Technische Universität Braunschweig

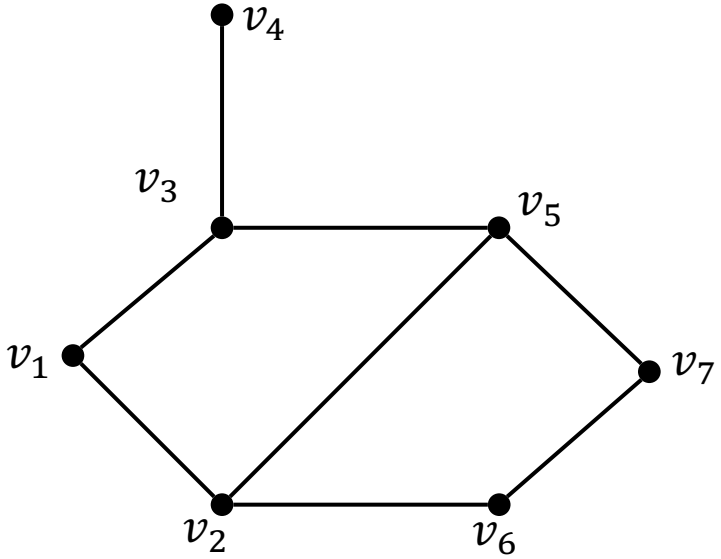
Algorithmen und Datenstrukturen – Übung #7

