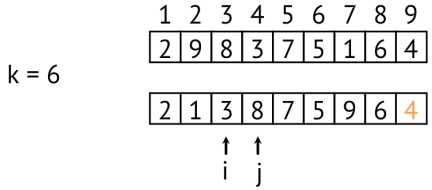
Beispiel



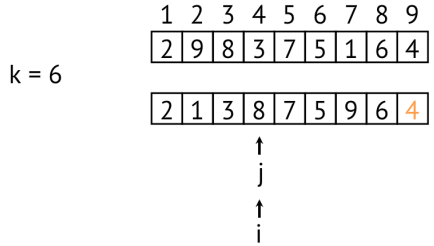
Beispiel



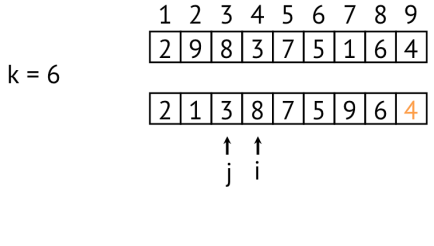
Beispiel



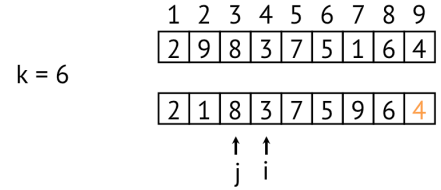
Beispiel



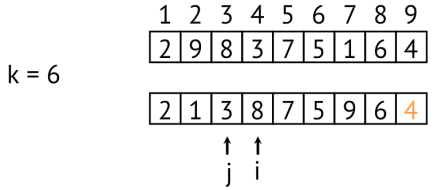
Beispiel



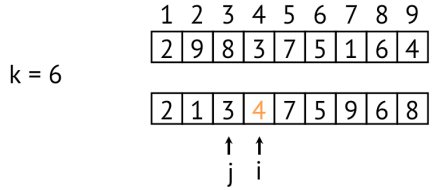
Beispiel



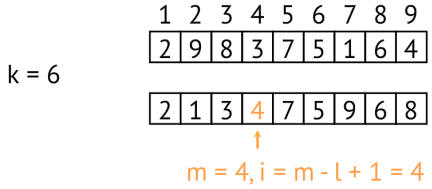
Beispiel



Beispiel



Beispiel



Beispiel

1	2	3	4	5	6	7	8	9
2	9	8	3	7	5	1	6	4

k = 6

2	1	3	4	7	5	9	6	8

Beispiel

1	2	3	4	5	6	7	8	9
2	9	8	3	7	5	1	6	4

k = 6

2	1	3	4	7	5	9	6	8

k = 2

Beispiel

1	2	3	4	5	6	7	8	9
2	9	8	3	7	5	1	6	4

k = 6

2	1	3	4	7	5	9	6	8

Pivot = 8

k = 2

Beispiel

1	2	3	4	5	6	7	8	9
2	9	8	3	7	5	1	6	4

k = 6

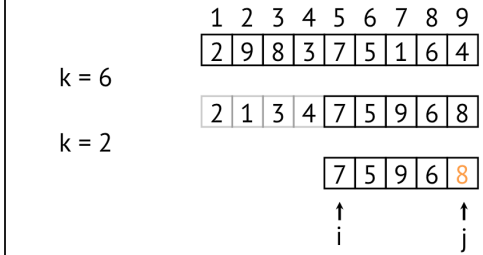
2	1	3	4	7	5	9	6	8

k = 2

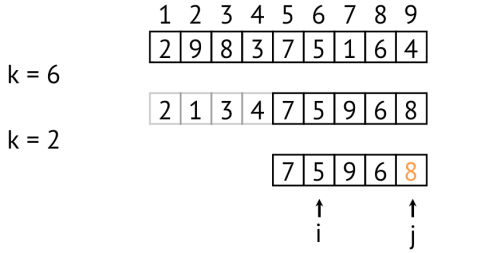
7	5	9	6	8

i j

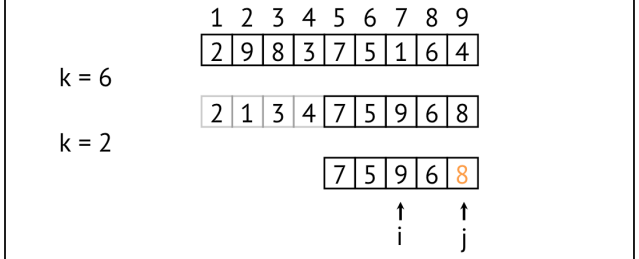
Beispiel



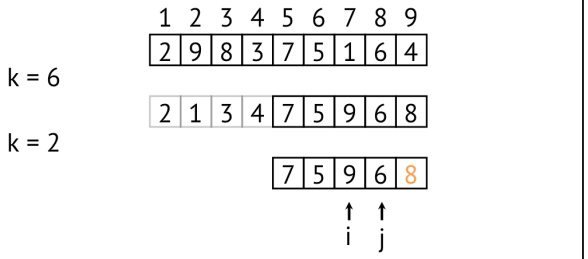
Beispiel



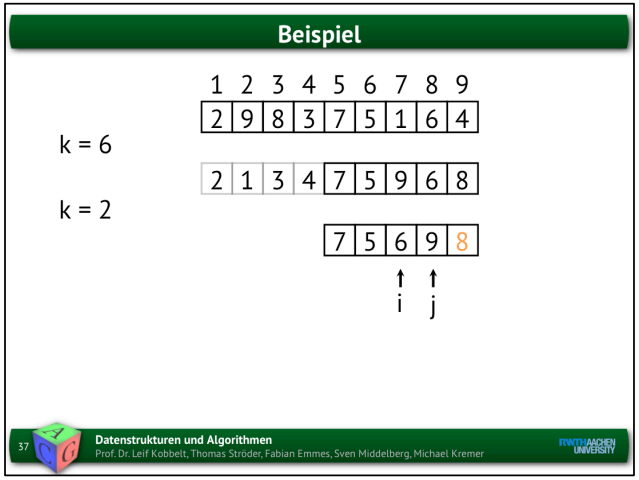
Beispiel



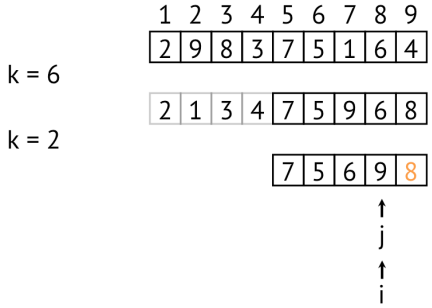
Beispiel



