Technische Universität Braunschweig

Algorithmen und Datenstrukturen – Übung #7

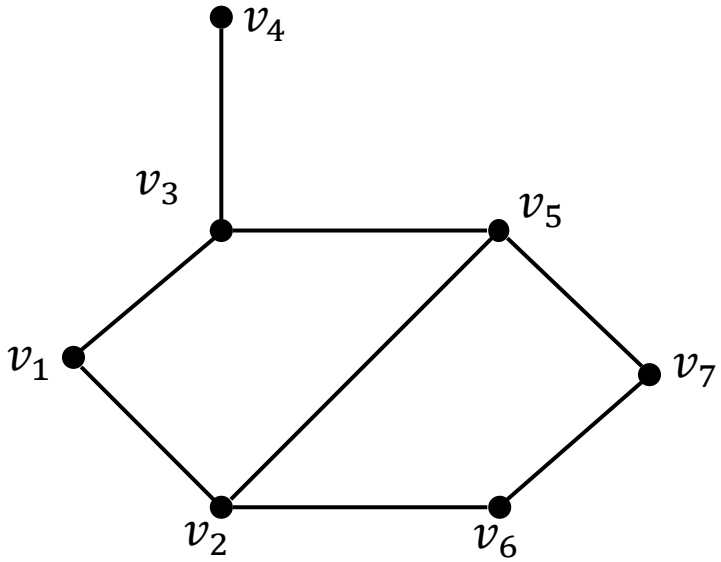
Wiederholung

Ramin Kosfeld und Chek-Manh Loi

05.02.2026

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

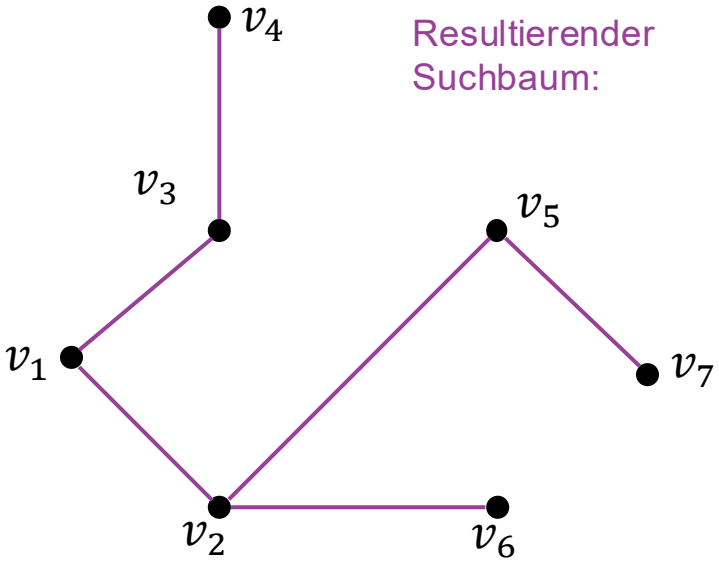


← out ← in

v_1
v_1, v_2
v_1, v_2, v_3
v_2, v_3
v_2, v_3, v_5
v_2, v_3, v_5, v_6
v_3, v_5, v_6
v_3, v_5, v_6, v_4
v_5, v_6, v_4
v_5, v_6, v_4, v_7
v_6, v_4, v_7
v_4, v_7
v_7
$\emptyset$

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

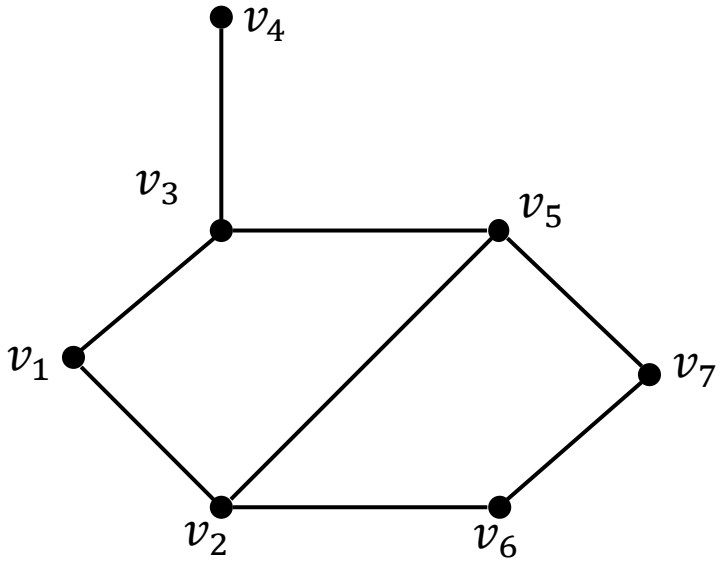


Resultierender Suchbaum:

← out	← in
v_1	
v_1, v_2	
v_1, v_2, v_3	
v_2, v_3	
v_2, v_3, v_5	
v_2, v_3, v_5, v_6	
v_3, v_5, v_6	
v_3, v_5, v_6, v_4	
v_5, v_6, v_4	
v_5, v_6, v_4, v_7	
v_6, v_4, v_7	
v_4, v_7	
v_7	
$\emptyset$	

Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out



v_1 ← in
→ out
v_1, v_2
v_1, v_2, v_5
v_1, v_2, v_5, v_3
v_1, v_2, v_5, v_3, v_4
v_1, v_2, v_5, v_3
v_1, v_2, v_5
v_1, v_2, v_5, v_7
v_1, v_2, v_5, v_7, v_6
v_1, v_2, v_5, v_7
v_1, v_2, v_5
v_1, v_2
v_1
$\emptyset$