Fragestunde

Ramin Kosfeld und Chek-Manh Loi

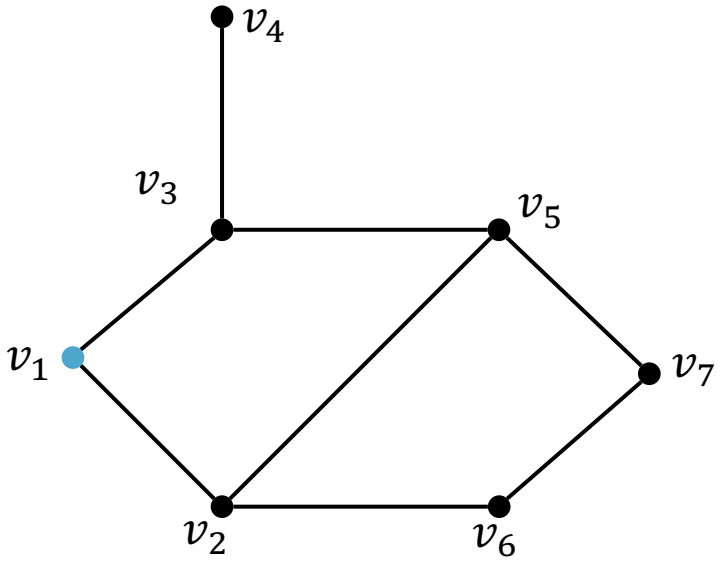
30.01.2025

Breitensuche



Breitensuche

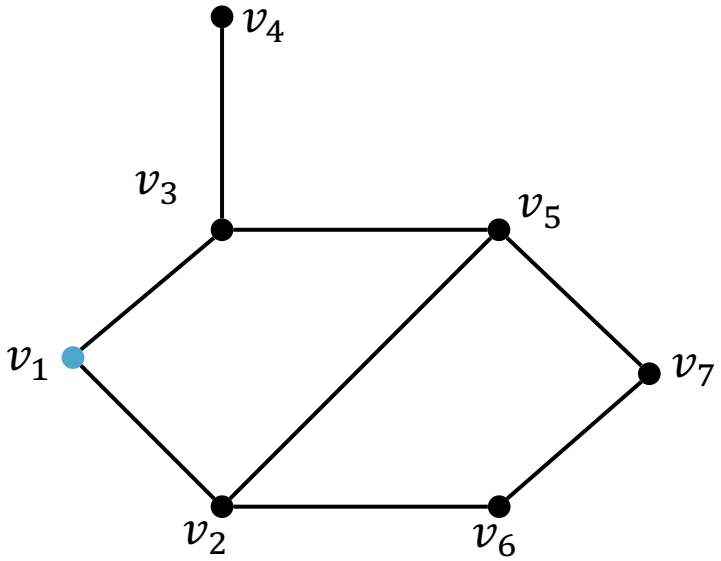
v_1



Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

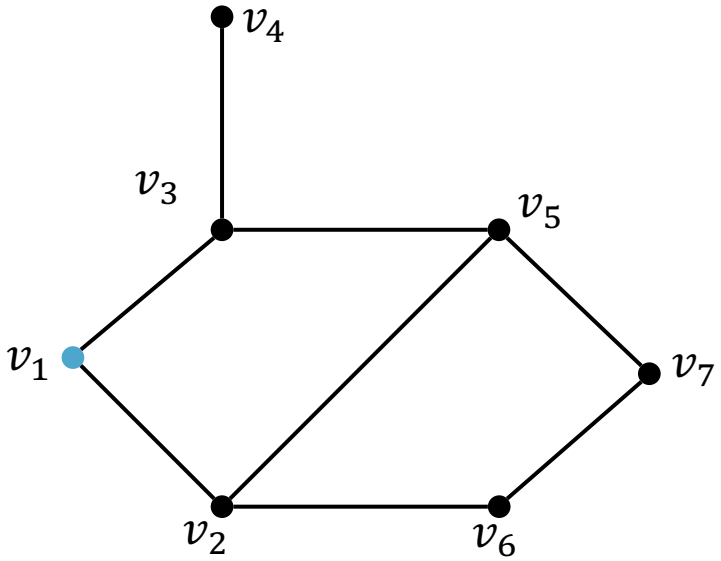
v_1



Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

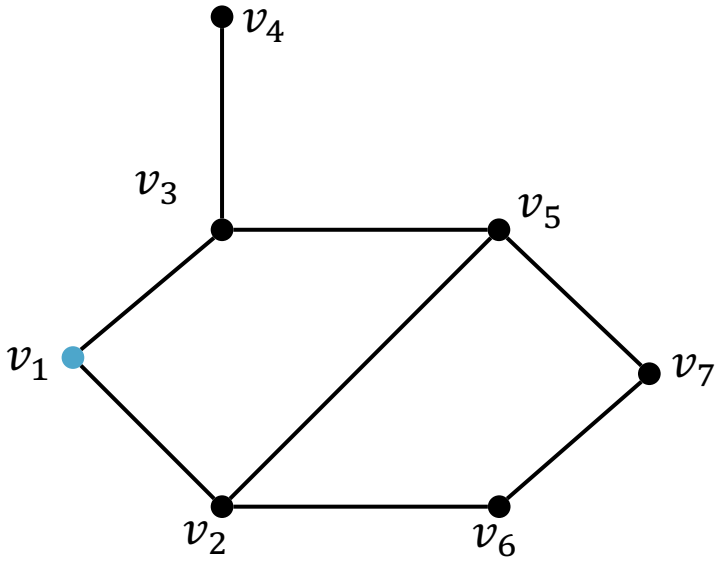
v_1 ← in



Breitensuche

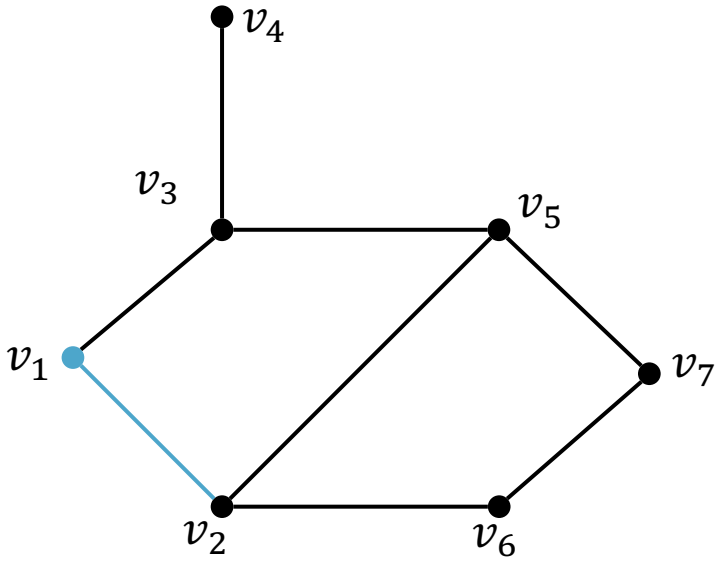
Benutze Warteschlange
Prinzip: First-in-first-out