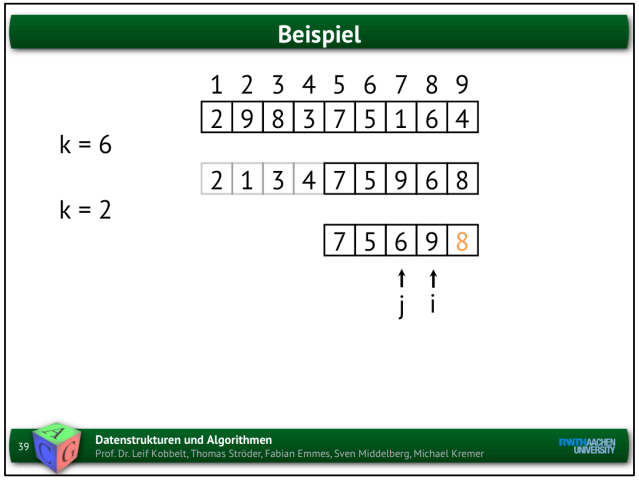
Beispiel



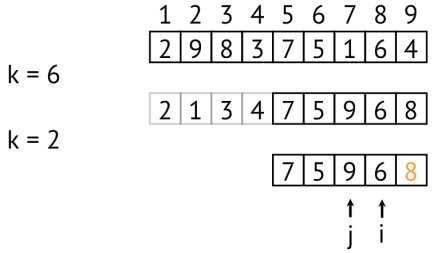
Beispiel



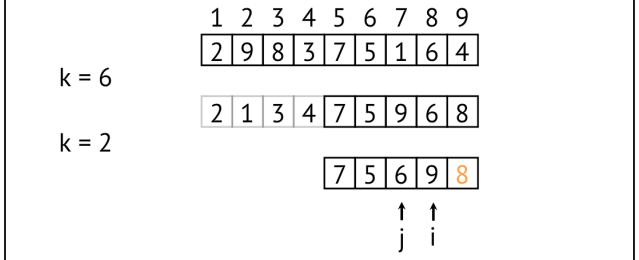
Beispiel



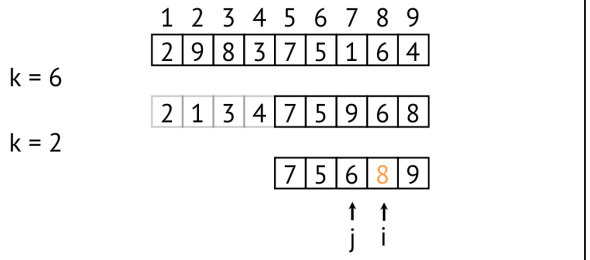
Beispiel



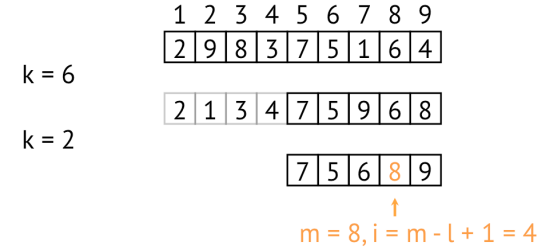
Beispiel



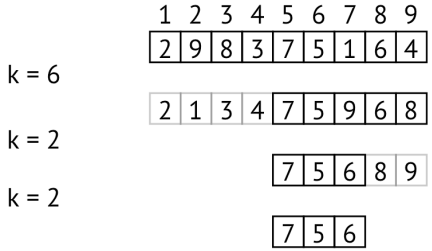
Beispiel



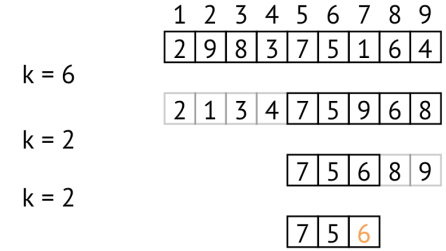
Beispiel



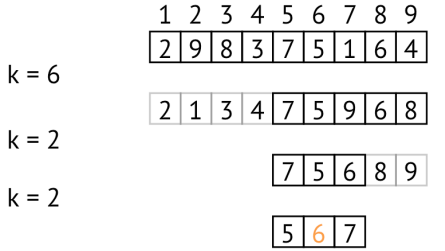
Beispiel



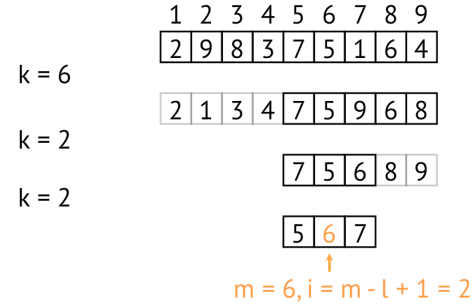
Beispiel



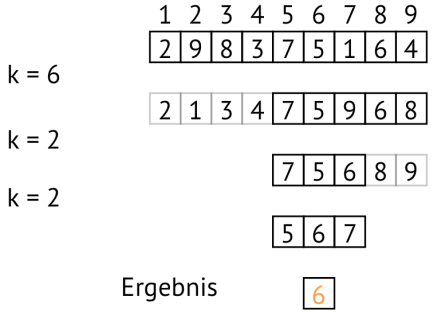
Beispiel



Beispiel



Beispiel



2.4 Suchen

Median der Mediane

- Select(A, k)
  - n ← |A|
  - if n < 50 then
    - return SortSelect(A, k)
  - M ← { SortSelect(A[5i+1]...A[5i+5], 3) : i = 0,...,⌊n/5⌋-1}
  - m ← Select(M, ⌊|M| / 2⌋)
  - $S_<$ ← { A[i] : A[i] < m }
  - $S_=$ ← { A[i] : A[i] = m }
  - $S_>$ ← { A[i] : A[i] > m }
  - if $|S_<| \geq k$ then
    - return Select($S_<$, k)
  - if $|S_<| + |S_=| = k$ then
    - return m
  - return Select($S_>$, k - $|S_<|$ - $|S_=|$)