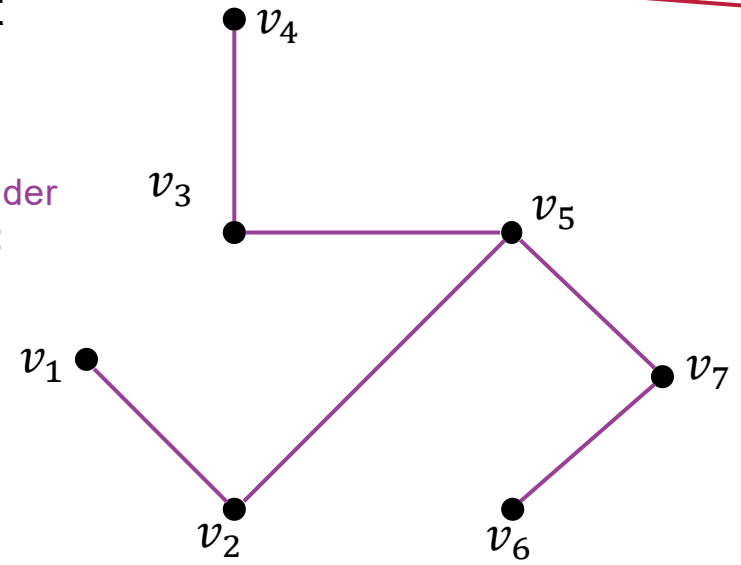
Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out

Nicht vergessen:

- Suchbaum angegeben!
- Immer den kleinsten Index benutzen!

Resultierender Suchbaum:

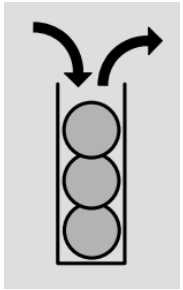


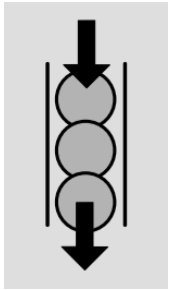
v_1, v_2, v_5
v_1, v_2, v_5, v_3
v_1, v_2, v_5, v_3, v_4
v_1, v_2, v_5, v_3
v_1, v_2, v_5
v_1, v_2, v_5, v_7
v_1, v_2, v_5, v_7, v_6
v_1, v_2, v_5, v_7
v_1, v_2, v_5
v_1, v_2
v_1
$\emptyset$

Laufzeiten
dynamische Datenstrukturen

Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

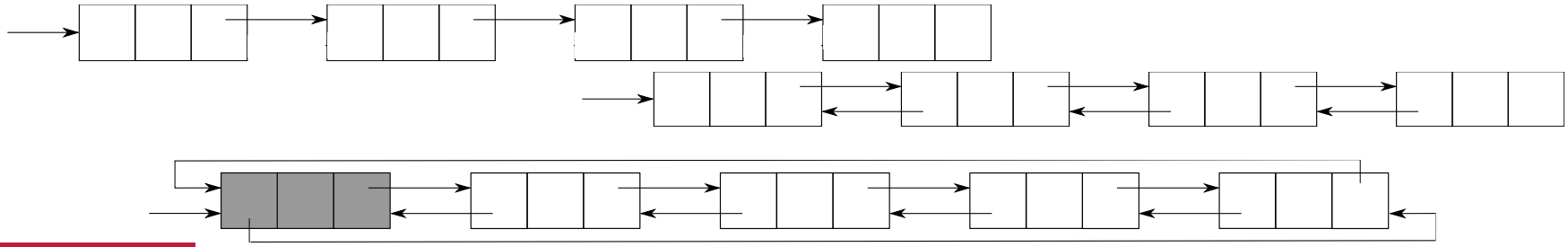
Datentyp	Stack	Queue
Einfügen	$O(1)$	$O(1)$
Nächstes Element	$O(1)$	$O(1)$
Löschen	$O(1)$	$O(1)$



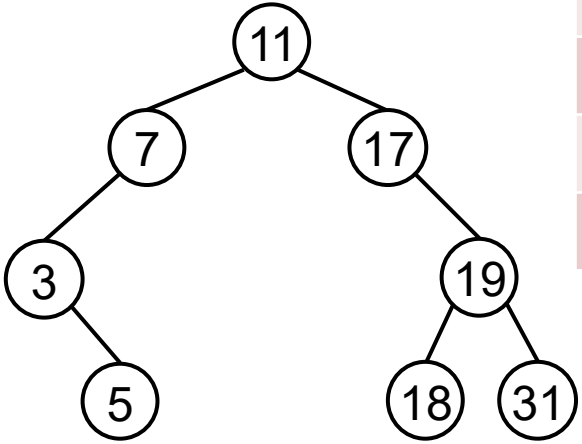


Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

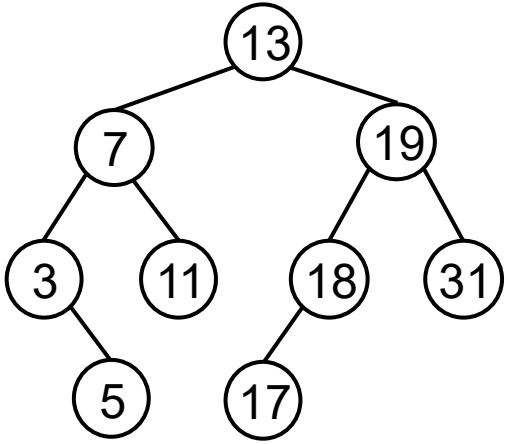
Datentyp	Listen		
Subtyp	Einfach	Doppelt	Zyklisch
Suchen	$O(n)$	$O(n)$	$O(n)$
Einfügen	$O(1)$	$O(1)$	$O(1)$
Löschen	$O(n)$	$O(1)$	$O(1)$
Traversierung	$O(n)$	$O(n)$	$O(n)$



