Technische Universität Braunschweig

Algorithmen und Datenstrukturen – Übung #7

Fragestunde

Ramin Kosfeld und Chek-Manh Loi

08.02.2023

Prüfung

Klausur
Algorithmen und Datenstrukturen
13.02.2024

Name: .....

Vorname: .....

Matr.-Nr.: .....

Studiengang: .....

☐ Bachelor ☐ Master ☐ Andere

Klausurcode:
Dieser wird benötigt, um das Ergebnis der Klausur abzurufen.

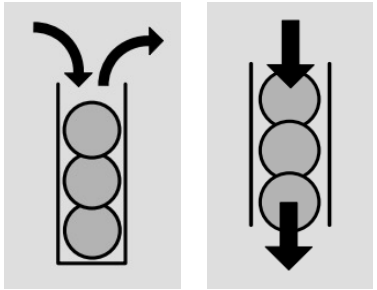
Hinweise:

- Bitte das Deckblatt in Druckschrift vollständig ausfüllen.
- Die Klausur besteht aus 13 Blättern, bitte auf Vollständigkeit überprüfen. Die Heftung darf nicht entfernt werden
- Erlaubte Hilfsmittel: keine
- Eigenes Papier ist nicht erlaubt.
- Die Rückseiten der Blätter dürfen beschrieben werden.
- Die Klausur ist mit 50 % der Punkte bestanden.
- Antworten, die *nicht* gewertet werden sollen, bitte deutlich durchstreichen. Kein Tippex verwenden!
- Mit *Bleistift* oder in *Rot* geschriebene Klausurteile können nicht gewertet werden.
- Werden mehrere Antworten gegeben, werten wir die mit der geringsten Punktzahl.
- Sämtliche Algorithmen, Datenstrukturen, Sätze und Begriffe beziehen sich, sofern nicht explizit anders angegeben, auf die in der Vorlesung vorgestellte Variante.
- Sofern nicht anders angegeben, sind alle Graphen als einfache Graphen zu verstehen.
- Die Bearbeitungszeit für die Klausur beträgt 120 Minuten.

Laufzeiten
dynamische Datenstrukturen

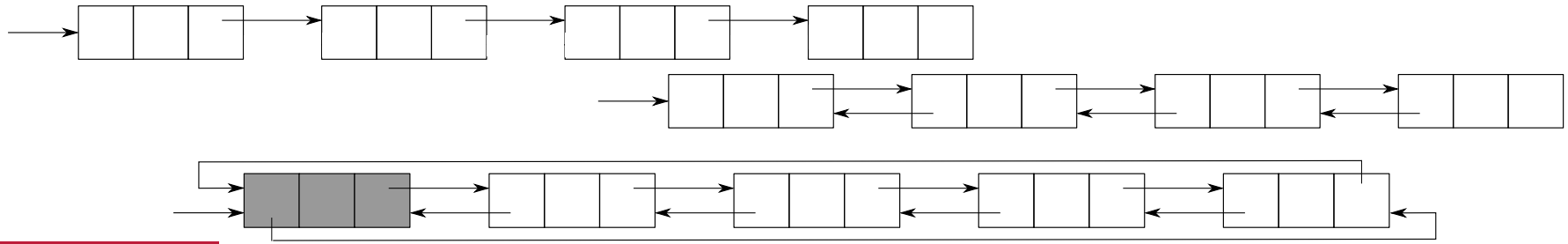
Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

Datentyp	Stack	Queue
Einfügen	$O(1)$	$O(1)$
Nächstes Element	$O(1)$	$O(1)$
Löschen	$O(1)$	$O(1)$