Ähnlich PartitionSelect, allerdings mit Garantie über Größe der Aufteilung

Median der Mediane

- Select(A, k)
  n ← |A|
  if n < 50 then
    return SortSelect(A, k)
  M ← { SortSelect(A[5i+1]...A[5i+5], 3) :
        i = 0,...,⌊n/5⌋-1}
  m ← Select(M, ⌊|M| / 2⌋)
  $S_<$ ← { A[i] : A[i] < m }
  $S_=$ ← { A[i] : A[i] = m }
  $S_>$ ← { A[i] : A[i] > m }
  if $|S_<| \geq k$ then
    return Select($S_<$, k)
  if $|S_<| + |S_=| = k$ then
    return m
  return Select($S_>$, k - $|S_<|$ - $|S_=|$)

Basisfall der Rekursion, die magische Konstante wird später klar.

Median der Mediane

- Select(A, k)
  n ← |A|
  if n < 50 then
    return SortSelect(A, k)
  M ← { SortSelect(A[5i+1]...A[5i+5], 3) :
        i = 0,...,⌊n/5⌋-1}
  m ← Select(M, ⌊|M| / 2⌋)
  $S_<$ ← { A[i] : A[i] < m }
  $S_=$ ← { A[i] : A[i] = m }
  $S_>$ ← { A[i] : A[i] > m }
  if $|S_<| \geq k$ then
    return Select($S_<$, k)
  if $|S_<| + |S_=| = k$ then
    return m
  return Select($S_>$, k - $|S_<|$ - $|S_=|$)

Bilde ⌊n/5⌋ Gruppen von je 5 Elementen (es bleiben 0-4 Elemente übrig) und bestimme deren Median.