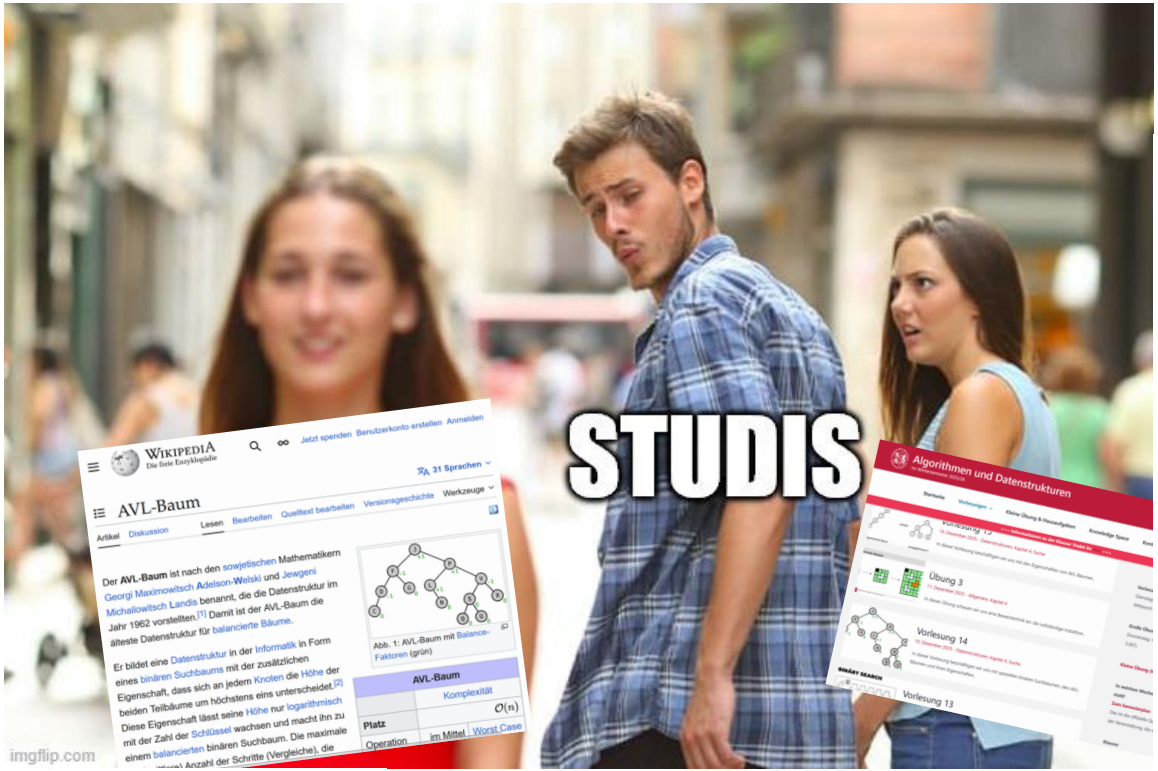
Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



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



AVL-Bäume – Restructure

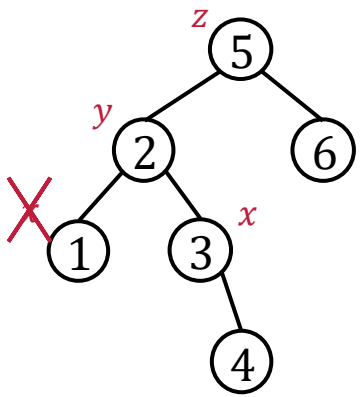


AVL-Bäume – Restructure

Bei Insert und Delete stellen sich nun folgende Fragen:

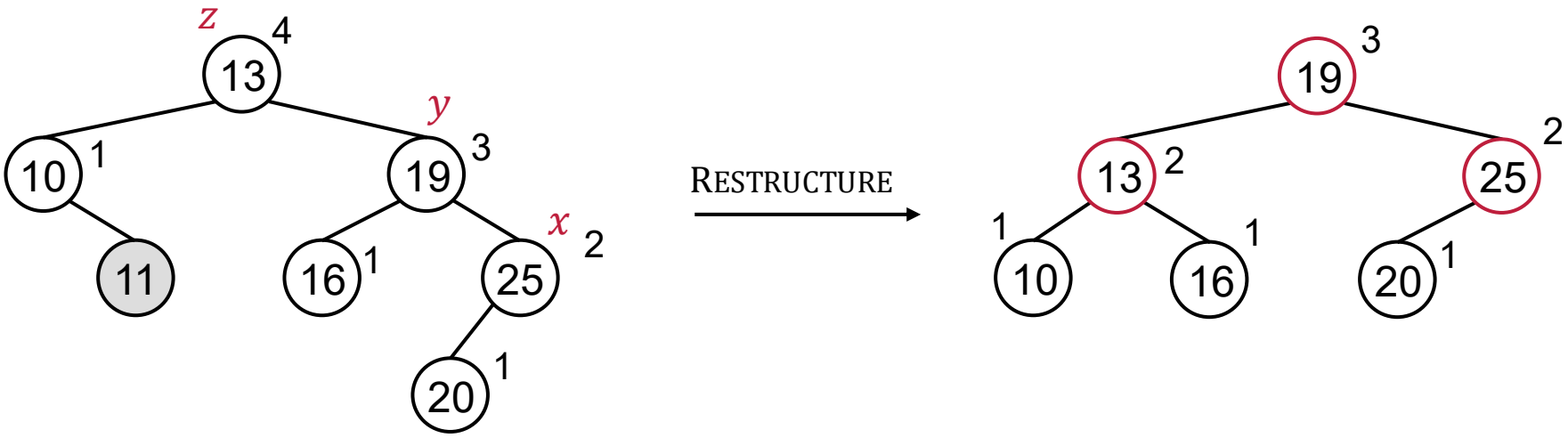
1. Welche Knoten werden unbalanciert?
2. Wie stellt man die Balance wieder her?
3. Welche Regeln sollte man berücksichtigen?

