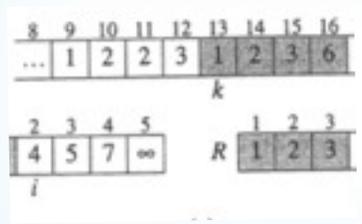
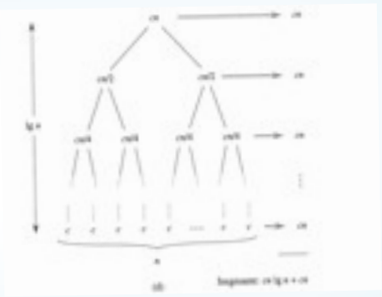




Kapitel 5.5: Quicksort

Algorithmen und Datenstrukturen
WS 2016/17

Prof. Dr. Sándor Fekete

EvaSys	Lehrevaluation Informatik Vorlesung Wintersemester 2012/2013	Electric Paper

TU Braunschweig — Prof. Dr. Sándor Fekete
Carl-Friedrich-Gauss-Fakultät - Informatik — Algorithmen und Datenstrukturen

Markieren Sie so: ☐☒☐☐☐ Bitte verwenden Sie einen Kugelschreiber oder nicht zu starken Filzstift. Dieser Fragebogen wird maschinell erfasst.
Korrektur: ☐■☐☒☐ Bitte beachten Sie im Interesse einer optimalen Datenerfassung die links gegebenen Hinweise beim Ausfüllen.

1. Persönliche Angaben

1.1 In welchem Fachsemester studieren Sie?
☐ 1./2. ☐ 3./4. ☐ 5./6.
☐ 7./8. ☐ 9./10. ☐ > 10

1.2 Welchen Abschluss streben Sie derzeit an? ☐ Diplom ☐ Bachelor ☐ Master

1.3 In welchem Studiengang studieren Sie?
☐ Informatik ☐ Nebenfach Informatik ☐ Medienwissenschaften
☐ Wirtschaftsinformatik ☐ Mobilität und Verkehr ☐ IST
☐ CSE ☐ 2-Fächer Bachelor ☐ sonstiges

2. Wie häufig waren Sie in der ...?

Große Übung: Hörsaal Kleine Übung: Seminarraum
(Bitte nur kleine und große Übungen bewerten, wenn sie angeboten wurden)

2.1	Vorlesung (Anwesenheit)	immer	☐	☐	☐	☐	☐ nie
2.2	Großen Übung (Anwesenheit)	immer	☐	☐	☐	☐	☐ nie
2.3	Kleinen Übung (Anwesenheit)	immer	☐	☐	☐	☐	☐ nie

3. Der/Die Lehrende war gut vorbereitet

3.1	Vorlesung (Vorbereitung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
3.2	Große Übung (Vorbereitung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
3.3	Kleine Übung (Vorbereitung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

4. Der/Die Lehrende wirkte kompetent

4.1	Vorlesung (Kompetenz)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
4.2	Große Übung (Kompetenz)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
4.3	Kleine Übung (Kompetenz)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

5. Der/Die Lehrende vermittelte die Inhalte verständlich

5.1	Vorlesung (Inhaltsvermittlung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
5.2	Große Übung (Inhaltsvermittlung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
5.3	Kleine Übung (Inhaltsvermittlung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

6. Zwischenfragen wurden verständlich beantwortet

6.1	Vorlesung (Zwischenfragen)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
6.2	Große Übung (Zwischenfragen)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
6.3	Kleine Übung (Zwischenfragen)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

7. Die Veranstaltung war gut strukturiert

7.1	Vorlesung (Struktur)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
7.2	Große Übung (Struktur)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
7.3	Kleine Übung (Struktur)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

8. Kann der Dozent begeistern?

8.1	Vorlesung (Begeisterung)	ja, sehr	☐	☐	☐	☐	☐ nein, gar nicht
8.2	Große Übung (Begeisterung)	ja, sehr	☐	☐	☐	☐	☐ nein, gar nicht
8.3	Kleine Übung (Begeisterung)	ja, sehr	☐	☐	☐	☐	☐ nein, gar nicht

9. Bewerten Sie die Lehrveranstaltung insgesamt!

9.1	Vorlesung (allg. Bewertung)	sehr gut	☐	☐	☐	☐	☐ sehr schlecht
9.2	Große Übung (allg. Bewertung)	sehr gut	☐	☐	☐	☐	☐ sehr schlecht
9.3	Kleine Übung (allg. Bewertung)	sehr gut	☐	☐	☐	☐	☐ sehr schlecht

EvaSys	Lehrevaluation Informatik Vorlesung Wintersemester 2012/2013	Electric Paper

TU Braunschweig — Prof. Dr. Sándor Fekete
Carl-Friedrich-Gauss-Fakultät - Informatik — Algorithmen und Datenstrukturen

Markieren Sie so: ☐☒☐☐☐ Bitte verwenden Sie einen Kugelschreiber oder nicht zu starken Filzstift. Dieser Fragebogen wird maschinell erfasst.
Korrektur: ☐■☐☒☐ Bitte beachten Sie im Interesse einer optimalen Datenerfassung die links gegebenen Hinweise beim Ausfüllen.

1. Persönliche Angaben

1.1 In welchem Fachsemester studieren Sie?
☐ 1./2. ☐ 3./4. ☐ 5./6.
☐ 7./8. ☐ 9./10. ☐ > 10

1.2 Welchen Abschluss streben Sie derzeit an? ☐ Diplom ☐ Bachelor ☐ Master

1.3 In welchem Studiengang studieren Sie?
☐ Informatik ☐ Nebenfach Informatik ☐ Medienwissenschaften
☐ Wirtschaftsinformatik ☐ Mobilität und Verkehr ☐ IST
☐ CSE ☐ 2-Fächer Bachelor ☐ sonstiges

2. Wie häufig waren Sie in der ...?

Große Übung: Hörsaal Kleine Übung: Seminarraum
(Bitte nur kleine und große Übungen bewerten, wenn sie angeboten wurden)

2.1	Vorlesung (Anwesenheit)	immer	☐	☐	☐	☐	☐ nie
2.2	Großen Übung (Anwesenheit)	immer	☐	☐	☐	☐	☐ nie
2.3	Kleinen Übung (Anwesenheit)	immer	☐	☐	☐	☐	☐ nie

3. Der/Die Lehrende war gut vorbereitet

3.1	Vorlesung (Vorbereitung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
3.2	Große Übung (Vorbereitung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
3.3	Kleine Übung (Vorbereitung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

4. Der/Die Lehrende wirkte kompetent

4.1	Vorlesung (Kompetenz)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
4.2	Große Übung (Kompetenz)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
4.3	Kleine Übung (Kompetenz)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

5. Der/Die Lehrende vermittelte die Inhalte verständlich

5.1	Vorlesung (Inhaltsvermittlung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
5.2	Große Übung (Inhaltsvermittlung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
5.3	Kleine Übung (Inhaltsvermittlung)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

6. Zwischenfragen wurden verständlich beantwortet

6.1	Vorlesung (Zwischenfragen)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
6.2	Große Übung (Zwischenfragen)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
6.3	Kleine Übung (Zwischenfragen)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

7. Die Veranstaltung war gut strukturiert

7.1	Vorlesung (Struktur)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
7.2	Große Übung (Struktur)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu
7.3	Kleine Übung (Struktur)	trifft zu	☐	☐	☐	☐	☐ trifft nicht zu

8. Kann der Dozent begeistern?

8.1	Vorlesung (Begeisterung)	ja, sehr	☐	☐	☐	☐	☐ nein, gar nicht
8.2	Große Übung (Begeisterung)	ja, sehr	☐	☐	☐	☐	☐ nein, gar nicht
8.3	Kleine Übung (Begeisterung)	ja, sehr	☐	☐	☐	☐	☐ nein, gar nicht

9. Bewerten Sie die Lehrveranstaltung insgesamt!

9.1	Vorlesung (allg. Bewertung)	sehr gut	☐	☐	☐	☐	☐ sehr schlecht
9.2	Große Übung (allg. Bewertung)	sehr gut	☐	☐	☐	☐	☐ sehr schlecht
9.3	Kleine Übung (allg. Bewertung)	sehr gut	☐	☐	☐	☐	☐ sehr schlecht

Heute!

Ankreuzliste für Übungsgruppen

Gruppe	Termin (14-täglich)	Raum	Tutor	Teilnehmer
1	Dienstag, 08:00 - 09:30	IZ 305	Sören van der Wall	[TXT]
2	Dienstag, 08:00 - 09:30	IZ 358	Jakob Keller	[TXT]
3	Dienstag, 13:15 - 14:45	IZ 358	Eva Bolle	[TXT]
4	Dienstag, 16:45 - 18:15	IZ 161	Micha Horlboge	[TXT]
4	Mittwoch, 08:00 - 09:30	IZ 358	Micha Horlboge	[TXT]
1	Mittwoch, 08:00 - 09:30	IZ 305	Sören van der Wall	[TXT]
7	Mittwoch, 13:15 - 14:45	IZ 305	Alexander Hill	[TXT]
2	Mittwoch, 15:00 - 16:30	IZ 358	Jakob Keller	[TXT]
7	Mittwoch, 15:00 - 16:30	IZ 160	Alexander Hill	[TXT]
10	Mittwoch, 16:45 - 18:15	IZ 358	Moritz Pfister	[TXT]
5	Mittwoch, 16:45 - 18:15	Container 3	Yannic Lieder	[TXT]
6	Donnerstag, 13:15 - 14:45	IZ 305	Antje Mönch	[TXT]
3	Donnerstag, 13:15 - 14:45	IZ 160	Eva Bolle	[TXT]
8	Donnerstag, 15:00 - 16:30	Container 3	Mai Hellmann	[TXT]
8	Donnerstag, 16:45 - 18:15	IZ 160	Mai Hellmann	[TXT]
10	Freitag, 11:30 - 13:00	IZ 358	Moritz Pfister	[TXT]
9	Freitag, 13:15 - 14:45	IZ 305	Dennis Luck	[TXT]
5	Freitag, 13:15 - 14:45	IZ 358	Yannic Lieder	[TXT]
9	Freitag, 15:00 - 16:30	IZ 305	Dennis Luck	[TXT]
6	Freitag, 15:00 - 16:30	IZ 358	Antje Mönch	[TXT]