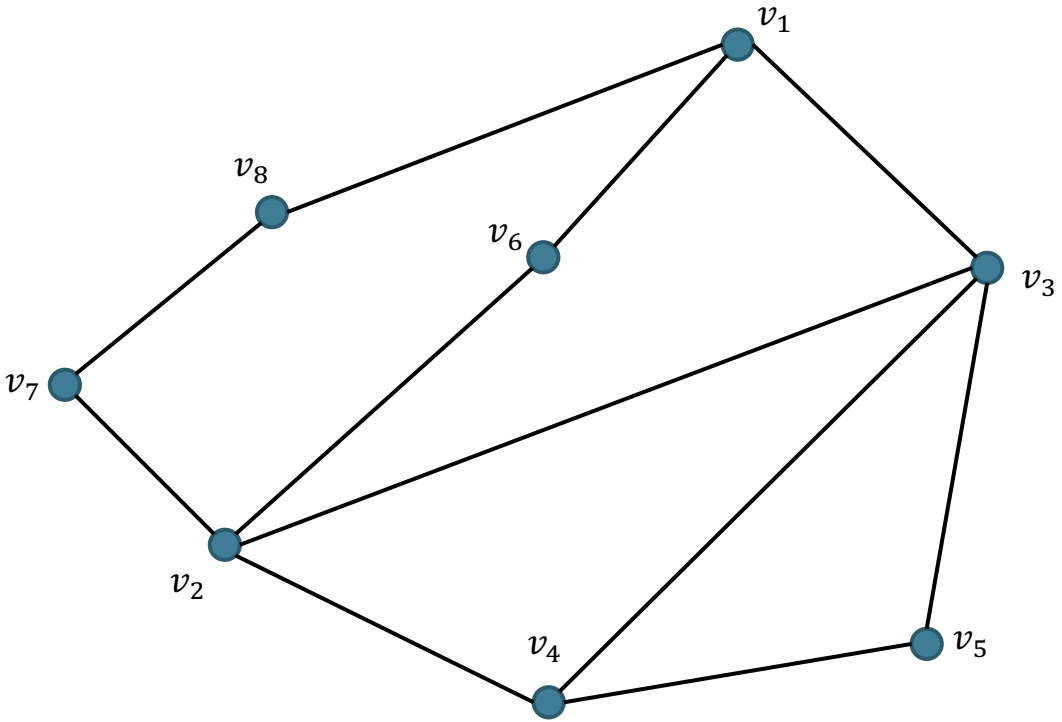


Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

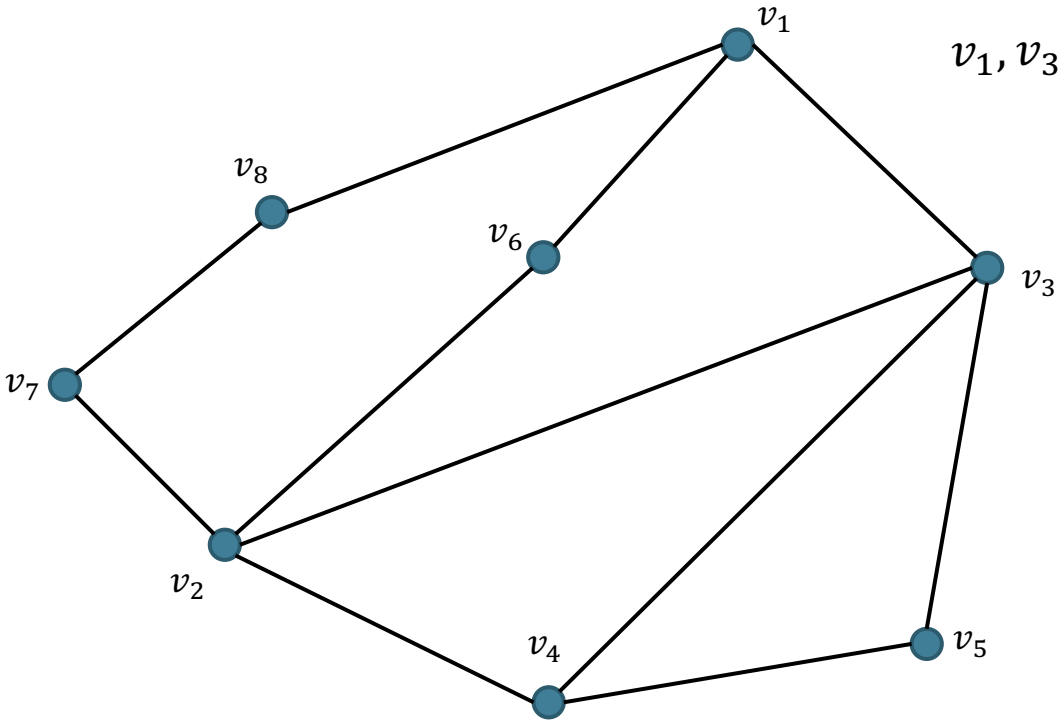
Datentyp	Listen		
Subtyp	Einfach	Doppelt	Zyklisch
Suchen	$O(n)$	$O(n)$	$O(n)$
Einfügen	$O(1)$	$O(1)$	$O(1)$
Löschen	$O(n)$	$O(1)$	$O(1)$
Traversierung	$O(n)$	$O(n)$	$O(n)$



Eulertouren

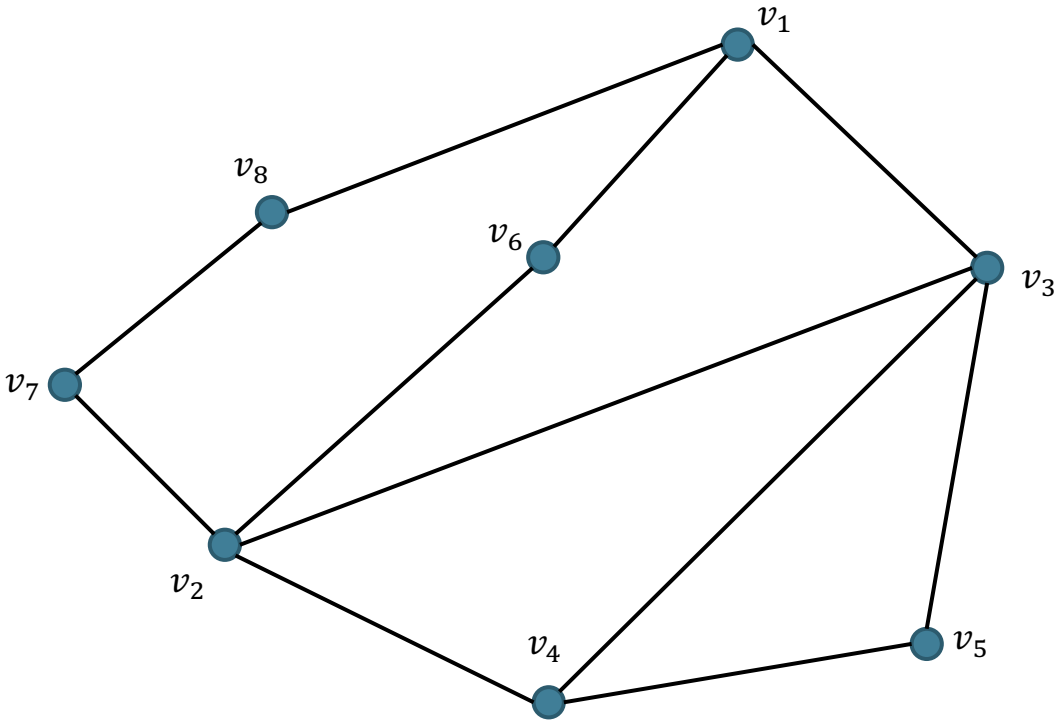


Eulertouren (Fleury)