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



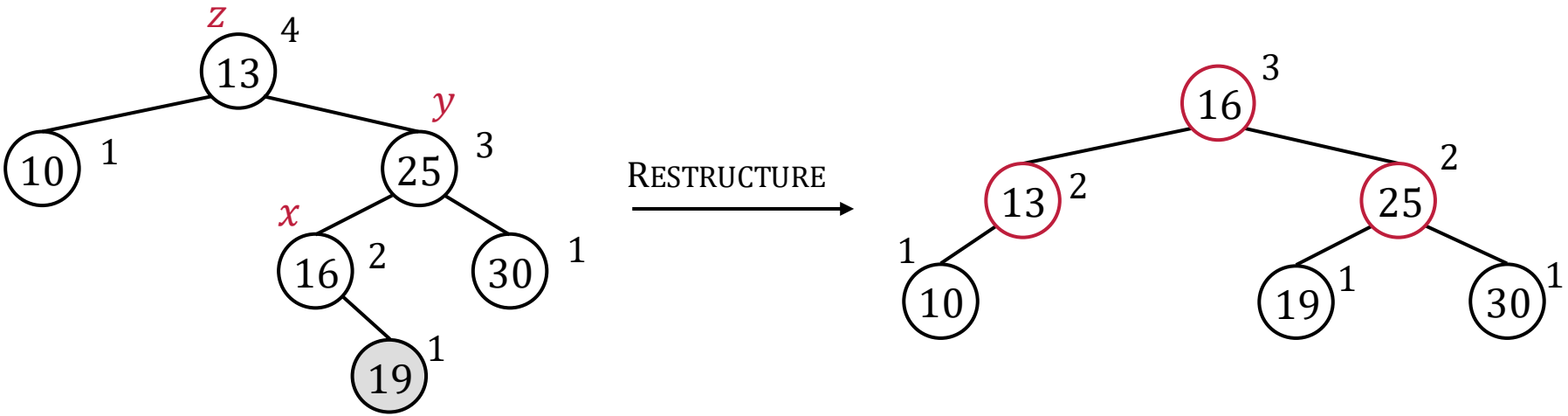
AVL-Bäume – Beispiele

DELETE$(T, 11)$



AVL-Bäume – Beispiele

INSERT$(T, 19)$



AVL Restructure - Zusammenfassung

Merkblatt!

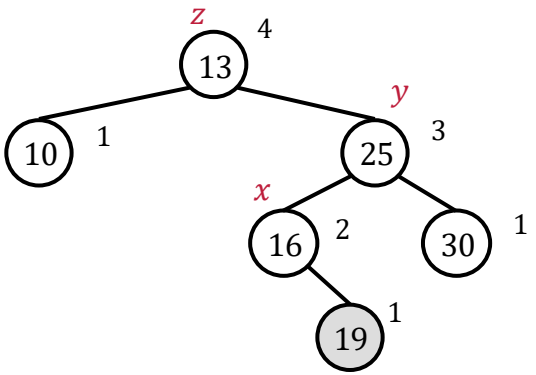
Ist nach INSERT oder DELETE die AVL-Eigenschaft verletzt, kann diese mit RESTRUCTURE wiederhergestellt werden.

Nach INSERT reicht *ein* RESTRUCTURE-Aufruf, um die Höhenbalanciertheit wiederherzustellen.

Nach DELETE können *mehrere* (genauer gesagt $O(\log n)$ viele) Aufrufe von RESTRUCTURE nötig sein.

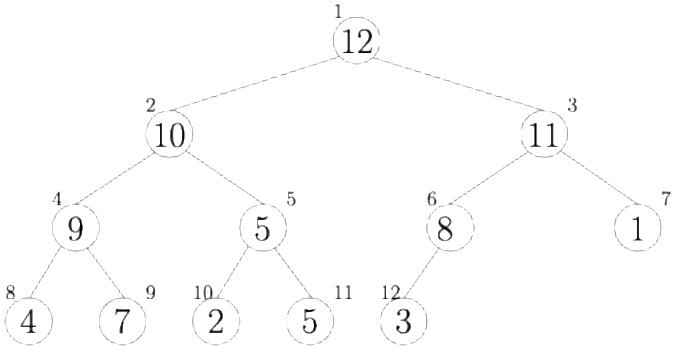
RESTRUCTURE verlangt drei Knoten x, y und z als Argumente:

- Starte beim tiefsten unbalancierten Knoten: Das ist z.
- Das Kind von z mit größerer Höhe ist y.
- Das Kind von y mit größerer Höhe ist x.



Laufzeiten – dynamische Datenstrukturen (partielle Sortierung)

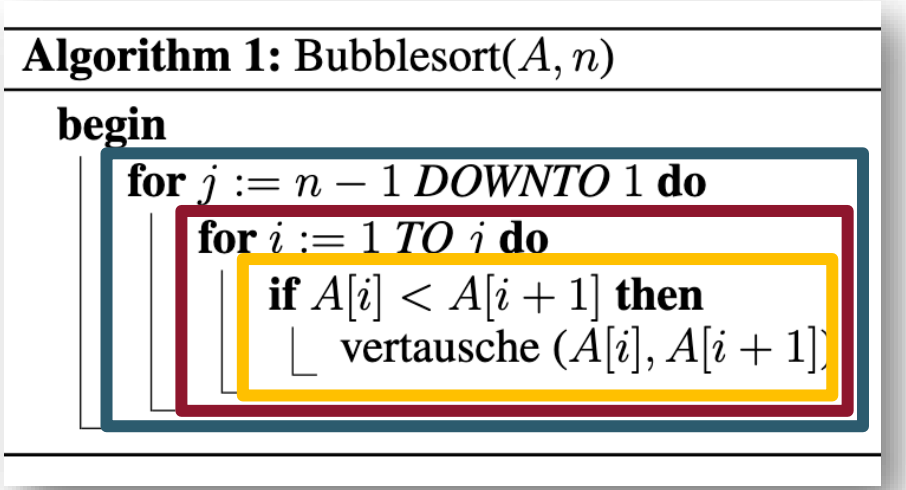
Datentyp	(Max)Heaps
Einfügen	$O(\log n)$
Löschen	$O(\log n)$
Minimum/Maximum	$O(1)$
Extrahiere Min/Max	$O(\log n)$



$$A = \overset{1}{[12}, \overset{2}{10}, \overset{3}{11}, \overset{4}{9}, \overset{5}{5}, \overset{6}{8}, \overset{7}{1}, \overset{8}{4}, \overset{9}{7}, \overset{10}{2}, \overset{11}{5}, \overset{12}{3}]$$

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:

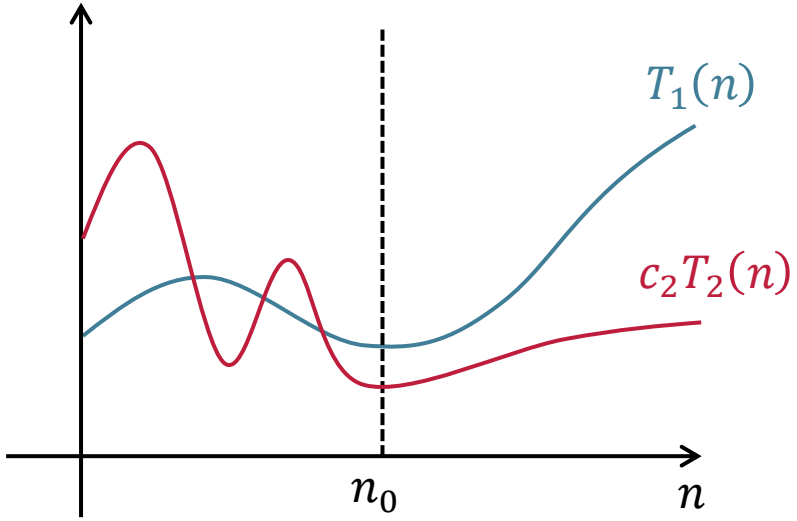
Es gibt Konstanten $n_0 \in \mathbb{N}$ und $c_2 \in \mathbb{R}^+$,

sodass für alle $n \geq n_0$ gilt:

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

$$\Leftrightarrow$$

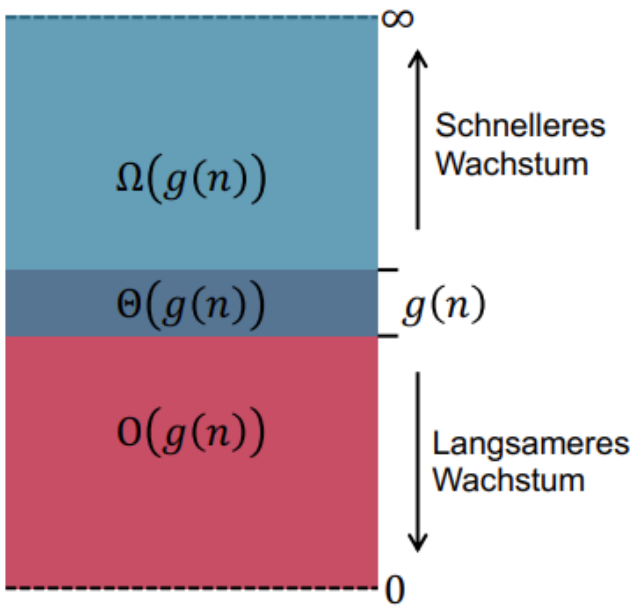
$$T_1(n) \in \Omega\big(T_2(n)\big)$$



„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

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

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! →

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

Tipps für Wachstumsanalyse

Mit „Abuse of Notation“:

$$O\big(f(n)\big) + O\big(g(n)\big) = O(\max\big(f(n), g(n)\big)$$

$$O\big(f(n)\big) \cdot O\big(g(n)\big) = O\big(f(n) \cdot g(n)\big)$$

$$\Omega\big(f(n)\big) + \Omega\big(g(n)\big) = \Omega(\max\big(f(n), g(n)\big)$$

$$\Omega\big(f(n)\big) \cdot \Omega\big(g(n)\big) = \Omega\big(f(n) \cdot g(n)\big)$$

Tipps für Wachstumsanalyse

$O\big(f(n)\big) - O\big(g(n)\big) = O\big(f(n)\big)$?

Achtung: Subtrahieren funktioniert so nur, wenn $f(n) \notin O\big(g(n)\big)$!

Damit also:

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

Wie erlangt man die Konstanten?

$$\frac{\Theta(n^2) - \Theta(n\log n) + \Theta(n) - \Theta(1)}{\Theta(n\log n) - \Theta(n)} \cdot \Theta(\log n) = \frac{\Theta(n^2)}{\Theta(n\log n)} \cdot \Theta(\log n)$$

$$= \Theta\left(\frac{n^2}{n\log n} \cdot \log n\right) = \Theta(n)$$

Wachstum von Funktionen

$f(n)$	$O(n)$	$\Omega(n)$	$O(n \log n)$	$\Omega(n \log n)$	$O(n^2)$	$\Omega(n^2)$
$3n + 6$	✖	✖	✖		✖	
$19n^2 - 30n$		✖		✖	✖	✖
2^n		✖		✖		✖
$\log(n!)$		✖	✖	✖	✖	
$\frac{12n^2-5}{3n \log n+5}$	✖		✖		✖	

Wachstum von Funktionen (Beweise)