5.5 Quicksort

5.5 Quicksort

Grundideen:

5.5 Quicksort

Grundideen:

- **Divide and Conquer**

5.5 Quicksort

Grundideen:

- **Divide and Conquer**
- **Jeweils Aufteilung in zwei Teilarrays**

5.5 Quicksort

<u>Grundideen:</u>

- **Divide and Conquer**
- **Jeweils Aufteilung in zwei Teilarrays**
- **Rekursiv: Sortieren der Teilarrays**

5.5 Quicksort

<u>Grundideen:</u>

- **Divide and Conquer**
- **Jeweils Aufteilung in zwei Teilarrays**
- **Rekursiv: Sortieren der Teilarrays**
- ***Kein* Merge-Schritt!**

5.5 Quicksort

Grundideen:

- **Divide and Conquer**
- **Jeweils Aufteilung in zwei Teilarrays**
- **Rekursiv: Sortieren der Teilarrays**
- ***Kein* Merge-Schritt!**
- **Stattdessen Aufteilung der Teilarrays anhand eines “Pivot”-Elements, das die Menge in kleinere und größere Elemente teilt.**

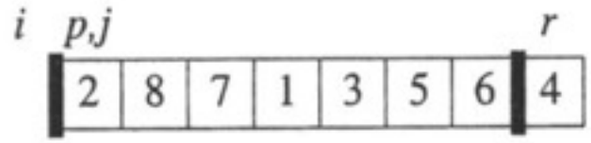
5.5 Quicksort

Grundideen:

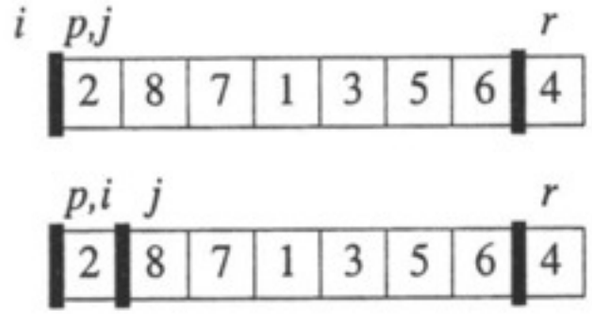
- **Divide and Conquer**
- **Jeweils Aufteilung in zwei Teilarrays**
- **Rekursiv: Sortieren der Teilarrays**
- ***Kein* Merge-Schritt!**
- **Stattdessen Aufteilung der Teilarrays anhand eines “Pivot”-Elements, das die Menge in kleinere und größere Elemente teilt.**
- **Balance der Aufteilung vorher nicht absehbar.**