← out v_1 ← in



Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

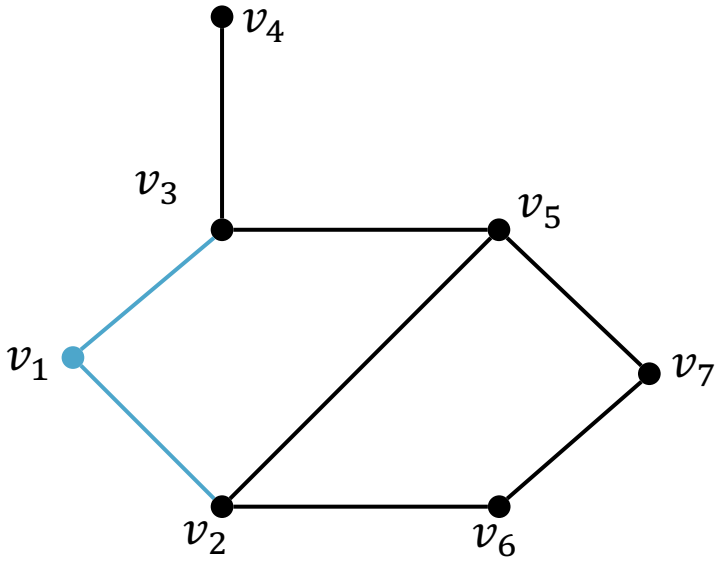


← out v_1 ← in

v_1, v_2

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out



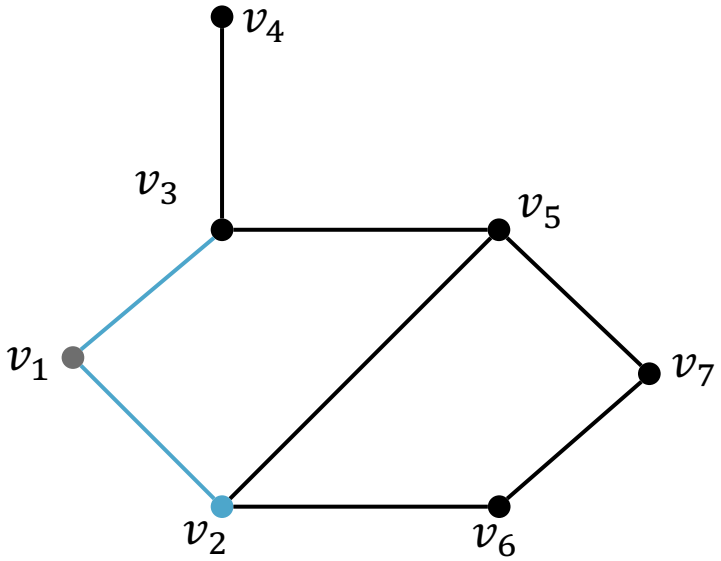
← out v_1 ← in

v_1, v_2

v_1, v_2, v_3

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out



← out ← in

v_1

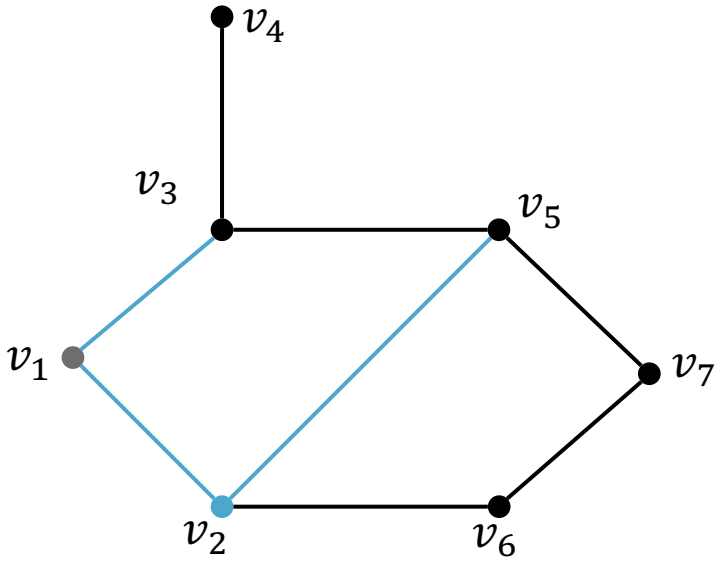
v_1, v_2

v_1, v_2, v_3

v_2, v_3