$v_1, v_3, v_2, v_6, v_1, v_8, v_7, v_2, v_4, v_3, v_5, v_4$

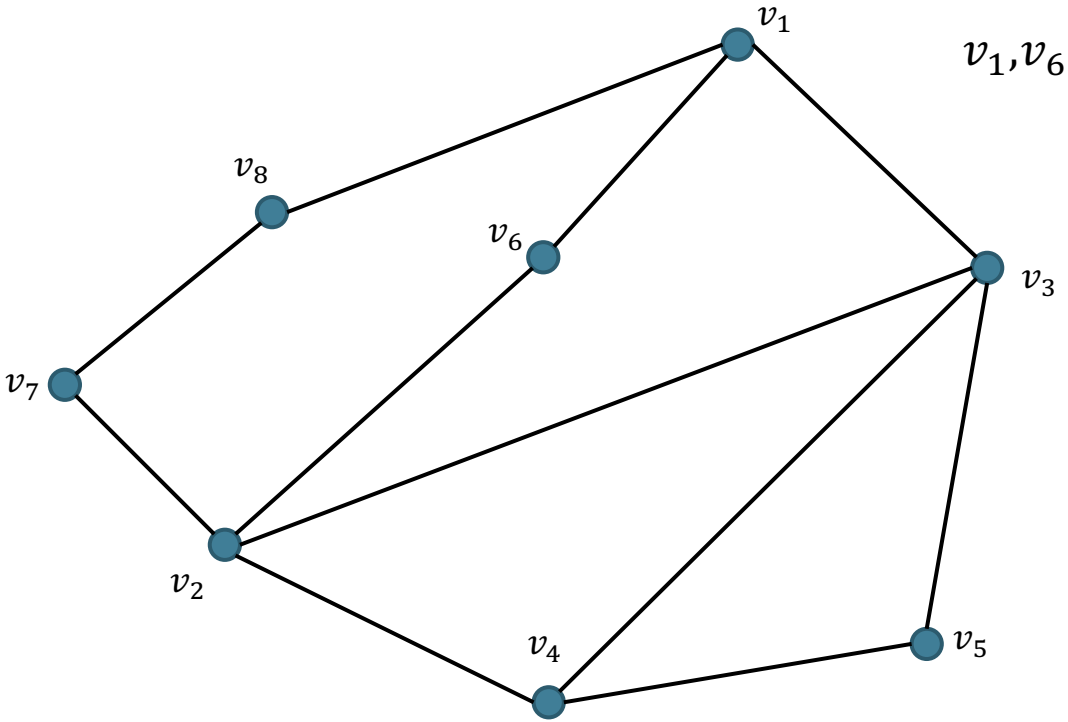
Eulertouren (Hierholzer)



$v_1, v_3, v_2, v_4, v_3, v_5, v_4$

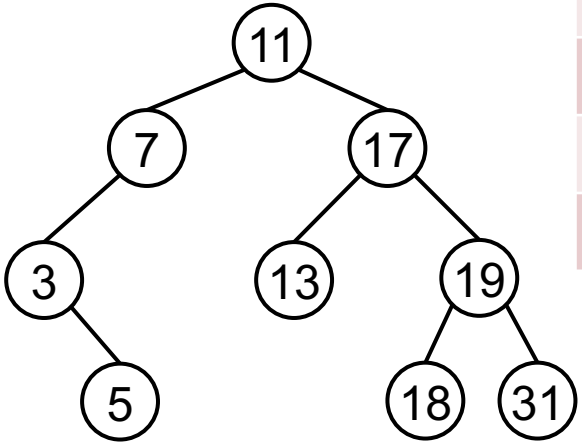
$v_1, v_6, v_2, v_7, v_8, v_1$

Eulertouren (Hierholzer)

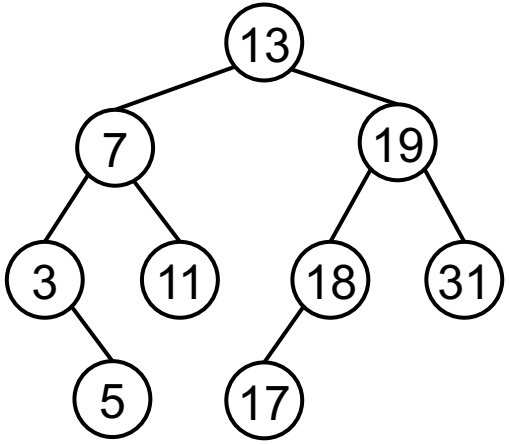


$v_1, v_6, v_2, v_7, v_8, v_1, v_3, v_2, v_4, v_3, v_5, v_4$

Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen	$O(h)$	$O(\log n)$
Einfügen	$O(h)$	$O(\log n)$
Löschen	$O(h)$	$O(\log n)$
Traversierung	$O(n)$	$O(n)$



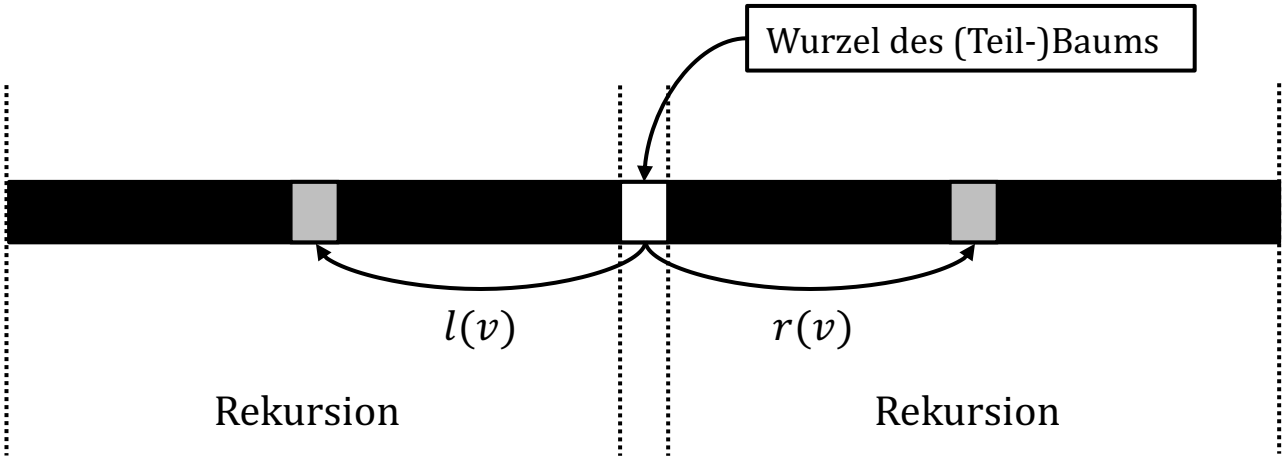
Frage: Wie schnell kann ein bel. bin. Suchbaum in einen AVL-Baum umstrukturiert werden?