5.5.1 Ablauf Quicksort

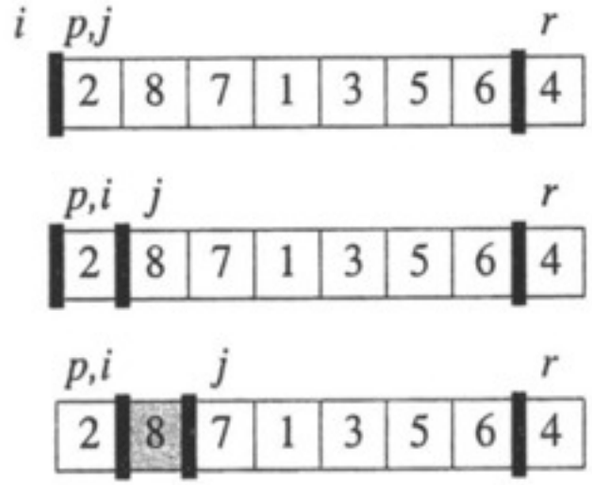
5.5.1 Ablauf Quicksort



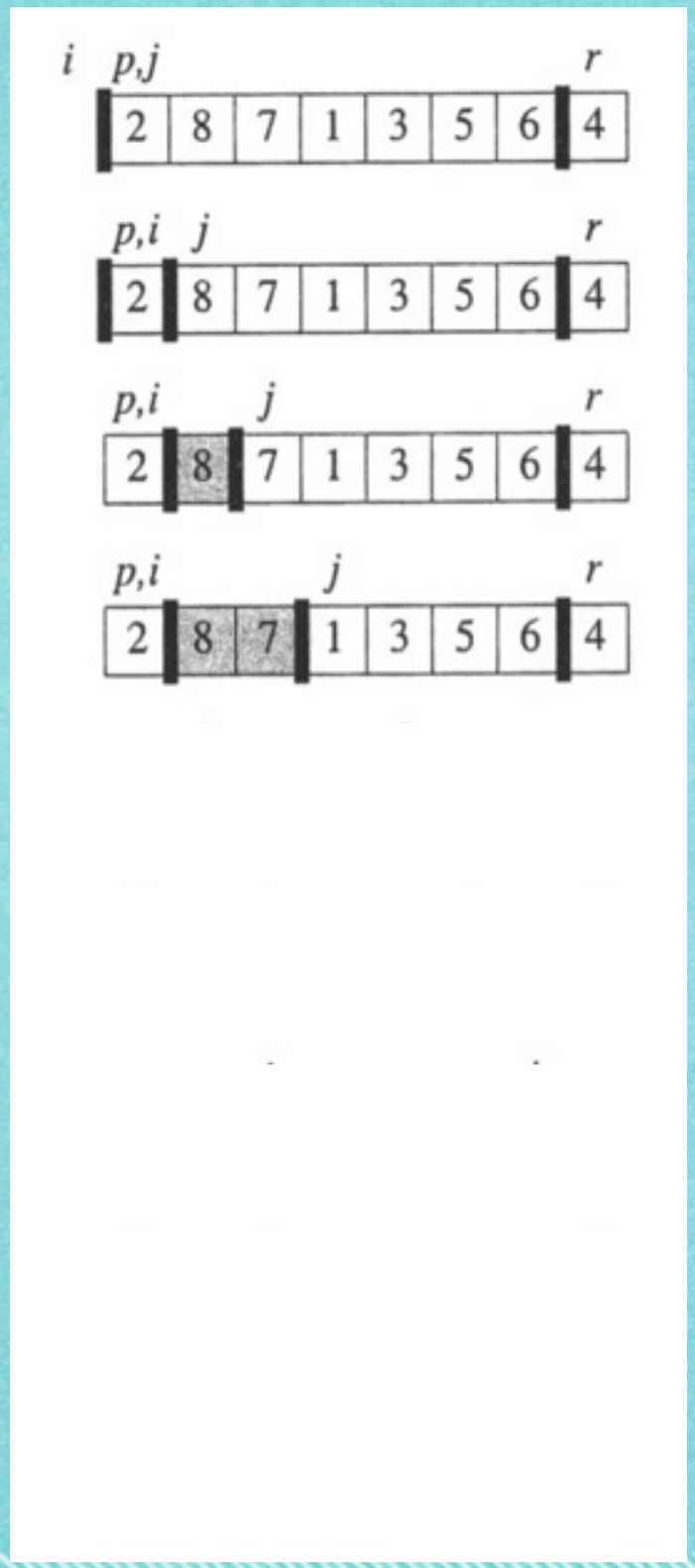
5.5.1 Ablauf Quicksort



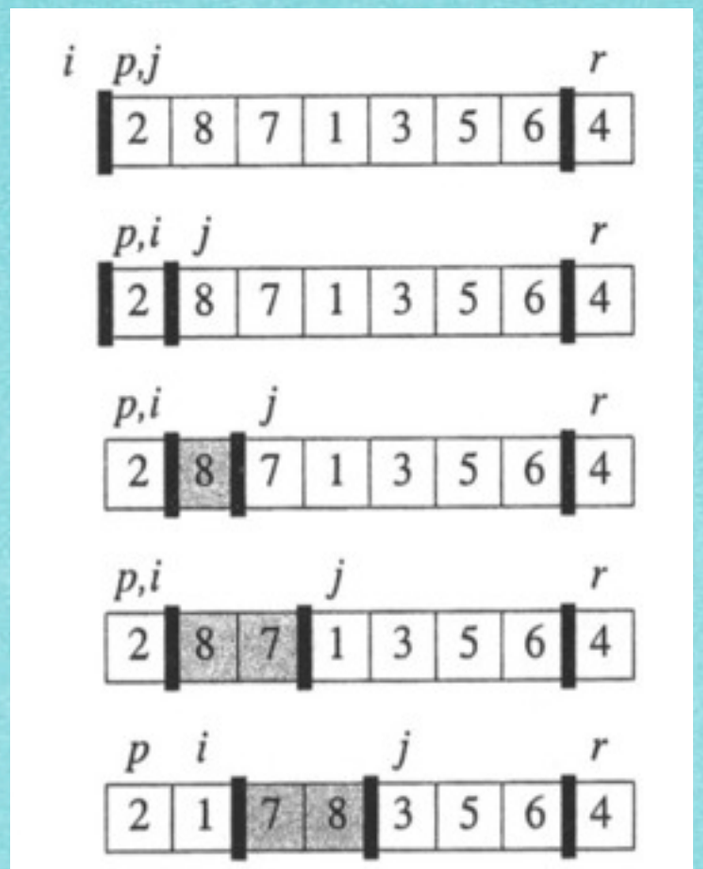
5.5.1 Ablauf Quicksort



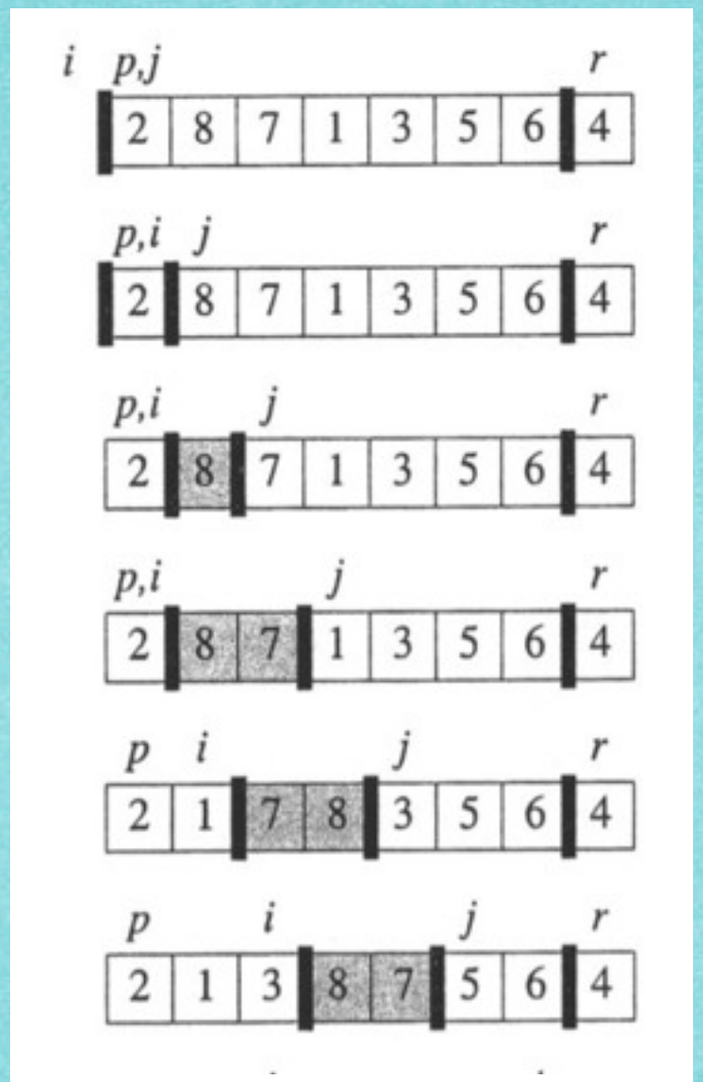
5.5.1 Ablauf Quicksort



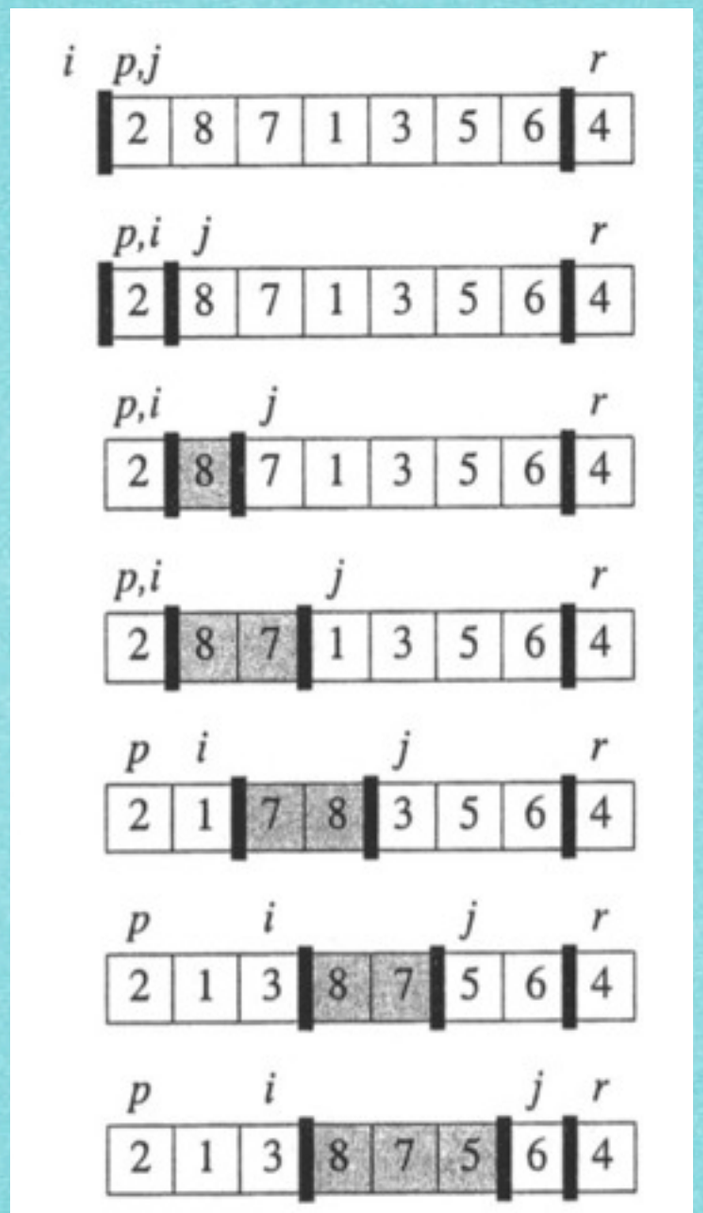
5.5.1 Ablauf Quicksort



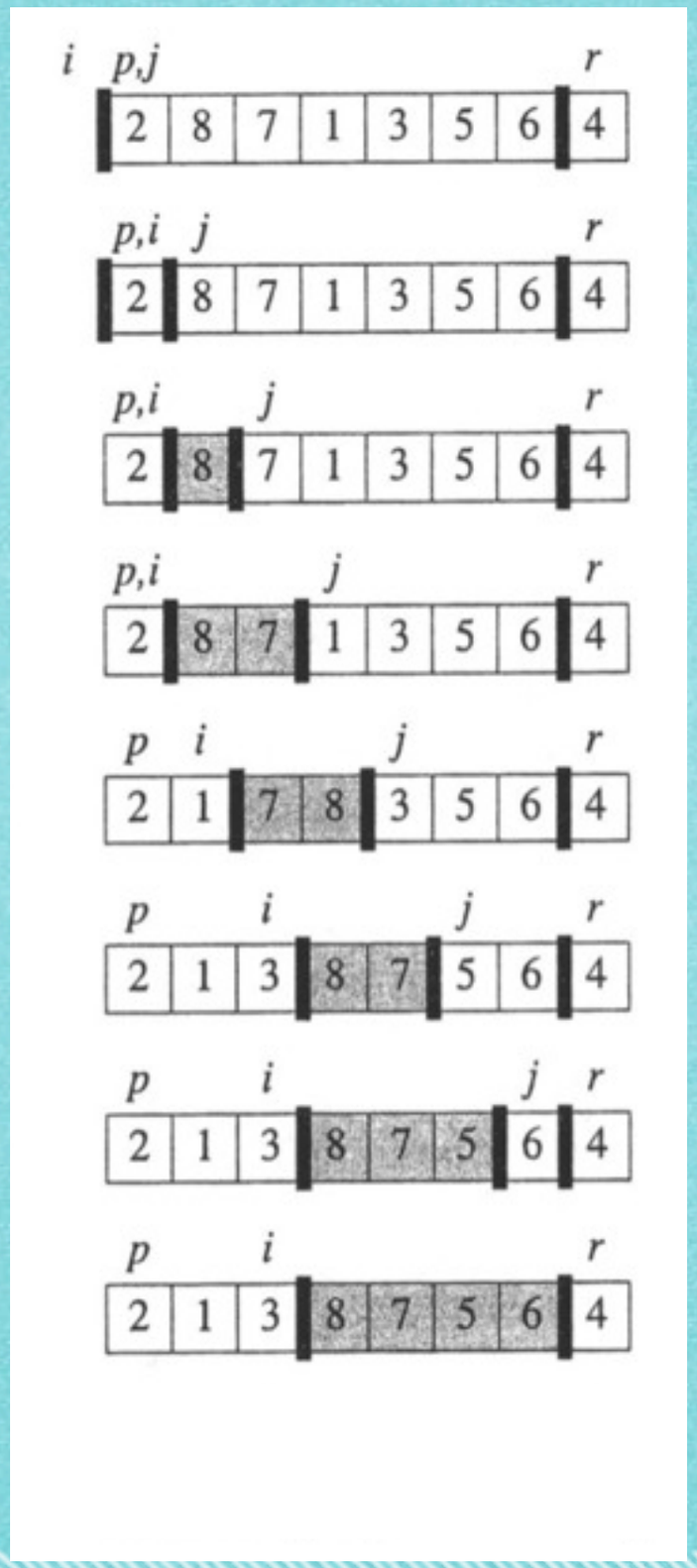
5.5.1 Ablauf Quicksort



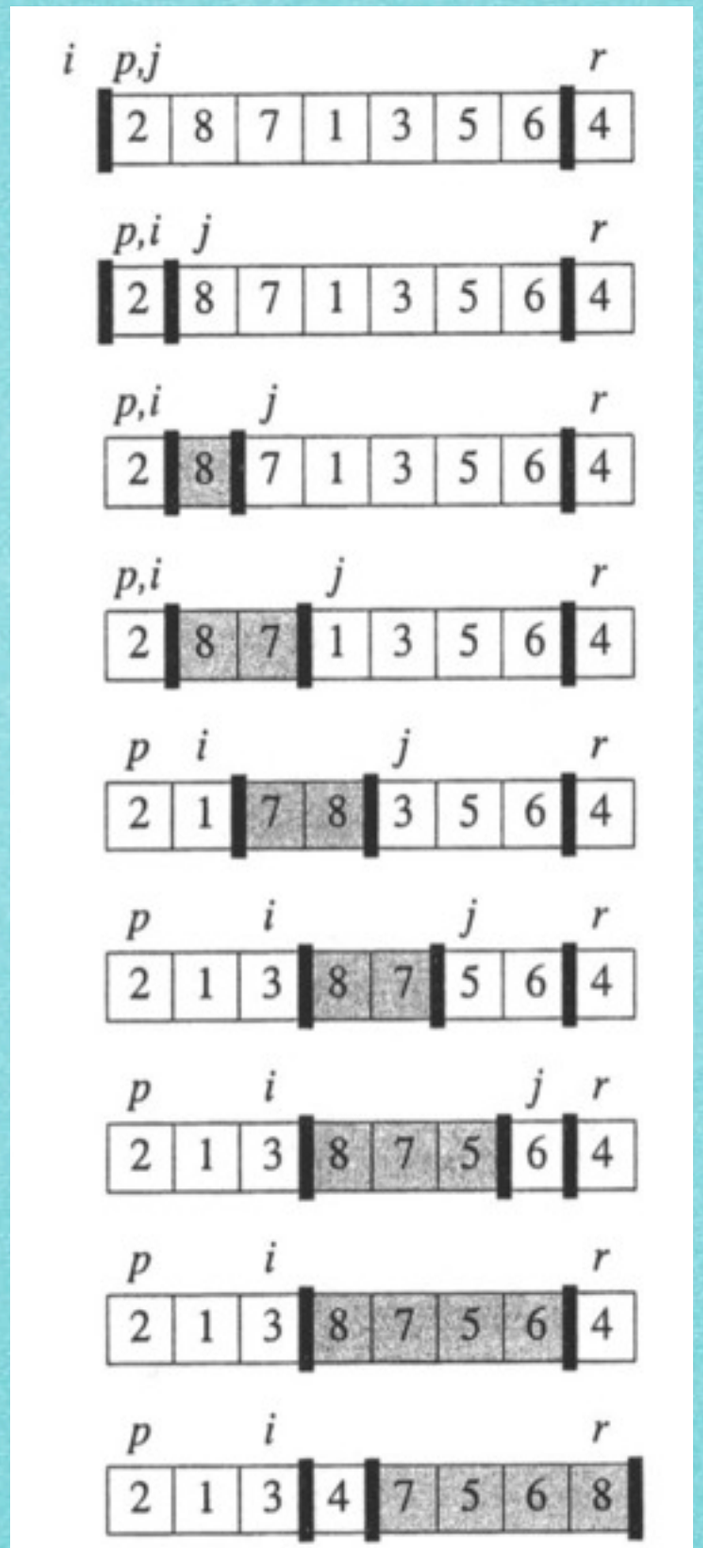
5.5.1 Ablauf Quicksort



5.5.1 Ablauf Quicksort

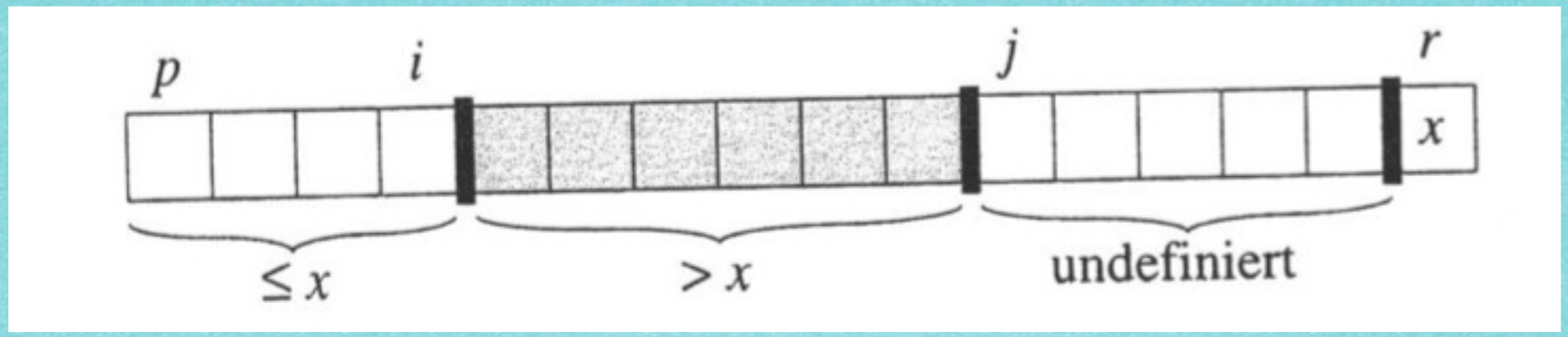


5.5.1 Ablauf Quicksort



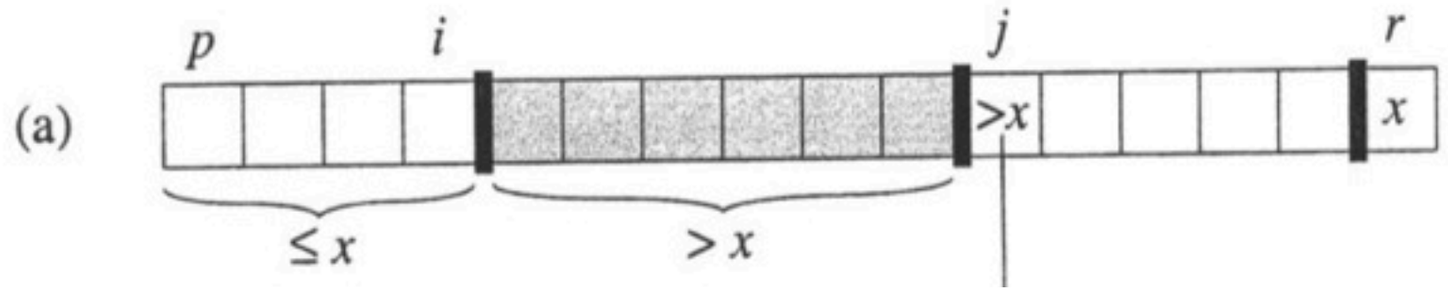
5.5.1 Ablauf Quicksort

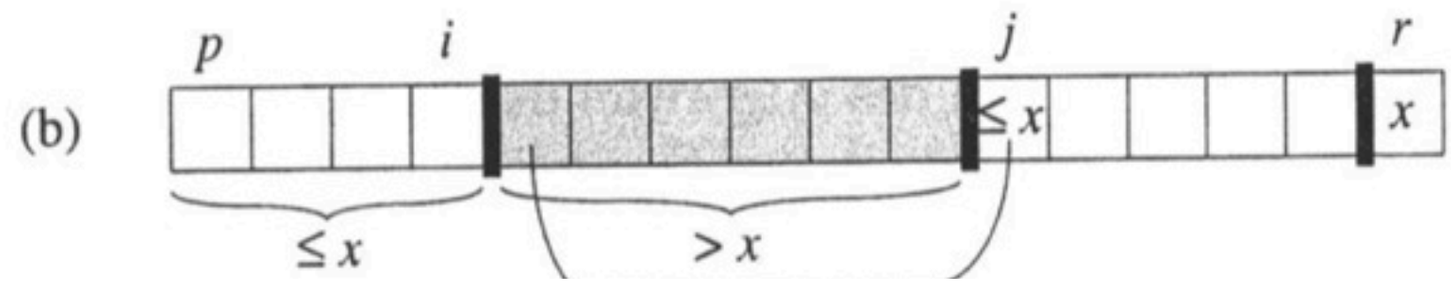
5.5.1 Ablauf Quicksort



5.5.1 Ablauf Quicksort

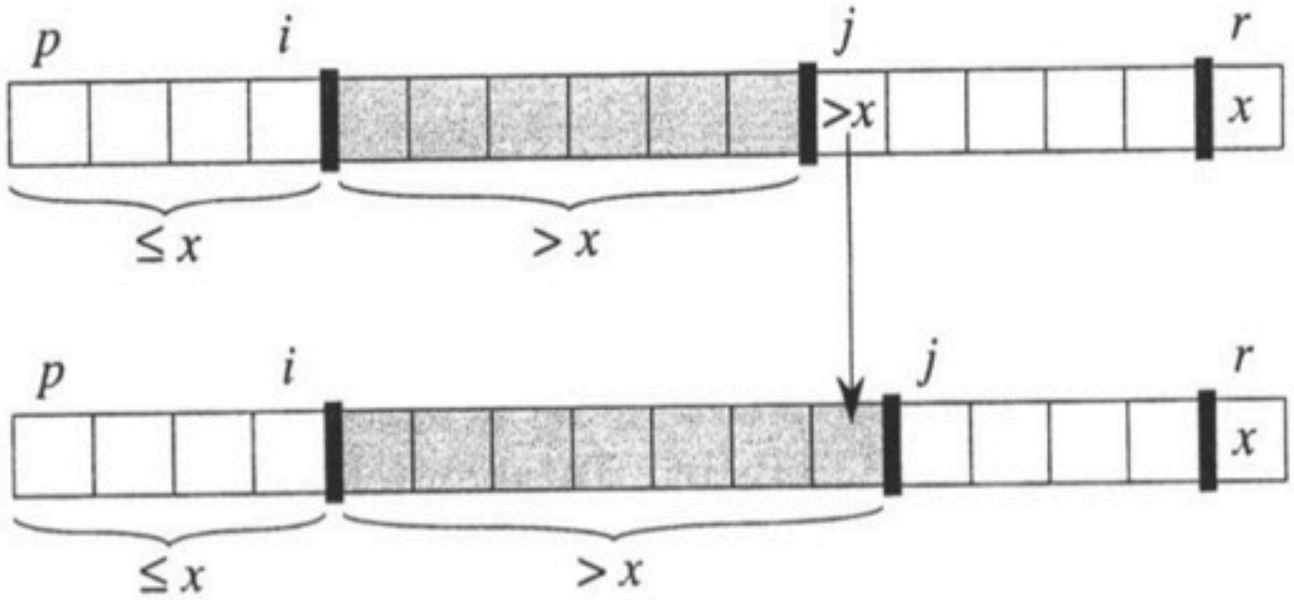
5.5.1 Ablauf Quicksort



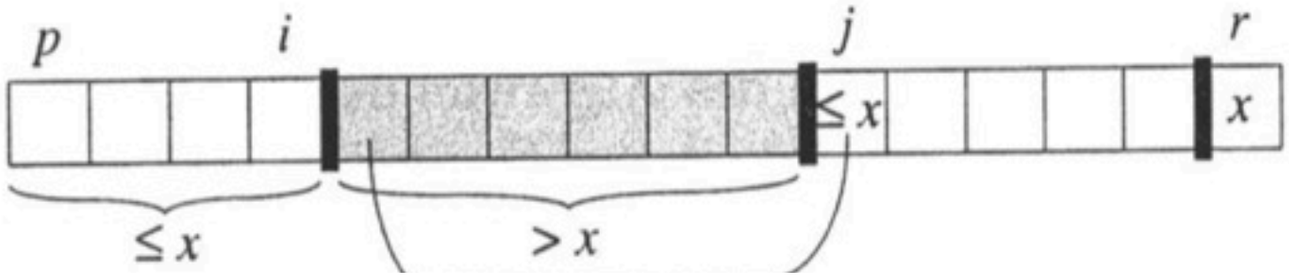


5.5.1 Ablauf Quicksort

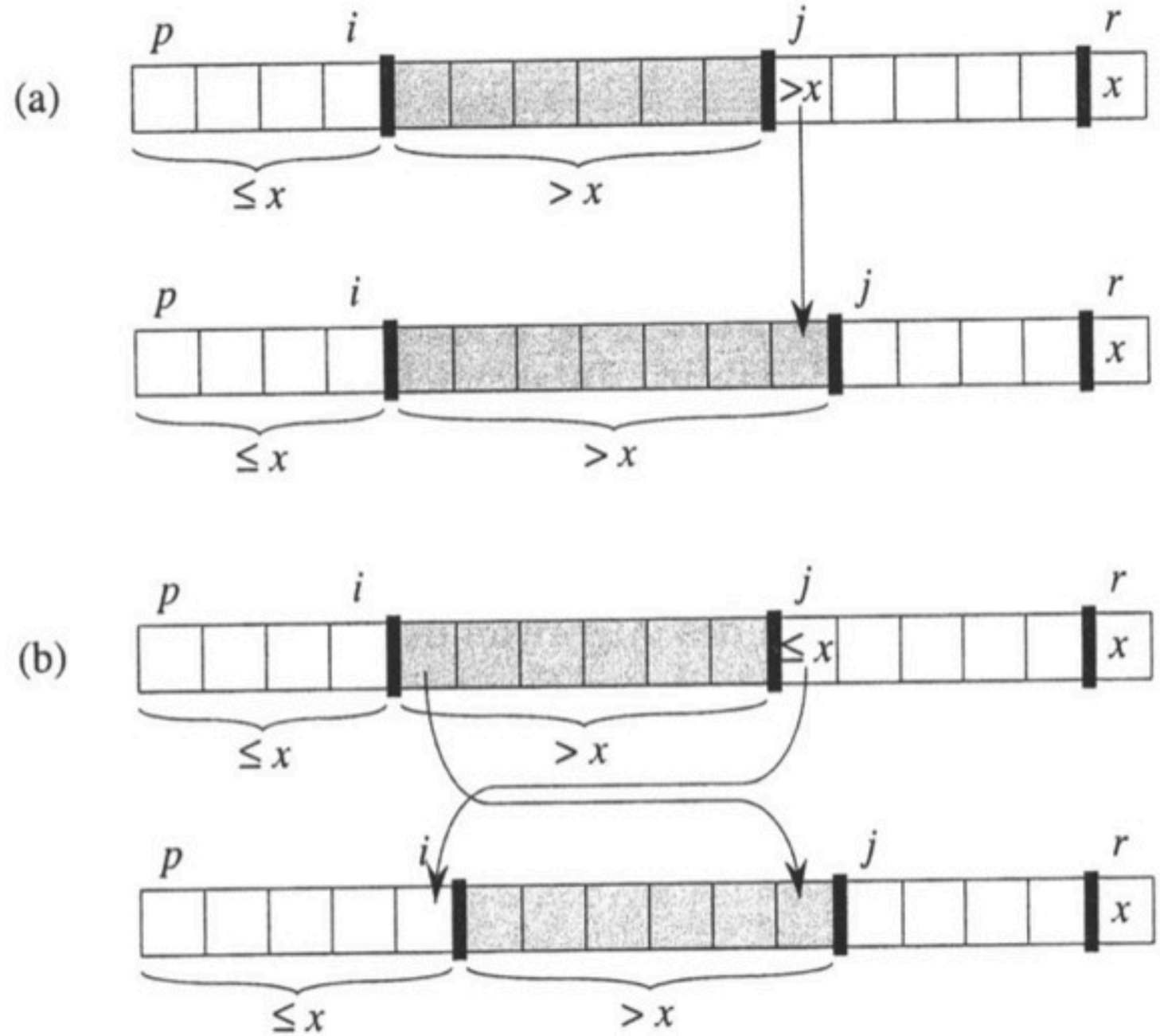
(a)



(b)



5.5.1 Ablauf Quicksort