Satz 3.12

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

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

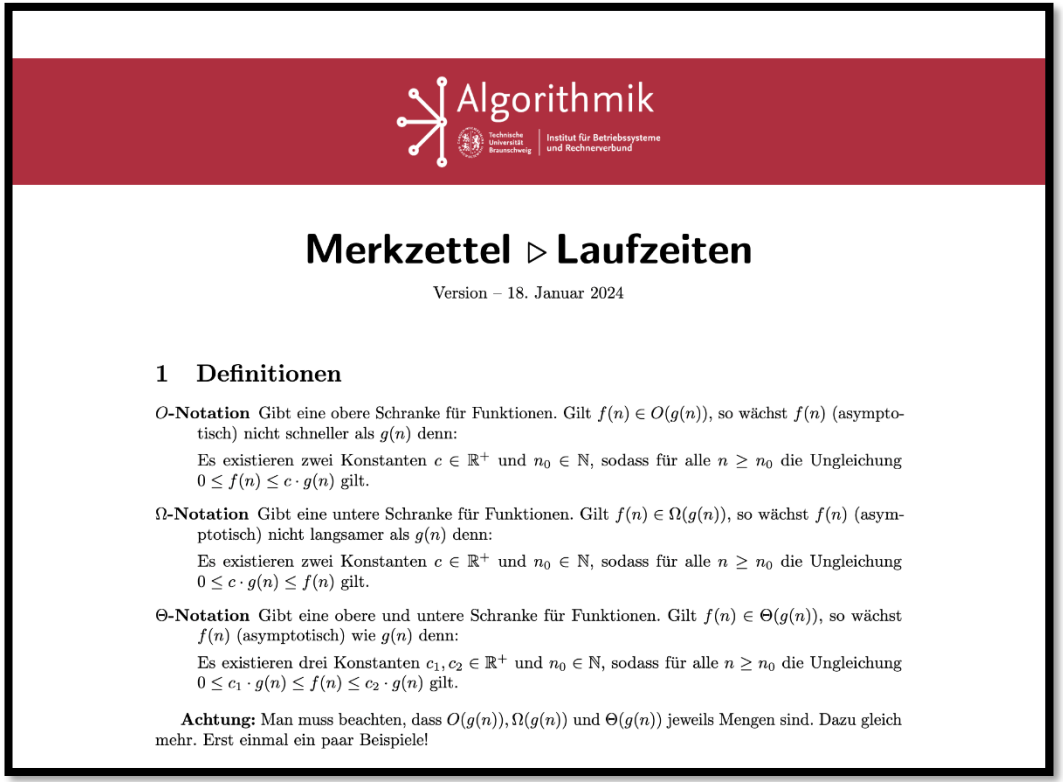
$\Leftrightarrow \exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}: \forall n \geq n_0: \quad 0 \leq f(n) \leq c \cdot g(n)$

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

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

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

Wachstum von Funktionen



Algorithmik

Technische Universität Braunschweig | Institut für Betriebssysteme und Rechnerverbund

Merkzettel ▷ Laufzeiten

Version – 18. Januar 2024

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 $0 \leq c \cdot g(n) \leq f(n)$ gilt.

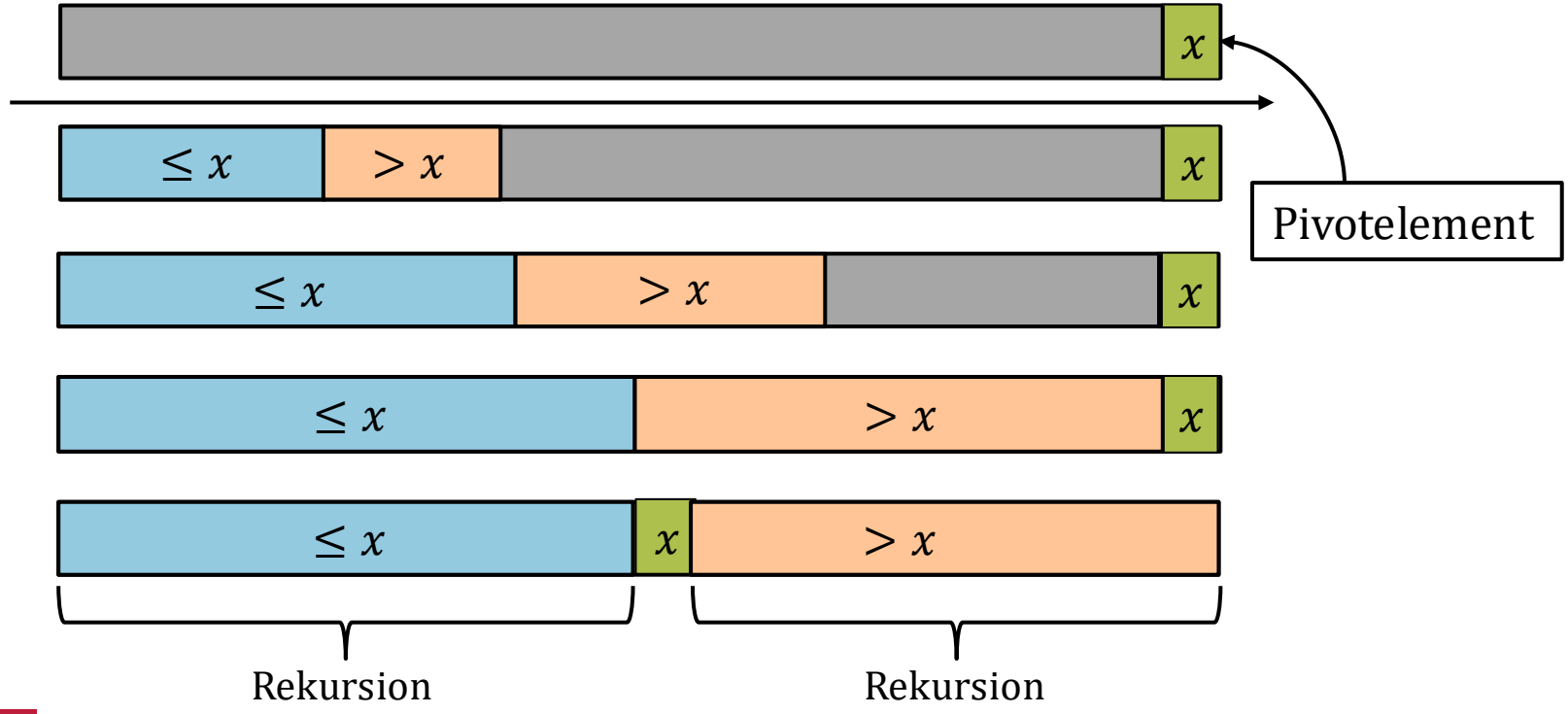
Θ-Notation Gibt eine obere und untere Schranke für Funktionen. Gilt $f(n) \in \Theta(g(n))$, so wächst $f(n)$ (asymptotisch) wie $g(n)$ denn:

Es existieren drei Konstanten $c_1, c_2 \in \mathbb{R}^+$ und $n_0 \in \mathbb{N}$, sodass für alle $n \geq n_0$ die Ungleichung $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ gilt.

Achtung: Man muss beachten, dass $O(g(n)), \Omega(g(n))$ und $\Theta(g(n))$ jeweils Mengen sind. Dazu gleich mehr. Erst einmal ein paar Beispiele!