Transform

Probieren wir folgende Strategie:

1. Transformiere Suchbaum in sortierts Arrays (durch *inorder* Traversierung).
2. Konstruiere aus dem sortierten Array einen Suchbaum.

Laufzeit $O(n)$:

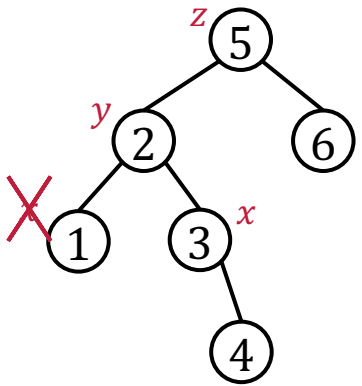


AVL-Bäume – Restructure

Bei Insert und Delete stellen sich nun folgende Fragen:

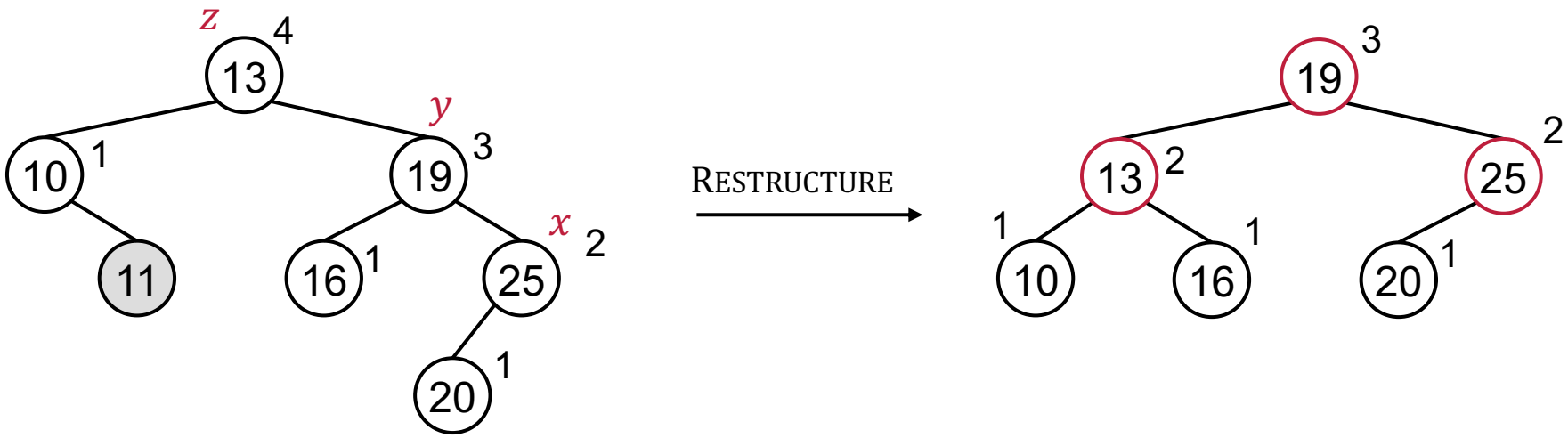
1. Welche Knoten werden unbalanciert?
2. Wie stellt man die Balance wieder her?
3. Welche Regeln sollte man berücksichtigen?