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out



out ← v_1 ← in

v_1, v_2

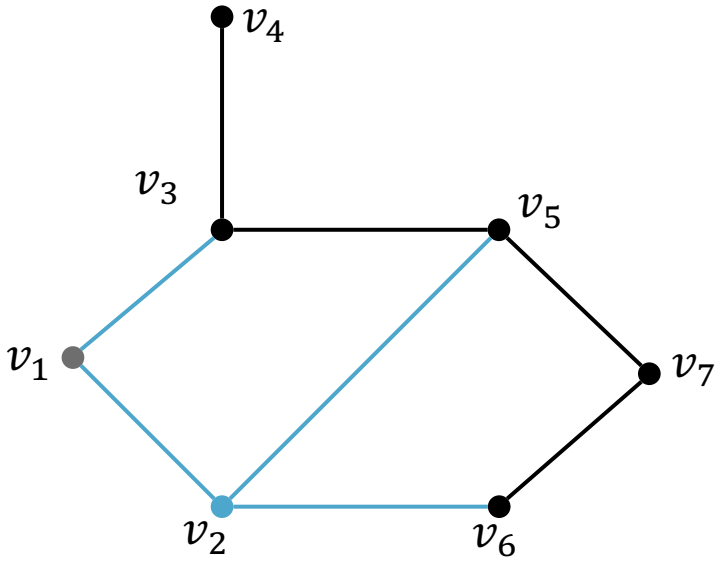
v_1, v_2, v_3

v_2, v_3

v_2, v_3, v_5

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out



← out v_1 ← in

v_1, v_2

v_1, v_2, v_3

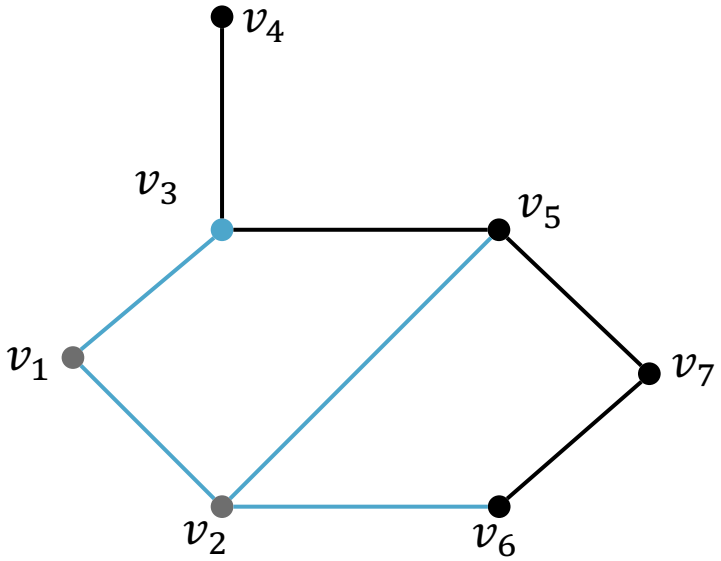
v_2, v_3

v_2, v_3, v_5

v_2, v_3, v_5, v_6

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

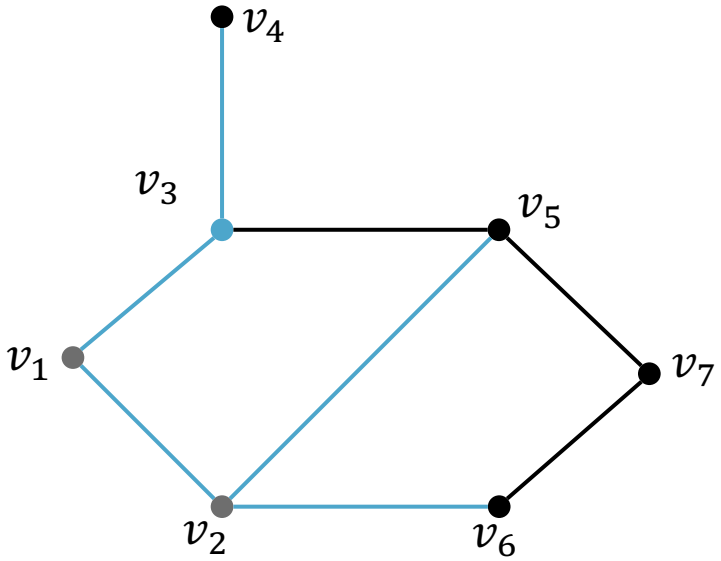


← out ← in