Median der Mediane

- Select(A, k)
  - n ← |A|
  - if n < 50 then
    - return SortSelect(A, k)
  - M ← { SortSelect(A[5i+1]...A[5i+5], 3) : i = 0,...,⌊n/5⌋-1}
  - m ← Select(M, ⌊|M| / 2⌋)
  - $S_<$ ← { A[i] : A[i] < m }
  - $S_=$ ← { A[i] : A[i] = m }
  - $S_>$ ← { A[i] : A[i] > m }
  - if $|S_<| \geq k$ then
    - return Select($S_<$, k)
  - if $|S_<| + |S_=| = k$ then
    - return m
  - return Select($S_>$, k - $|S_<|$ - $|S_=|$)

Bestimme rekursiv den Median der Mediane.

Median der Mediane

- Select(A, k)
  - n ← |A|
  - if n < 50 then
    - return SortSelect(A, k)
  - M ← { SortSelect(A[5i+1]...A[5i+5], 3) : i = 0,...,⌊n/5⌋-1}
  - m ← Select(M, ⌊|M| / 2⌋)
  - $S_<$ ← { A[i] : A[i] < m }
  - $S_=$ ← { A[i] : A[i] = m }
  - $S_>$ ← { A[i] : A[i] > m }
  - if $|S_<| \geq k$ then
    - return Select($S_<$, k)
  - if $|S_<| + |S_=| = k$ then
    - return m
  - return Select($S_>$, k - $|S_<|$ - $|S_=|$)

Nach Voraussetzung sind alle Elemente verschieden und damit $|S_=|=|\{m\}|=1$.