1. Starte bei tiefstem unbalancierten Knoten: Das ist z.
2. Wähle Kinder (x, y) nach deren Höhe aus.



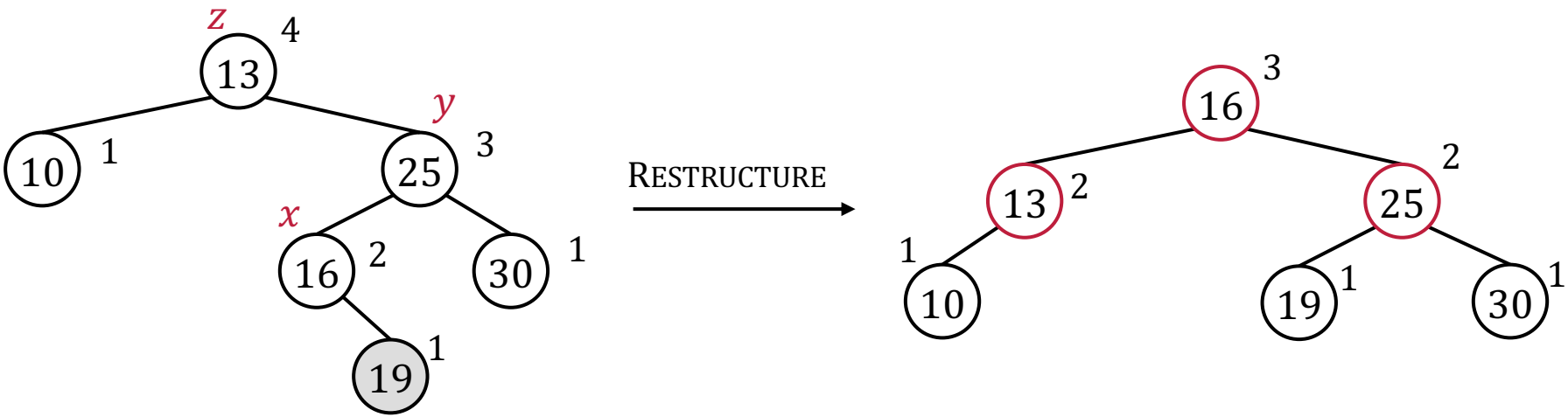
AVL-Bäume – Beispiele

DELETE$(T, 11)$



AVL-Bäume – Beispiele

INSERT$(T, 19)$

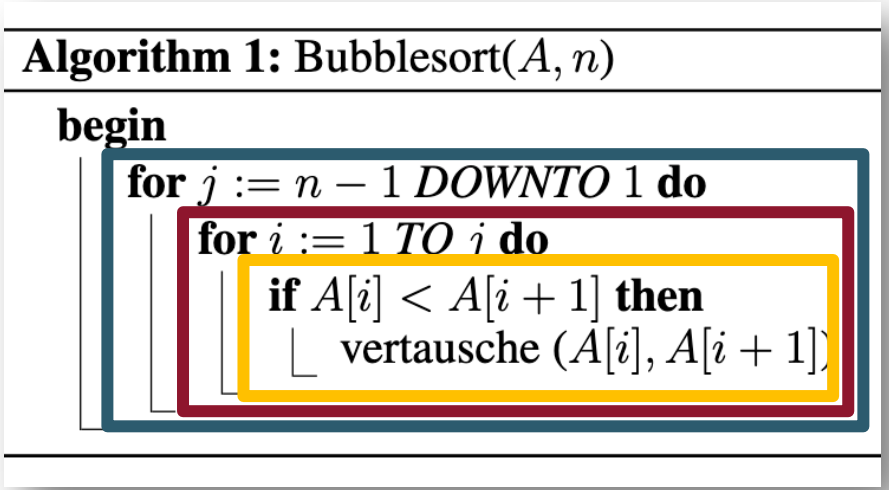


Laufzeiten – dynamische Datenstrukturen (partielle Sortierung)

Datentyp	(Max)Heaps
Einfügen	$O(\log n)$
Löschen	$O(\log n)$
Minimum/Maximum	$O(1)$
Extrahiere Min/Max	$O(\log n)$

Laufzeitanalyse

Sortieren mit Bubblesort



Klausur WiSe 11/12

Laufzeit:

$$\#Iterationen * \#Iterationen * Box$$
$$\#Iterationen * \#Iterationen * O(1)$$
$$\#Iterationen * O(n) * O(1)$$
$$O(n) * O(n) * O(1)$$
$$\Rightarrow O(n^2)$$

Wachstum von Funktionen

Ω-Notation

Definition:

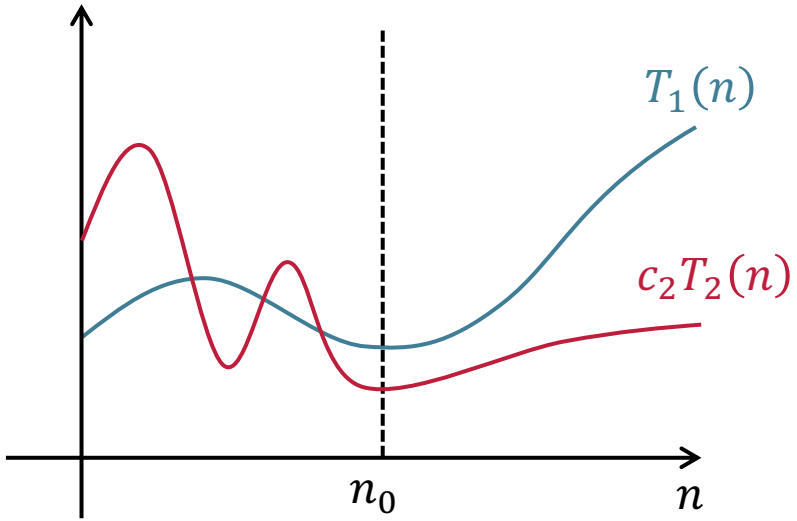
Es gibt Konstanten $n_0 \in \mathbb{N}$ und $c_2 \in \mathbb{R}^+$,

sodass für alle $n \geq n_0$ gilt:

$$T_1(n) \geq c_2 T_2(n) \geq 0$$

$$\Leftrightarrow$$

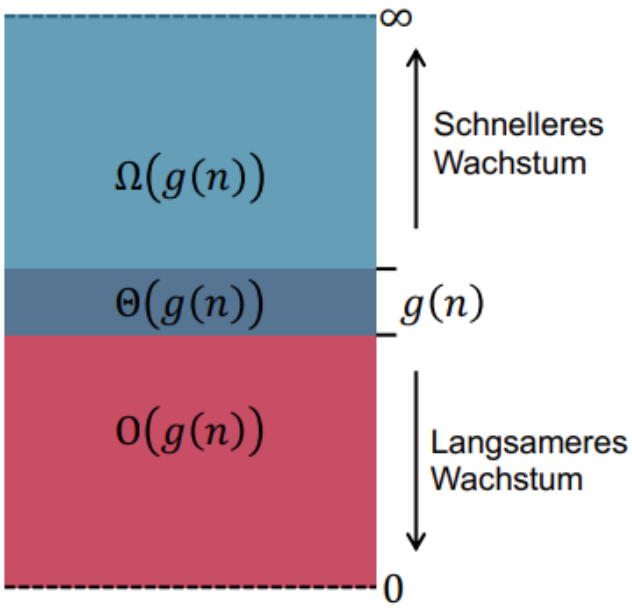
$$T_1(n) \in \Omega\big(T_2(n)\big)$$



„T_1 wächst (asymptotisch) mindestens so schnell wie T_2“

Wachstum von Funktionen

Vergleichen von Klassen



Hierarchie-Ausschnitt:

$$O(1) \subset O(\log^a n) \subset O(n^b) \subset O(c^n) \subset O(n!) \subset O(n^n)$$

$$a, b > 0, c > 1$$

Bei Ω dreht sich das Inklusionszeichen um!

Wo passt dort nun $O(n \log n)$ rein?

Wie steht das zu $O(n \log \log n)$?

Wir wissen:

$$O(\log \log n) \subset O(\log n)$$

Also muss gelten:

$$O(n \log \log n) \subset O(n \log n)$$

Wachstum von Funktionen (Bestimmen der Klasse)

Wachstum von Funktionen

Bestimmen von Klassen

Laufzeiten Merkzettel

1 Definitionen

O-Notation Gibt eine obere Schranke für Funktionen. Gilt $f(n) \in O(g(n))$, so wächst $f(n)$ (asymptotisch) nicht schneller als $g(n)$ denn:

Es existieren zwei Konstanten $c \in \mathbb{R}^+$ und $n_0 \in \mathbb{N}$, sodass für alle $n \geq n_0$ die Ungleichung $0 \leq f(n) \leq c \cdot g(n)$ gilt.