Quicksort

Quicksort – Prinzip



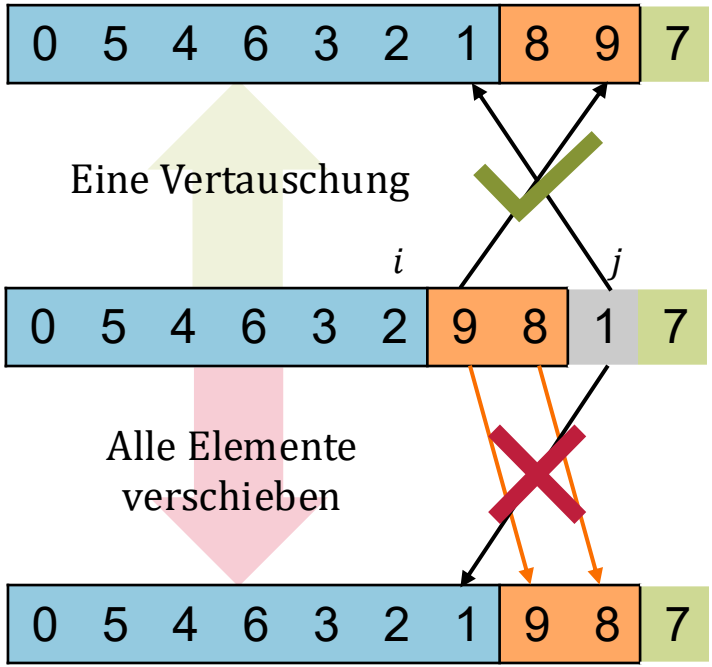
Quicksort – Details

- Innerhalb der rot und blau markierten Bereiche gibt es *keine* Sortierung! Und auch keine Garantie, dass die Elemente während des Partition-Aufrufs diese Reihenfolge behalten.
- Wir “verschieben” nicht den ganzen roten Bereich, sondern tauschen einzelne Paare, um lineare Laufzeit zu erreichen!

→ Pro Iteration reicht eine Vertauschung aus!

- Es gibt verschiedene Varianten zu pivotisieren, und daher verschiedene Versionen von Quicksort.

→ In dieser VL wird stets die hier gezeigte Variante genutzt.



Sortierverfahren

Algorithmus	Laufzeit	Idee

Sortierverfahren

Algorithmus	Laufzeit	Idee
Mergesort	$O(n \log n)$	Sortiere Teilarrays und merge sie. Beginne mit kleinstem Teilarray (z.B. der Größe 1)
Bubblesort (P-Blatt 4)	$O(n^2)$	Iteriere n-mal über das Array und vertausche falsch-stehende benachbarte Objekte.
Heapsort (5. Große Übung)	$O(n \log n)$	Betrachte Array als Heap, entferne iterativ das größte Element und stelle die Heap-Eigenschaft wieder her.
Quicksort	$O(n^2)$	Pivotisiere und sortiere rekursiv auf beiden Teilarrays weiter.
Radixsort	$O\big(d(n+k)\big)$	Sortiere Zahlen iterativ nach d Ziffern mit Werten 1 bis k.
Selectionsort	$O(n^2)$	Setze das i-t kleinste Element in der i-ten Iteration an die i-te Stelle
Insertionsort	$O(n^2)$	Setze das i-te Element des Arrays in $A[1] \dots A[i]$ an die richtige Stelle.

Mediane

Mediane – Definition

Rang-k Element m in X:

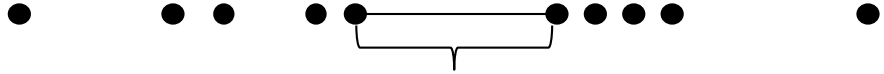
$$|\{x \in X : x \leq m\}| \geq k$$

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

Für einen Median m in X gilt:

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

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



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

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

$k = 2$

```
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

$X := \{14, 23, 15, 25, 17, 19, 20, 21, 22, 24, 16, 18\}$ $k = 2$