Median der Mediane

- Select(A, k)
  - n ← |A|
  - if n < 50 then
    - return SortSelect(A, k)
  - M ← { SortSelect(A[5i+1]...A[5i+5], 3) : i = 0,...,⌊n/5⌋-1}
  - m ← Select(M, ⌊|M| / 2⌋)
  - $S_<$ ← { A[i] : A[i] < m }
  - $S_=$ ← { A[i] : A[i] = m }
  - $S_>$ ← { A[i] : A[i] > m }
  - if $|S_<| \geq k$ then
    - return Select($S_<$, k)
  - if $|S_<| + |S_=| = k$ then
    - return m
  - return Select($S_>$, k - $|S_<|$ - $|S_=|$)

Das k-te Element liegt in $S_<$

Median der Mediane

- Select(A, k)
  - n ← |A|
  - if n < 50 then
    - return SortSelect(A, k)
  - M ← { SortSelect(A[5i+1]...A[5i+5], 3) : i = 0,...,⌊n/5⌋-1}
  - m ← Select(M, ⌊|M| / 2⌋)
  - $S_<$ ← { A[i] : A[i] < m }
  - $S_=$ ← { A[i] : A[i] = m }
  - $S_>$ ← { A[i] : A[i] > m }
  - if $|S_<| \geq k$ then
    - return Select($S_<$, k)
  - if $|S_<| + |S_=| = k$ then
    - return m
  - return Select($S_>$, k - $|S_<|$ - $|S_=|$)

Das k-te Element liegt in $S_==\{m\}$

Median der Mediane

- Select(A, k)
  n ← |A|
  if n < 50 then
    return SortSelect(A, k)
  M ← { SortSelect(A[5i+1]...A[5i+5], 3) :
      i = 0,...,⌊n/5⌋-1}
  m ← Select(M, ⌊|M| / 2⌋)
  $S_<$ ← { A[i] : A[i] < m }
  $S_=$ ← { A[i] : A[i] = m }
  $S_>$ ← { A[i] : A[i] > m }
  if $|S_<| \geq k$ then
    return Select($S_<$, k)
  if $|S_<| + |S_=| = k$ then
    return m
  return Select($S_>$, k - $|S_<|$ - $|S_=|$)

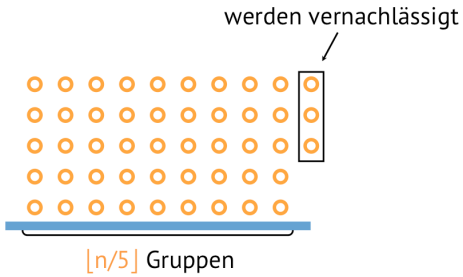
Das k-te Element ist das (k-$|S_<|$-$|S_=|$) te Element aus $S_>$

Median der Mediane: Aufwand

Ausgangs-Situation:

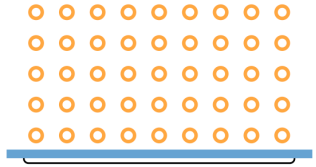
Median der Mediane: Aufwand

Anordnung in 5er Gruppen:



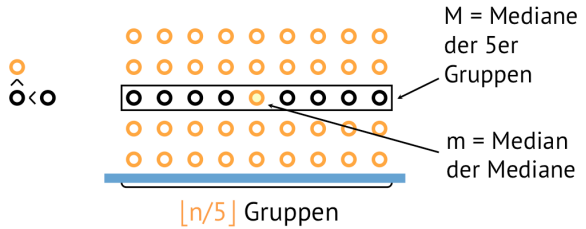
Median der Mediane: Aufwand

Anordnung in 5er Gruppen:



Median der Mediane: Aufwand

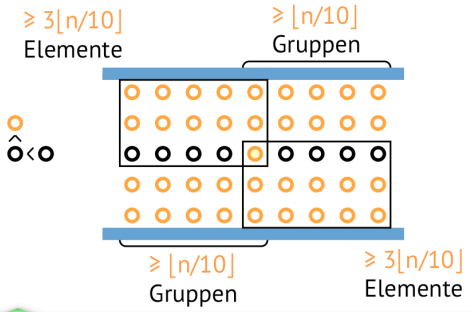
Anordnung in 5er Gruppen, sortiert dargestellt:



Konstanter Aufwand O(1) für Median der 5er Gruppen
Rekursiv auf Median der 5er Gruppen anwenden

Median der Mediane: Aufwand

Anordnung in 5er Gruppen, sortiert dargestellt:



Median der Mediane: Aufwand

- mindestens $3\lfloor n/10 \rfloor$ Elemente sind $\leqslant m$
- höchstens $n - 3\lfloor n/10 \rfloor$ Elemente sind $> m$
- für $n > 50$ ist $|S_>| < n - 3\lfloor n/10 \rfloor < 3n/4$
- für $n > 50$ ist $|S_<| < n - 3\lfloor n/10 \rfloor < 3n/4$

n=47	$n - 3\lfloor n/10 \rfloor = 35$	$3n/4 = 35.25$
n=48	$n - 3\lfloor n/10 \rfloor = 36$	$3n/4 = 36.00$
n=49	$n - 3\lfloor n/10 \rfloor = 37$	$3n/4 = 36.75$
n=50	$n - 3\lfloor n/10 \rfloor = 35$	$3n/4 = 37.50$
n=51	$n - 3\lfloor n/10 \rfloor = 36$	$3n/4 = 38.25$

Median der Aufwand

- Select(A, k)

```
n ← |A|                                              O(1)
if n < 50 then
   return SortSelect( A, k )
M ← { SortSelect( A[5i+1]...A[5i+5], 3 ) :           O(n)
      i = 0,...,⌊n/5⌋-1}
m ← Select( M, ⌊|M| / 2⌋)                            T(n/5)
S< ← { A[ i ] : A[ i ] < m }
S= ← { A[ i ] : A[ i ] = m }                         O(n)
S> ← { A[ i ] : A[ i ] > m }
if |S<| ⩾ k then
   return Select( S<, k )
if |S<| + |S=| = k then                              <T(3n/4)
   return m
return Select( S>, k - |S<| - |S=| )
```

Median der Aufwand

- Rekurrenz

$$T(n) \leq \begin{cases} c_1 & \text{für} \quad n < 50 \\ c_2 n + T(\frac{n}{5}) + T(\frac{3n}{4}) & \text{für} \quad n \geq 50 \end{cases}$$

- Behauptung: für $c=\max(c_1, 20c_2)$ ist

$T(n) < cn = O(n)$

Median der Aufwand

- Induktionsanfang: für $n<50$ okay, da $c>c_1$
- Induktionsannahme: $T(m) \leq cm$ für $m<n$
- Induktionsschluss:

$$\begin{aligned} T(n) &\leq c_2 n + T(n/5) + T(3n/4) \\ &\leq c_2 n + cn/5 + 3cn/4 \\ &\leq c_2 n + 19cn/20 \\ &\leq cn + n(c_2 - c/20) \\ &\leq cn \end{aligned}$$

$$20c_2 \leq c \Leftrightarrow c_2 \leq c/20 \Leftrightarrow c_2 - c/20 \leq 0$$

Selektion: Zusammenfassung

- Selektion durch Sortieren: $O(n \times \log n)$
- Selektion durch Partitionieren
  - Worst case: $O(n^2)$
  - Average case: $O(n)$
- Median der Mediane: $O(n)$