Ω-Notation Gibt eine untere Schranke für Funktionen. Gilt $f(n) \in \Omega(g(n))$, so wächst $f(n)$ (asymptotisch) nicht langsamer als $g(n)$ denn:

Es existieren zwei Konstanten $c \in \mathbb{R}^+$ und $n_0 \in \mathbb{N}$, sodass für alle $n \geq n_0$ die Ungleichung

2 O-Notation

Tipps zum Abschätzen von Funktionen bei der O-Notation:

1. Bei Polynomen können Subtrahenden einfach ignoriert werden. Das Weglassen macht die Funktion nur größer.
2. Bei Polynomen können alle Exponenten von (positiven) Summanden auf den Grad des Polynoms hochgestuft werden. Das macht die Funktion größer.
3. Bei Funktionen die kein Polynom sind, können andere Methoden zum Abschätzen vorteilhaft sein, z.B. das Benutzen von monoton-wachsenden Funktionen (Logarithmieren, Potenzieren[1], Wurzelziehen, etc.).

3 Ω-Notation

Tipps zum Abschätzen von Funktionen bei der Ω-Notation:

1. Bei Polynomen können (positive) Summanden einfach ignoriert werden. Das Weglassen macht die Funktion nur kleiner.
2. Bei Polynomen können alle Exponenten von Subtrahenden **nicht** auf den Grad des Polynoms hochgestuft werden. Das würde die Funktion zwar kleiner machen, aber unter Umständen wird dadurch die Funktion negativ.
3. Bei Funktionen die kein Polynom sind, können andere Methoden zum Abschätzen vorteilhaft sein, z.B. das Benutzen von monoton-wachsenden Funktionen (Logarithmieren, Potenzieren, Wurzelziehen, etc.).

Wachstum von Funktionen

Bestimmen von Klassen

$$\frac{3n^2 - 5n\log n + 23n - 40}{3n\log n - 6n} \cdot \log n \in \Omega(n)$$

Also $n_0 \geq 32$ und $c_1 = \frac{2}{3}$.

Beweis (Ω-Notation)

$$\frac{3n^2 - 5n\log n + 23n - 40}{3n\log n - 6n} \cdot \log n \geq \frac{3n^2 - 5n\log n + 23n - 40}{3n\log n} \cdot \log n$$

$$\geq \frac{1}{3n}(3n^2 - 5n\log n + 23n - 40)$$

Ab $n_0 \geq 2$, da $40 \leq 20n$ gelten muss! →

$$\geq \frac{1}{3n}(3n^2 - 5n\log n + 23n - 20n) \geq \frac{1}{3n}(3n^2 - 5n\log n)$$

Ab $n_0 \geq 32$, da $5\log n \leq n$ gelten muss! →

$$\geq \frac{1}{3n}(3n^2 - n^2) \geq \frac{2}{3}n$$

Wachstum von Funktionen (Beweise)

Satz 3.12

Seien $f, g: \mathbb{N} \to \mathbb{R}$, dann gilt $f(n) \in O\big(g(n)\big) \Leftrightarrow g(n) \in \Omega\big(f(n)\big)$

$f(n) \in O\big(g(n)\big)$

$\Leftrightarrow \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}: \forall n \geq n_0: \quad 0 \leq f(n) \leq c \cdot g(n)$

$\Leftrightarrow \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}: \forall n \geq n_0: \quad 0 \leq \frac{1}{c} \cdot f(n) \leq g(n)$

$\Leftrightarrow \exists c' \in \mathbb{R}^+, n_0 \in \mathbb{N}: \forall n \geq n_0: \quad 0 \leq c' \cdot f(n) \leq g(n)$ (nämlich $c' = \frac{1}{c}$)

$\Leftrightarrow g(n) \in \Omega\big(f(n)\big)$

Mediane

Mediane – Definition

Rang-k Element m in X:

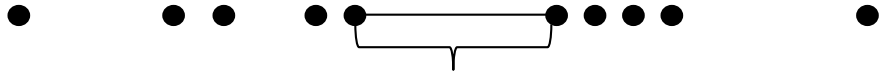
$$|\{x \in X : x \leq m\}| \geq k$$

$$|\{x \in X : x \geq m\}| \geq n - k + 1$$

Für einen Median m in X gilt:

$$|\{x \in X : x < m\}| \leq \left\lfloor \frac{n}{2} \right\rfloor$$

$$|\{x \in X : x > m\}| \leq \left\lfloor \frac{n}{2} \right\rfloor$$



Jeder Punkt in diesem Bereich ist ein Median!

Bei $X = \{1,2,3,4,5,6,7,8\}$ sind sowohl 4 als auch 5 ein Median.

Beispiel

$X := \{14, 23, 15, 25, 17, 19, 20, 21, 22, 24, 16, 18\}$

$k = 2$

Fünfergruppen:

14	19	16
23	20	18
15	21	
25	22	
17	24	

```
1:  function FindRankElement(X, k)
2:      if |X| ≤ 5 then Sortiere X und gib das Rang-k Element zurück.
3:      Teile X in t := ⌈|X|/5⌉ Fünfergruppen X_1, ..., X_t auf.
4:      Bestimme für jedes X_i den Median m_i.
5:      m := FindRankElement({m_1, ..., m_t}, ⌈t/2⌉)      ▷ Finde Median der Mediane.
6:      X_< = {x ∈ X | x < m}
7:      X_= = {x ∈ X | x = m}
8:      X_> = {x ∈ X | x > m}
9:      if k ≤ |X_<| then
10:         return FindRankElement(X_<, k)
11:     else if k ≤ |X_= ∪ X_<| then
12:         return m
13:     else
14:         return FindRankElement(X_>, k − |X_< ∪ X_=|)
```

Algorithmus 1: Algorithmus zum Finden eines Rang-k Elements

Beispiel

$X := \{14, 23, 15, 25, 17, 19, 20, 21, 22, 24, 16, 18\}$

Sortierte Fünfergruppen:

14	19	16
15	20	18
17	21	
23	22	
25	24	

Mediane: $\{17, 21, 16\}$

```
1:  function FindRankElement(X, k)
2:      if |X| ≤ 5 then Sortiere X und gib das Rang-k Element zurück.
3:      Teile X in t := ⌈|X|/5⌉ Fünfergruppen X_1, ..., X_t auf.
4:      Bestimme für jedes X_i den Median m_i.
5:      m := FindRankElement({m_1, ..., m_t}, ⌊t/2⌋)      ▷ Finde Median der Mediane.
6:      X_< = {x ∈ X | x < m}
7:      X_= = {x ∈ X | x = m}
8:      X_> = {x ∈ X | x > m}
9:      if k ≤ |X_<| then
10:         return FindRankElement(X_<, k)
11:     else if k ≤ |X_= ∪ X_<| then
12:         return m
13:     else
14:         return FindRankElement(X_>, k − |X_< ∪ X_=|)
```

Algorithmus 1: Algorithmus zum Finden eines Rang-k Elements

Beispiel

$$X := \{14, 23, 15, 25, 17, 19, 20, 21, 22, 24, 16, 18\}$$

Mediane: $\{17, 21, 16\}$

Median der Mediane: 17

1: **function** FINDRANKELEMENT(X, k)
2: **if** $|X| \leq 5$ **then** Sortiere X und gib das Rang-k Element zurück.
3: Teile X in $t := \left\lceil \frac{|X|}{5} \right\rceil$ Fünfergruppen $X_1, \ldots, X_t$ auf.
4: Bestimme für jedes X_i den Median m_i.
5: $m :=$ FINDRANKELEMENT($\{m_1, \ldots, m_t\}, \lceil \frac{t}{2} \rceil$) ▷ Finde Median der Mediane.
6: $X_< = \{x \in X \mid x < m\}$
7: $X_= = \{x \in X \mid x = m\}$
8: $X_> = \{x \in X \mid x > m\}$
9: **if** $k \leq |X_<|$ **then**
10: **return** FINDRANKELEMENT($X_<, k$)
11: **else if** $k \leq |X_= \cup X_<|$ **then**
12: **return** m
13: **else**
14: **return** FINDRANKELEMENT($X_>, k - |X_< \cup X_=|$)

Algorithmus 1: Algorithmus zum Finden eines Rang-k Elements

Beispiel

$$X := \{14, 23, 15, 25, 17, 19, 20, 21, 22, 24, 16, 18\}$$

Median der Mediane: 17

$X_< = \{14,15,16\}$

$X_= = \{17\}$

$X_> = \{23, 25, 19, 20, 21, 22, 24, 18\}$

```
1: function FindRankElement(X, k)
2:     if |X| ≤ 5 then Sortiere X und gib das Rang-k Element zurück.
3:     Teile X in t := ⌈|X|/5⌉ Fünfergruppen X_1, ..., X_t auf.
4:     Bestimme für jedes X_i den Median m_i.
5:     m := FindRankElement({m_1, ..., m_t}, ⌈t/2⌉)      ▷ Finde Median der Mediane.
6:     X_< = {x ∈ X | x < m}
7:     X_= = {x ∈ X | x = m}
8:     X_> = {x ∈ X | x > m}
9:     if k ≤ |X_<| then
10:        return FindRankElement(X_<, k)
11:    else if k ≤ |X_= ∪ X_<| then
12:        return m
13:    else
14:        return FindRankElement(X_>, k − |X_< ∪ X_=|)
```

Algorithmus 1: Algorithmus zum Finden eines Rang-k Elements

Beispiel

$$X := \{14, 23, 15, 25, 17, 19, 20, 21, 22, 24, 16, 18\}$$

Median der Mediane: 17

$X_< = \{14,15,16\}$

$X_= = \{17\}$

$X_> = \{23, 25, 19, 20, 21, 22,24, 18\}$

$|X_<| = 3$
$|X_=| = 1$
$|X_>| = 8$

$k = 2$

```
1:  function FindRankElement(X, k)
2:      if |X| ≤ 5 then Sortiere X und gib das Rang-k Element zurück.
3:      Teile X in t := ⌈|X|/5⌉ Fünfergruppen X_1, ..., X_t auf.
4:      Bestimme für jedes X_i den Median m_i.
5:      m := FindRankElement({m_1, ..., m_t}, ⌈t/2⌉)      ▷ Finde Median der Mediane.
6:      X_< = {x ∈ X | x < m}
7:      X_= = {x ∈ X | x = m}
8:      X_> = {x ∈ X | x > m}
9:      if k ≤ |X_<| then
10:         return FindRankElement(X_<, k)
11:     else if k ≤ |X_= ∪ X_<| then
12:         return m
13:     else
14:         return FindRankElement(X_>, k − |X_< ∪ X_=|)
```

Algorithmus 1: Algorithmus zum Finden eines Rang-k Elements

Beispiel

$$X := \{14, 23, 15, 25, 17, 19, 20, 21, 22, 24, 16, 18\}$$

Median der Mediane: 17

$X_< = \{14,15,16\}$

$X_= = \{17\}$

$X_> = \{23, 25, 19, 20, 21, 22,24, 18\}$

$|X_<| = 3$
$|X_=| = 1$
$|X_>| = 8$

$k = 2$

```
1:  function FindRankElement(X, k)
2:      if |X| ≤ 5 then Sortiere X und gib das Rang-k Element zurück.
3:      Teile X in t := ⌈|X|/5⌉ Fünfergruppen X_1, ..., X_t auf.
4:      Bestimme für jedes X_i den Median m_i.
5:      m := FindRankElement({m_1, ..., m_t}, ⌈t/2⌉)      ▷ Finde Median der Mediane.
6:      X_< = {x ∈ X | x < m}
7:      X_= = {x ∈ X | x = m}
8:      X_> = {x ∈ X | x > m}
9:      if k < |X_<| then
10:         return FindRankElement(X_<, k)
11:     else if k ≤ |X_= ∪ X_<| then
12:         return m
13:     else
14:         return FindRankElement(X_>, k − |X_< ∪ X_=|)
```

Algorithmus 1: Algorithmus zum Finden eines Rang-k Elements

Suche in $|X_<|$ nach dem Rang-2 Element → 15 ist das gesuchte Element.