5.5.2 Algorithmische Beschreibung

Algorithmus 5.1 1

5.5.2 Algorithmische Beschreibung

Algorithmus 5.1 1

INPUT: Subarray von A=[1,...,n],

5.5.2 Algorithmische Beschreibung

Algorithmus 5.1 1

INPUT: Subarray von A=[1,...,n],
der bei Index p beginnt und bei Index r endet, d.h. A[p,...,r]

5.5.2 Algorithmische Beschreibung

Algorithmus 5.1 1

INPUT: Subarray von A=[1,...,n],
der bei Index p beginnt und bei Index r endet, d.h. A[p,...,r]

OUTPUT: Sortierter Subarray

5.5.2 Algorithmische Beschreibung

Algorithmus 5.1 1

INPUT: Subarray von A=[1,...,n],
der bei Index p beginnt und bei Index r endet, d.h. A[p,...,r]

OUTPUT: Sortierter Subarray

5.5.2 Algorithmische Beschreibung

Algorithmus 5.1 1

INPUT: Subarray von A=[1,...,n],
der bei Index p beginnt und bei Index r endet, d.h. A[p,...,r]

OUTPUT: Sortierter Subarray

```
QUICKSORT(A, p, r)
1  if p < r
2      then q ← PARTITION(A, p, r)
3           QUICKSORT(A, p, q − 1)