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

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

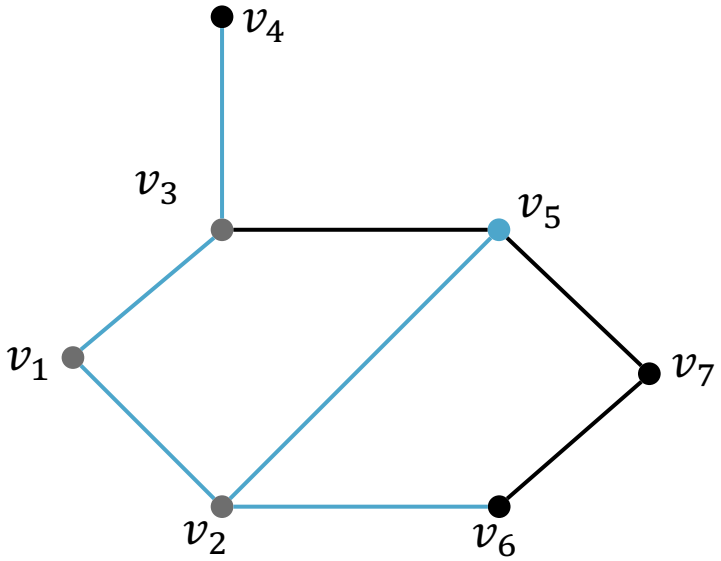


← out ← in

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

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out



out ← v_1 ← in

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

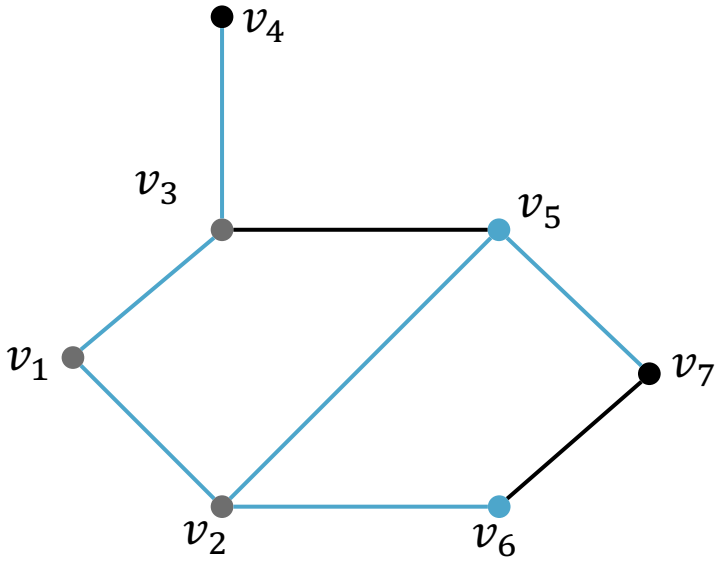
v_3, v_5, v_6

v_3, v_5, v_6, v_4

v_5, v_6, v_4