Quicksort

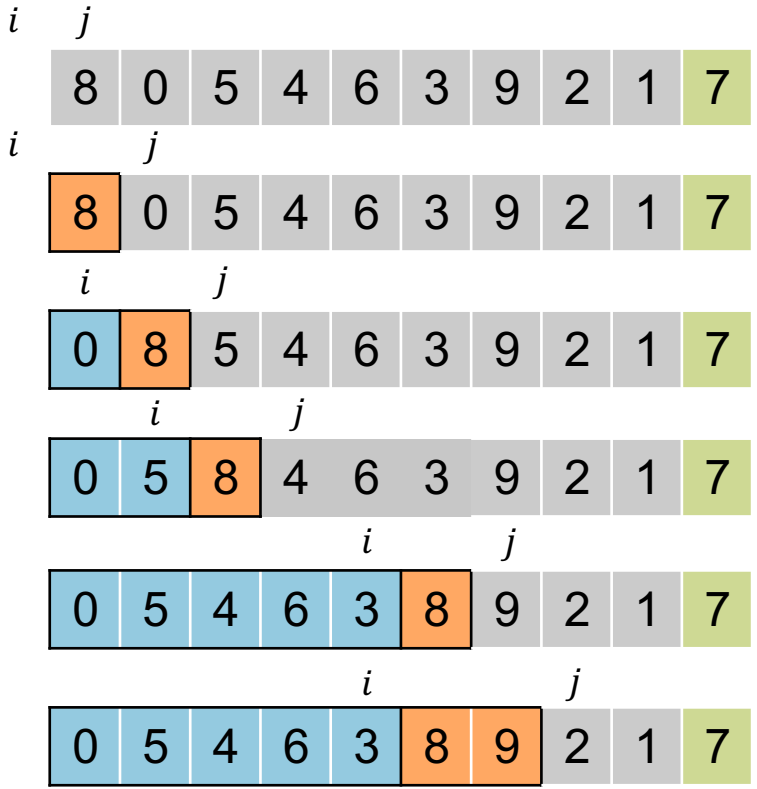
Quicksort – Partition

Pivotelement x

Letztes Element im Array

Zwei Zeiger

- i: Letzte Position mit Zahlen $\leq x$
- j: Erste Position mit nicht verglichenen Elementen



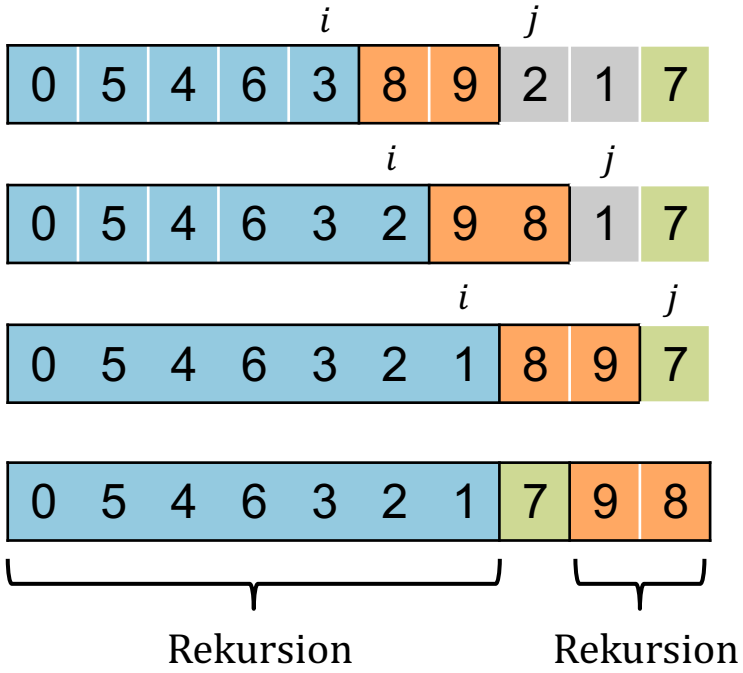
Quicksort – Partition

Pivotelement x

Letztes Element im Array

Zwei Zeiger

- i: Letzte Position mit Zahlen $< x$
- j: Erste Position mit nicht verglichenen Elementen



Induktionsbeweise

Handshake-Lemma

Satz. Sei $G = (V, E)$ ein Graph. Dann besitzt G eine gerade Anzahl an ungeraden Knoten.

Beweis per Induktion über die Anzahl an Kanten m.

IA: Graph ohne Kanten ($m = 0$) besitzt nur gerade Knoten (alle Grad 0).

IV: Annahme gelte für beliebiges, aber festes $m \in \mathbb{N}$.

IS:

- Betrachte Graph mit $m + 1$ Kanten.
- Entferne eine Kante $e = \{u, v\}$.
- Nach IV besitzt $G \setminus e$ gerade Anzahl ungerader Knoten.
- Nach Einfügen von e gilt Eigenschaft wieder (siehe Tabelle).

Ohne e		Mit e		Änderung
u	v	u	v	-
0	0	1	1	+2
1	0	0	1	+0
1	1	0	0	-2

Grad modulo 2

Fragen?

Bei Fragen per Mails:
Erstmal an eure Tutor*innen.

Dabei beachten:
Fragen wie „Ich habe Thema X nicht verstanden. Kannst du das noch mal erklären?“ helfen weder euch, noch uns!

Im Sommer…

…gibt es den zweiten Teil zu AuD (im Wahlpflichtbereich)

Algorithmen und Datenstrukturen 2

Prof. Sándor Fekete

- Einführung in Komplexitätstheorie
- Heuristiken
- Exakte Methoden
- Approximationen
- Hashing

Fragerunde

Viel Erfolg
und
frohes Schaffen!