```
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

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

$k = 2$

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

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

$X_= = \{17\}$

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

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

```
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 ✨

Induktionsbeweise

Handshake-Lemma

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

Beweis per Induktion über

IA:

IV:

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).

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 $m \in \mathbb{N}$.
(Also: Aussage gilt für Graph mit m Kanten).

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).

Knotengerade modulo 2

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

Checkliste: Induktion

- IA?
  - Muss man mehrere Startfälle abdecken?
- IV?
  - Nicht vergessen, Gratis-Punkte!
  - Starke Induktion?
- IS?
  - Starte bei $n + 1$
  - Komme irgendwie auf n
  - Setze IV ein (dort gilt unsere Aussage, und die nutzen wir)
  - Zeige damit, dass Aussage auch für $n + 1$ gilt

Klausur

Klausur

Eckdaten

- 18.02.2026 von 08:00 Uhr bis 10:00
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Infos zur Klausur

2. Februar 2026 | Allgemein

Hey zusammen!

Die Klausur für Algorithmen und Datenstrukturen findet am **Mittwoch, den 18. Februar 2025** von **08:00-10:00 Uhr** statt.

Die Raumaufteilung werden wir spätestens am 17.02.26 auf dieser Seite und per Mailingliste bekannt geben. Die möglichen Räume sind Audimax, Bunker und die ZI-Hörsäle auf dem Grotrian (Campusplan).

Bitte beachtet die folgenden Punkte zur Klausur:

- Mitbringen:
  - Studierendenausweis (plus Lichtbildausweis, falls kein Foto auf dem Studiausweis vorhanden),
  - dokumentenechter Stift (kein Bleistift, kein rot!)
  - Kein Tippex!
  - Wörterbuch, falls ihr das benötigt
- Weitere Unterlagen oder Hilfsmittel (z.B. Taschenrechner) sind *nicht* erlaubt!
- Eigenes Papier ist *nicht* erlaubt, wir stellen Papier
- Bitte seid 15 min vor der Klausur an der Räumen

Wir wünschen euch ganz viel Erfolg!!! 🙂

Klausur

Eckdaten

- 18.02.2026 von 08:00 Uhr bis 10:00
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Vorbereitung

- Hausaufgaben
- Tutorien (Tutor*innen fragen)
- Vorlesungsvideos & Folien
- Skript
- Alte Klausuren

Bei Fragen …

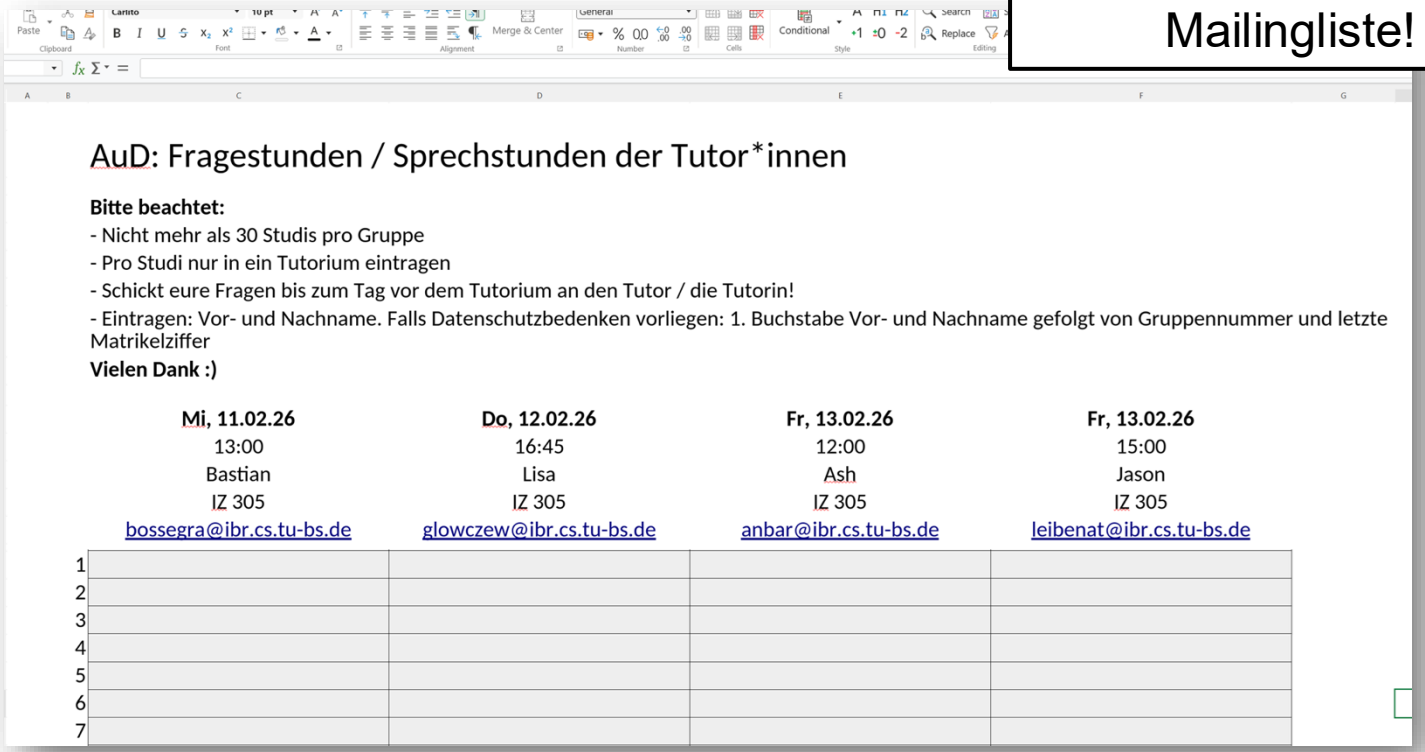
Bei Fragen per Mail:
Erstmal an eure Tutor*innen.

Dabei beachten:
Fragen wie „Ich habe Thema X nicht verstanden. Erklär das noch mal!“ helfen weder euch, noch uns! → Konkrete Fragen :)

Zusatztutorien

Zusatztutorien

Link zur Anmeldung per Mailingliste!



AuD: Fragestunden / Sprechstunden der Tutor*innen

Bitte beachtet:

- Nicht mehr als 30 Studis pro Gruppe
- Pro Studi nur in ein Tutorium eintragen
- Schickt eure Fragen bis zum Tag vor dem Tutorium an den Tutor / die Tutorin!
- Eintragen: Vor- und Nachname. Falls Datenschutzbedenken vorliegen: 1. Buchstabe Vor- und Nachname gefolgt von Gruppennummer und letzte Matrikelziffer

Vielen Dank :)

	Mi, 11.02.26	**Do, 12.02.26**	**Fr, 13.02.26**	**Fr, 13.02.26**
	13:00	16:45	12:00	15:00
	Bastian	Lisa	Ash	Jason
	IZ 305	IZ 305	IZ 305	IZ 305
	bossegra@ibr.cs.tu-bs.de	glowczew@ibr.cs.tu-bs.de	anbar@ibr.cs.tu-bs.de	leibenat@ibr.cs.tu-bs.de
1				
2				
3				
4				
5				
6				
7				

Veranstaltungen im Sommer

Veranstaltung	Empfehlung	Inhalte
Algorithmen und Datenstrukturen 2 Prof. Sándor Fekete (Wahlpflicht Informatik)	AuD 1 bestanden (2. Semester)	Optimierungsprobleme Algorithmische Ansätze Komplexitätstheorie
Theoretische Informatik 2 Dr. Arne Schmidt (Pflicht für Informatikys)	Theo Inf 1 bestanden (4. Semester)	Turing Maschinen Berechenbarkeit Komplexitätstheorie
Einführung in alg. Geometrie Dr. Arne Schmidt (Wahlpflicht Informatik)	AuD 1 bestanden (ggf. Theo Inf 2 bestanden)	Range Queries Lokalisierung Motion Planning Kunstgallerie-Problem

Fragerunde

Viel Erfolg!!!