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out



out ← v_1 ← in

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

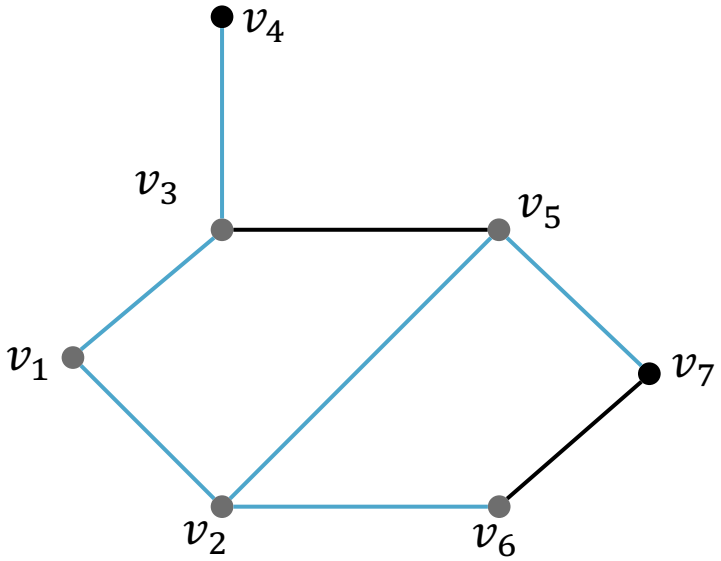
v_3, v_5, v_6, v_4

v_5, v_6, v_4

v_5, v_6, v_4, v_7

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

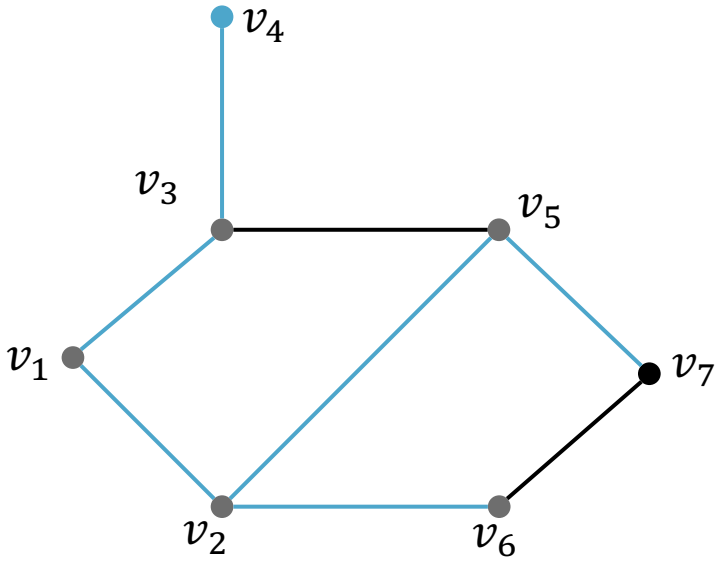


← out ← in

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

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