4           QUICKSORT(A, q + 1, r)
```

5.5.2 Algorithmische Beschreibung

Algorithmus 5.1 1

INPUT: Subarray von A=[1,...,n],
der bei Index p beginnt und bei Index r endet, d.h. A[p,...,r]

OUTPUT: Sortierter Subarray

```
QUICKSORT(A, p, r)
1  if p < r
2      then q ← PARTITION(A, p, r)
3           QUICKSORT(A, p, q − 1)
4           QUICKSORT(A, q + 1, r)
```

Subroutine 5.1 2

Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

Subroutine 5.1 2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]

Subroutine 5.1 2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit $A[i] \leq A[q]$ und $A[q] < A[j]$ für i=p,...,q-1 und j=q+1,...,r

Subroutine 5.1 2

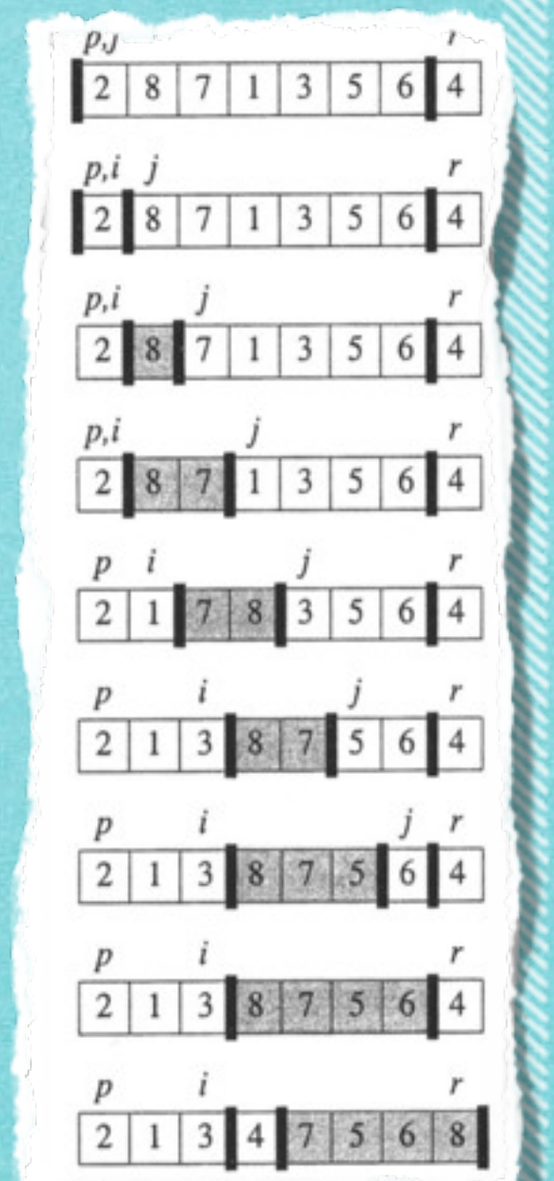
INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit $A[i] \leq A[q]$ und $A[q] < A[j]$ für i=p,...,q-1 und j=q+1,...,r

Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit A[i]≤A[q] und A[q]<A[j] für i=p,...,q-1 und j=q+1,...,r

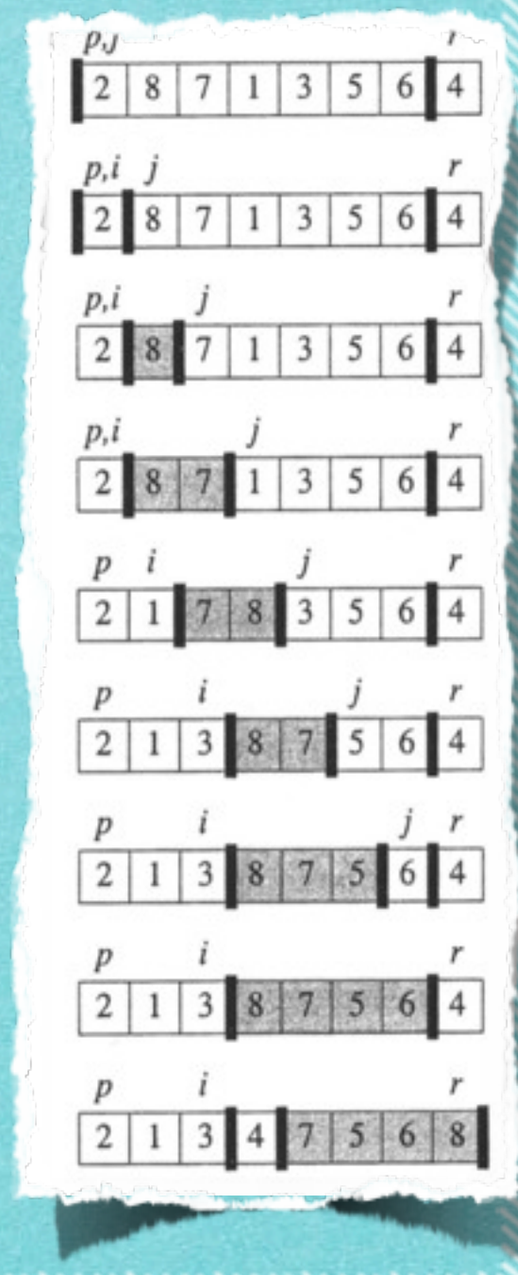


Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit $A[i] \leq A[q]$ und $A[q] < A[j]$ für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
```

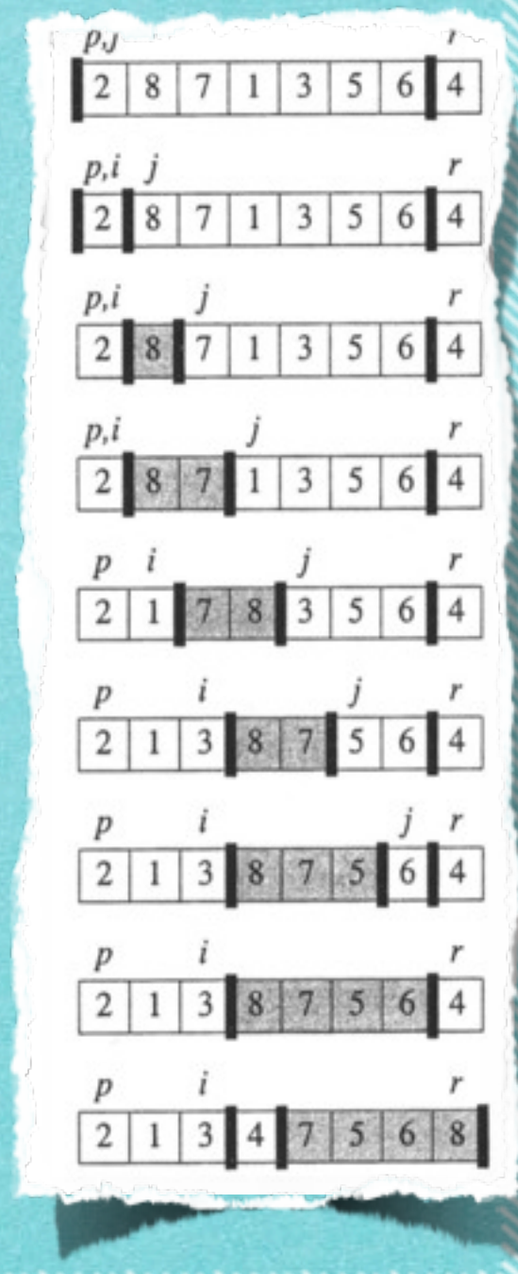


Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit A[i]≤A[q] und A[q]<A[j] für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
1  x ← A[r]
```

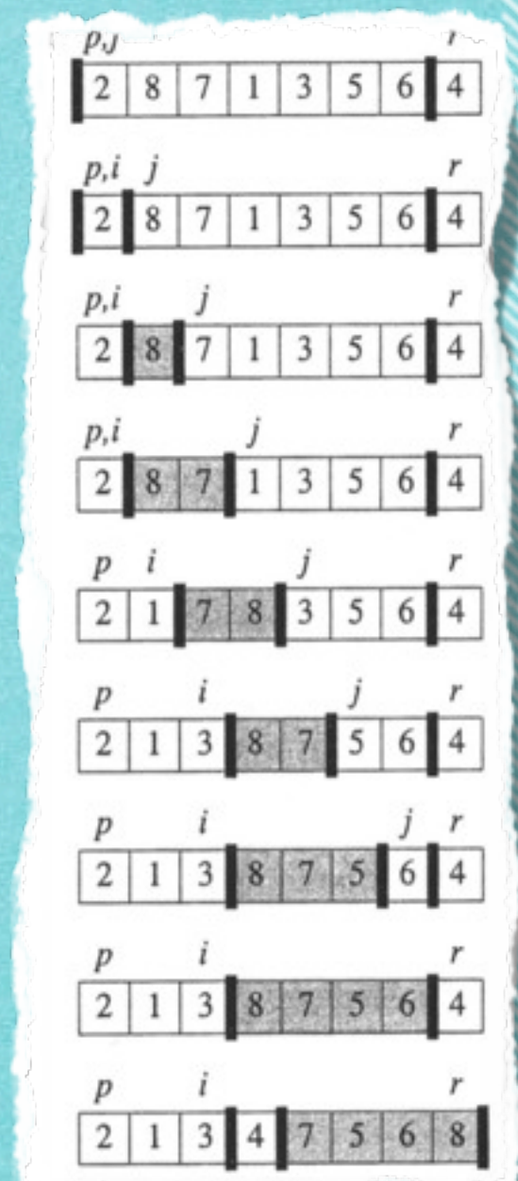


Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit A[i]≤A[q] und A[q]<A[j] für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
```

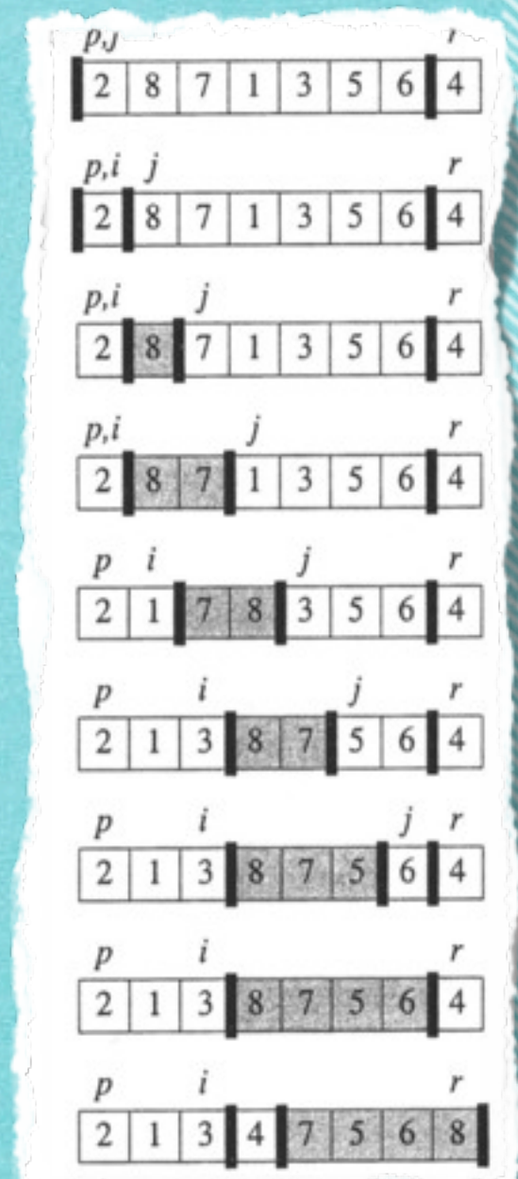


Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit $A[i] \leq A[q]$ und $A[q] < A[j]$ für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
```

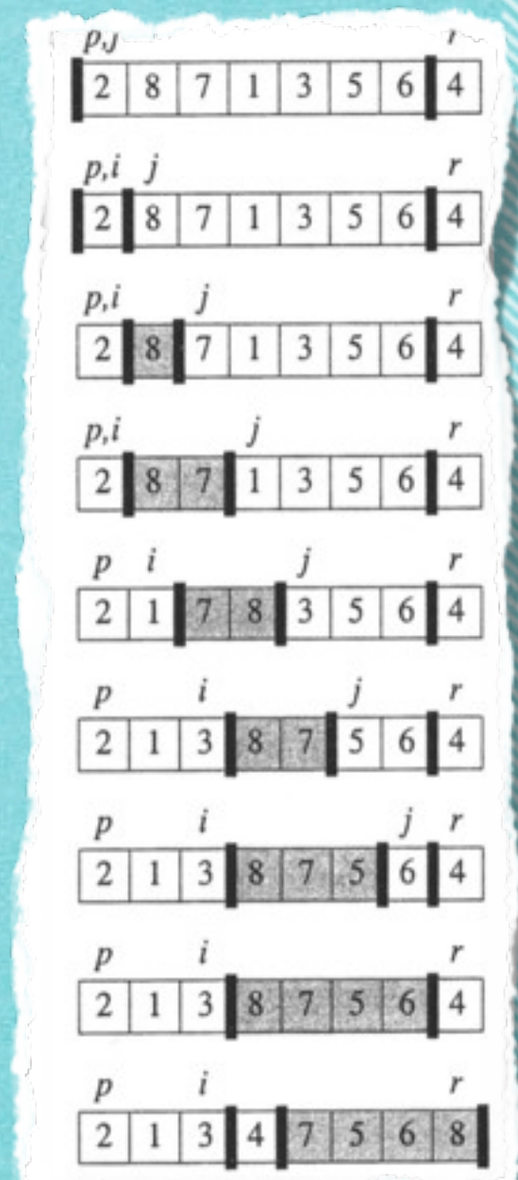


Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit $A[i] \leq A[q]$ und $A[q] < A[j]$ für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4      do if A[j] ≤ x
```

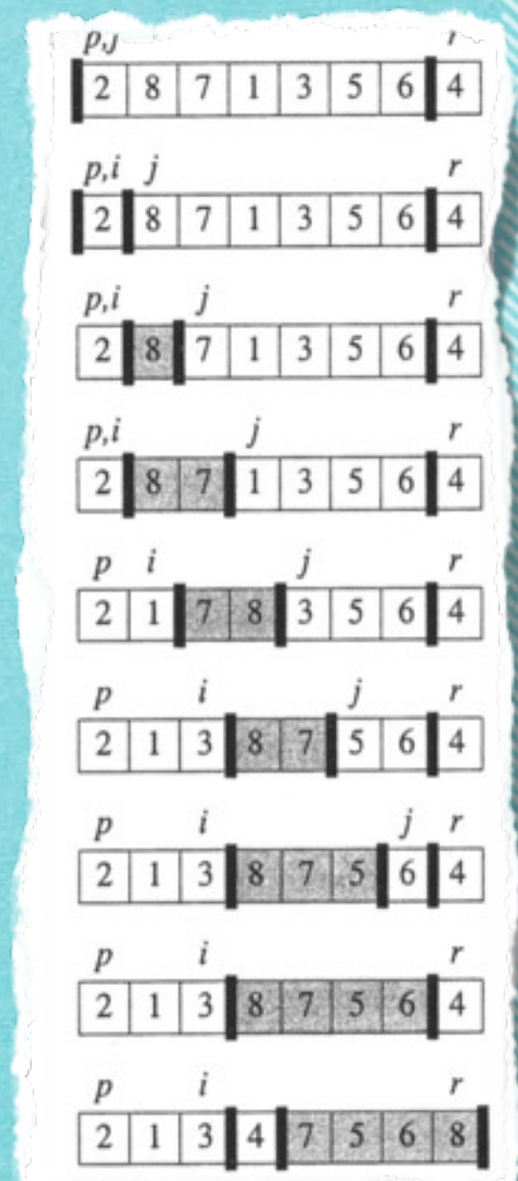


Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit $A[i] \leq A[q]$ und $A[q] < A[j]$ für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4      do if A[j] ≤ x
5          then i ← i + 1
6               vertausche A[i] ↔ A[j]
```

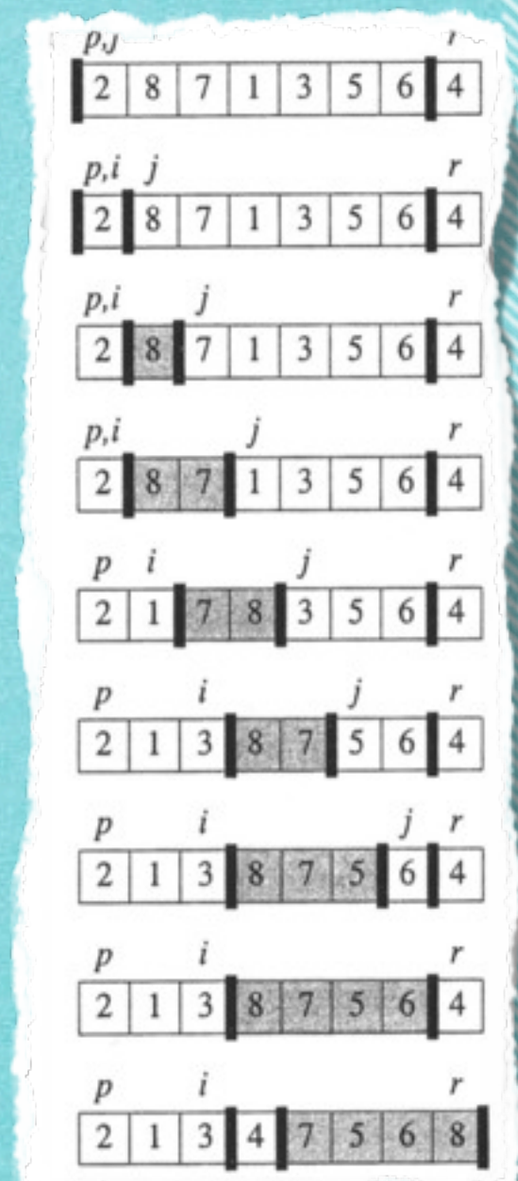


Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit A[i]≤A[q] und A[q]<A[j] für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   vertausche A[i] ↔ A[j]
7  vertausche A[i + 1] ↔ A[r]
```

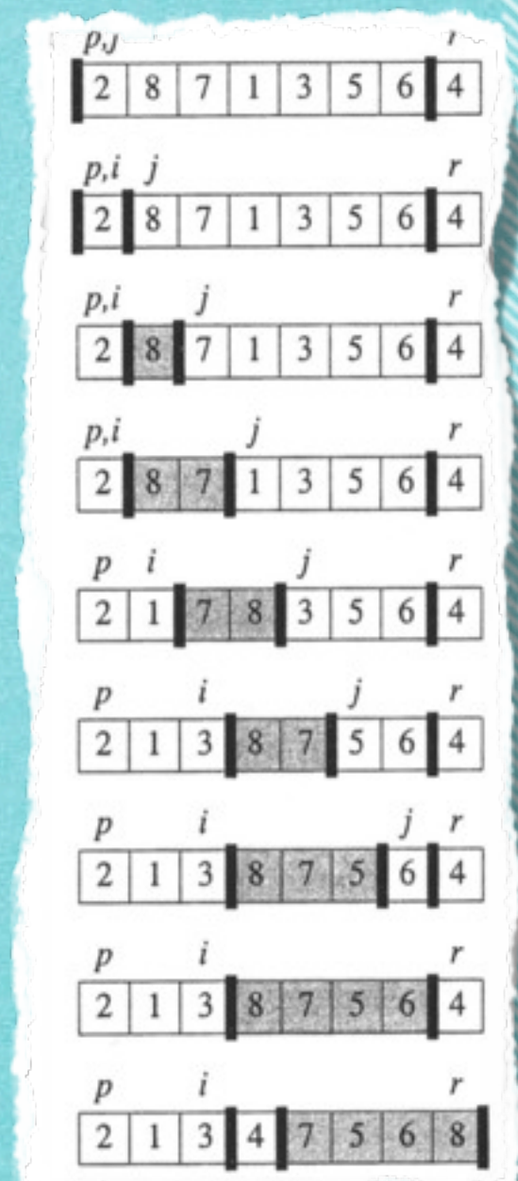


Subroutine 5.1.2

INPUT: Subarray von A=[1,...,n], d.h. A[p,...,r]

OUTPUT: Zwei Subarrays A[p,...,q-1] und A[q+1,...,r]
mit A[i]≤A[q] und A[q]<A[j] für i=p,...,q-1 und j=q+1,...,r

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4        do if A[j] ≤ x
5              then i ← i + 1
6                   vertausche A[i] ↔ A[j]
7  vertausche A[i + 1] ↔ A[r]
8  return i + 1
```



5.5.3 Laufzeit von Quicksort

5.5.3 Laufzeit von Quicksort

Wie viele Schritte benötigt Quicksort für einen Array der Länge n?

5.5.3 Laufzeit von Quicksort

Wie viele Schritte benötigt Quicksort für einen Array der Länge n?

<u>Unterscheidung:</u>

5.5.3 Laufzeit von Quicksort

Wie viele Schritte benötigt Quicksort für einen Array der Länge n?

<u>Unterscheidung:</u>

- **Worst Case**

5.5.3 Laufzeit von Quicksort

Wie viele Schritte benötigt Quicksort für einen Array der Länge n?

<u>Unterscheidung:</u>

- **Worst Case**
- **Best Case**

5.5.3 Laufzeit von Quicksort

Wie viele Schritte benötigt Quicksort für einen Array der Länge n?

<u>Unterscheidung:</u>

- **Worst Case**
- **Best Case**
- **Average Case**

5.5.3 Quicksort: Worst Case

5.5.3 Quicksort: Worst Case

Schlechtester Fall: Pivot liegt extremal, d.h. nach PARTITION ist ein Teilarray n-1 lang, der andere hat Länge 0.

5.5.3 Quicksort: Worst Case

Schlechtester Fall: Pivot liegt extremal, d.h. nach PARTITION ist ein Teilarray n-1 lang, der andere hat Länge 0.

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \ . \end{aligned}$$

5.5.3 Quicksort: Worst Case

Schlechtester Fall: Pivot liegt extremal, d.h. nach PARTITION ist ein Teilarray n-1 lang, der andere hat Länge 0.

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \ . \end{aligned}$$

Man sieht (und beweist leicht):

5.5.3 Quicksort: Worst Case

Schlechtester Fall: Pivot liegt extremal, d.h. nach PARTITION ist ein Teilarray n-1 lang, der andere hat Länge 0.

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \ . \end{aligned}$$

Man sieht (und beweist leicht):

$$\sum_{i=0}^{n} \Theta(n-i) = \Theta(n^2)$$

5.5.3 Quicksort: Best Case

5.5.3 Quicksort: Best Case

Bester Fall: Pivot liegt genau in der Mitte, d.h. nach PARTITION haben beide Teilarrays i.W. die Länge n/2.

5.5.3 Quicksort: Best Case

Bester Fall: Pivot liegt genau in der Mitte, d.h. nach PARTITION haben beide Teilarrays i.W. die Länge n/2.

$$T(n) \leq 2T(n/2) + \Theta(n)$$

5.5.3 Quicksort: Best Case

Bester Fall: Pivot liegt genau in der Mitte, d.h. nach PARTITION haben beide Teilarrays i.W. die Länge n/2.

$$T(n) \leq 2T(n/2) + \Theta(n)$$

Man sieht, z.B. mit dem Mastertheorem:

5.5.3 Quicksort: Best Case

Bester Fall: Pivot liegt genau in der Mitte, d.h. nach PARTITION haben beide Teilarrays i.W. die Länge n/2.

$$T(n) \leq 2T(n/2) + \Theta(n)$$

Man sieht, z.B. mit dem Mastertheorem:

$$\Theta(n \log n)$$

5.5.3 Quicksort: Best Case

5.5.3 Quicksort: Best Case

Das funktioniert auch noch, wenn PARTITION ein *annähernd* balanciertes Ergebnis liefert:

5.5.3 Quicksort: Best Case

Das funktioniert auch noch, wenn PARTITION ein *annähernd* balanciertes Ergebnis liefert:

$$T(n) \leq T(9n/10) + T(n/10) + cn$$

5.5.3 Quicksort: Best Case

Das funktioniert auch noch, wenn PARTITION ein *annähernd* balanciertes Ergebnis liefert:

$$T(n) \leq T(9n/10) + T(n/10) + cn$$

Man sieht wieder mit dem Mastertheorem:

5.5.3 Quicksort: Best Case

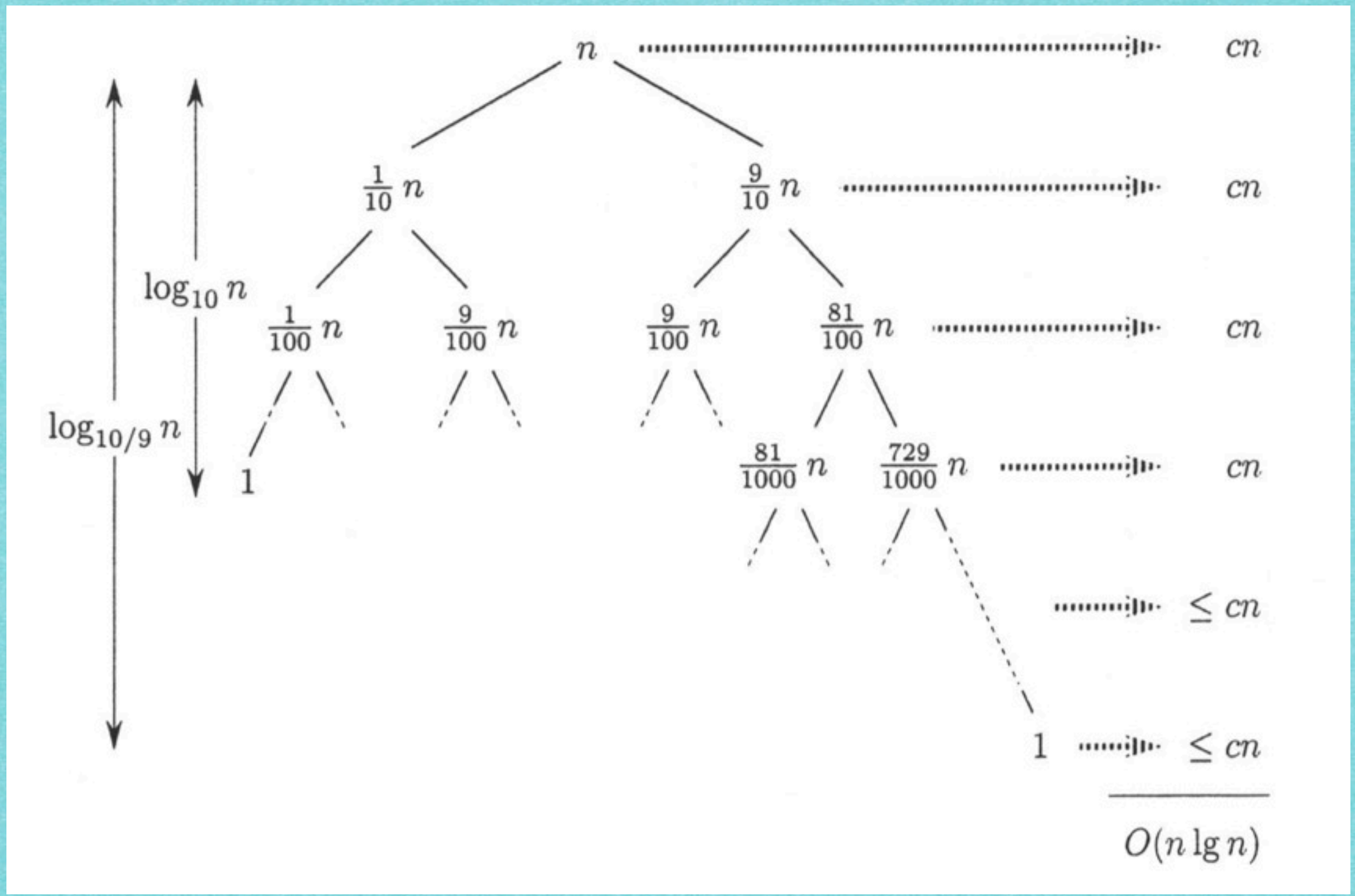
Das funktioniert auch noch, wenn PARTITION ein *annähernd* balanciertes Ergebnis liefert:

$$T(n) \leq T(9n/10) + T(n/10) + cn$$

Man sieht wieder mit dem Mastertheorem:

$$\Theta(n \log n)$$

5.5.3 Quicksort: Best Case



5.5.3 Quicksort: Average Case

5.5.3 Quicksort: Average Case

Satz 5.13 (Durchschnittliche Laufzeit von Quicksort)

5.5.3 Quicksort: Average Case

<u>Satz 5.13 (Durchschnittliche Laufzeit von Quicksort)</u>
Für einen n-elementigen Array A hat Quicksort eine erwartete Laufzeit von O(n log n).

5.5.3 Quicksort: Average Case

5.5.3 Quicksort: Average Case

Warum interessiert das?

5.5.3 Quicksort: Average Case

Warum interessiert das?

5.5.3 Quicksort: Average Case

Warum interessiert das?

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1981**

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1981**

Zlatan Ibrahimovic

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1987**

*** 1981**

Zlatan Ibrahimovic

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1987**

Sebastian Wild

*** 1981**

Zlatan Ibrahimovic

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1987**

Sebastian Wild

*** 1981**

Zlatan Ibrahimovic

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1987**

Sebastian Wild

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1987**

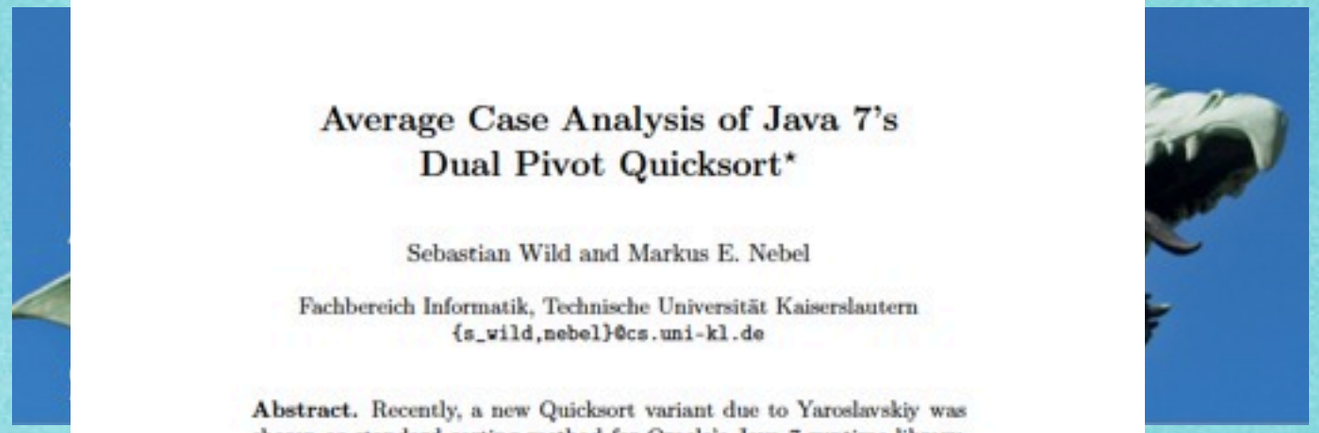
Sebastian Wild

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1987**

Sebastian Wild



Average Case Analysis of Java 7's Dual Pivot Quicksort*

Sebastian Wild and Markus E. Nebel

Fachbereich Informatik, Technische Universität Kaiserslautern
{s_wild,nebel}@cs.uni-kl.de

Abstract. Recently, a new Quicksort variant due to Yaroslavskiy was chosen as standard sorting method for Oracle's Java 7 runtime library. The decision for the change was based on empirical studies showing that on average, the new algorithm is faster than the formerly used classic Quicksort. Surprisingly, the improvement was achieved by using a dual pivot approach, an idea that was considered not promising by several theoretical studies in the past. In this paper, we identify the reason for this unexpected success. Moreover, we present the first precise average case analysis of the new algorithm showing e. g. that a random permutation of length n is sorted using $1.9n \ln n - 2.46n + \mathcal{O}(\ln n)$ key comparisons and $0.6n \ln n + 0.08n + \mathcal{O}(\ln n)$ swaps.

1 Introduction

Due to its efficiency in the average, Quicksort has been used for decades as general purpose sorting method in many domains, e. g. in the C and Java standard libraries or as UNIX's system sort. Since its publication in the early 1960s by Hoare [1], classic Quicksort (Algorithm 1) has been intensively studied and many modifications were suggested to improve it even further, one of them being the following: Instead of partitioning the input file into two subfiles separated by a single pivot, we can create s partitions out of $s-1$ pivots.

Sedgewick considered the case $s = 3$ in his PhD thesis [2]. He proposed and analyzed the implementation given in Algorithm 2. However, this dual pivot Quicksort variant turns out to be clearly inferior to the much simpler classic algorithm. Later, Hennequin studied the comparison costs for any constant s in his PhD thesis [3], but even for arbitrary $s \geq 3$, he found no improvements that would compensate for the much more complicated partitioning step.[1] These negative results may have discouraged further research along these lines.

Recently, however, Yaroslavskiy proposed the new dual pivot Quicksort implementation as given in Algorithm 3 at the Java core library mailing list[2]. He

* This research was supported by DFG grant NE 1379/3-1.

[1] When s depends on n, we basically get the Samplesort algorithm from [4]. [5], [6] or [7] show that Samplesort can beat Quicksort if hardware features are exploited. [2] even shows that Samplesort is asymptotically optimal with respect to comparisons. Yet, due to its inherent intricacies, it has not been used much in practice.

[2] The discussion is archived at `http://permalink.gmane.org/gmane.comp.java.openjdk.core-libs.devel/2628`.

L. Epstein and P. Ferragina (Eds.): ESA 2012, LNCS 7501, pp. 825–836, 2012.
© Springer-Verlag Berlin Heidelberg 2012

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1987**

Sebastian Wild

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1987**

Sebastian Wild

	Comparisons
Classic Quicksort (Algorithm 1)	$2(n+1)\mathcal{H}_{n+1} - \frac{8}{3}(n+1)$ $\approx 2n\ln n - 1.51n + \mathcal{O}(\ln n)$
Sedgewick (Algorithm 2)	$\frac{32}{15}(n+1)\mathcal{H}_{n+1} - \frac{856}{225}(n+1) + \frac{3}{2}$ $\approx 2.13n\ln n - 2.57n + \mathcal{O}(\ln n)$
Yaroslavskiy (Algorithm 3)	$\frac{19}{10}(n+1)\mathcal{H}_{n+1} - \frac{711}{200}(n+1) + \frac{3}{2}$ $\approx 1.9n\ln n - 2.46n + \mathcal{O}(\ln n)$

5.5.3 Quicksort: Average Case

Warum interessiert das?

*** 1987**

Sebastian Wild

	Comparisons
Classic Quicksort (Algorithm 1)	$2(n+1)\mathcal{H}_{n+1} - \frac{8}{3}(n+1)$ $\approx 2n\ln n - 1.51n + \mathcal{O}(\ln n)$
Sedgewick (Algorithm 2)	$\frac{32}{15}(n+1)\mathcal{H}_{n+1} - \frac{856}{225}(n+1) + \frac{3}{2}$ $\approx 2.13n\ln n - 2.57n + \mathcal{O}(\ln n)$
Yaroslavskiy (Algorithm 3)	$\frac{19}{10}(n+1)\mathcal{H}_{n+1} - \frac{711}{200}(n+1) + \frac{3}{2}$ $\approx 1.9n\ln n - 2.46n + \mathcal{O}(\ln n)$

“Best Paper Award”

Mehr an der Tafel!

Mehr an der Tafel!

s.fekete@tu-bs.de