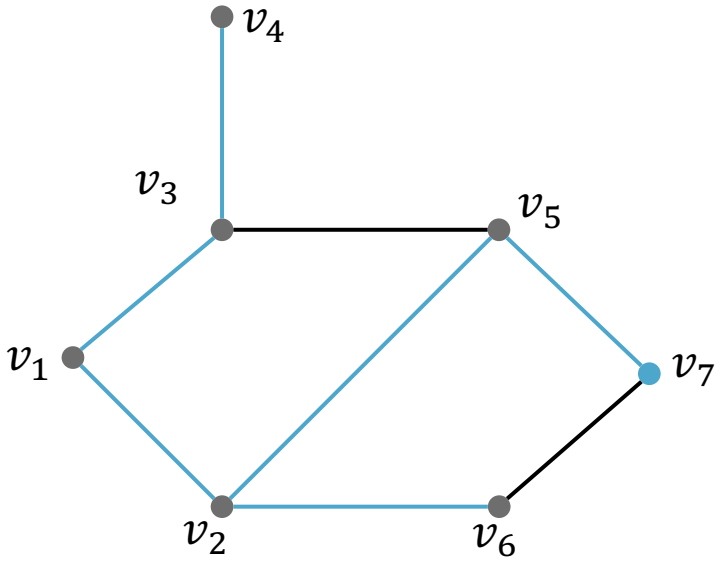


← out ← in

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

Breitensuche

Benutze Warteschlange
Prinzip: First-in-first-out

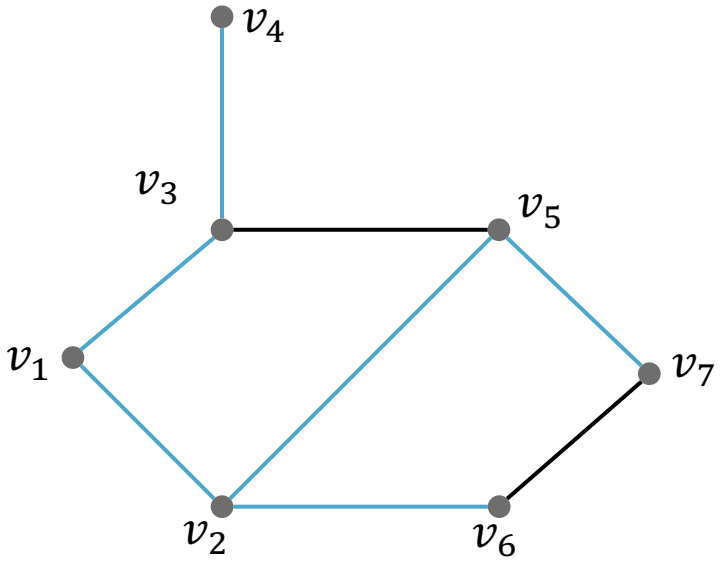


← out ← in

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

Breitensuche

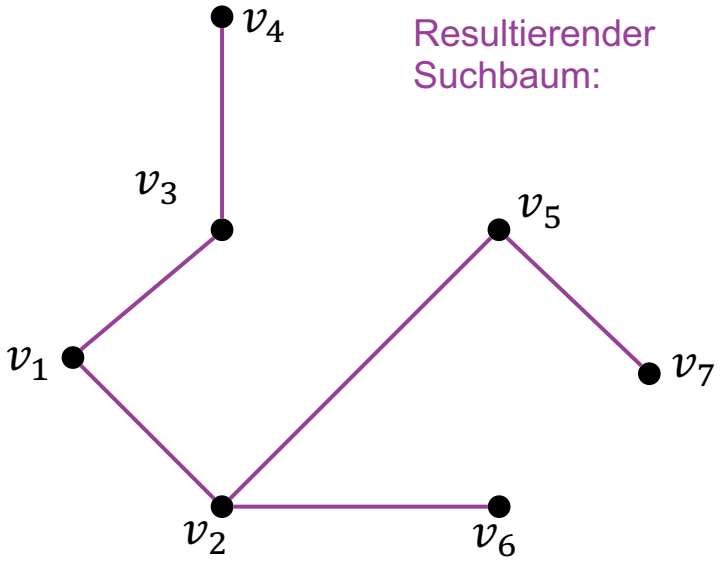
Benutze Warteschlange
Prinzip: First-in-first-out



← 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

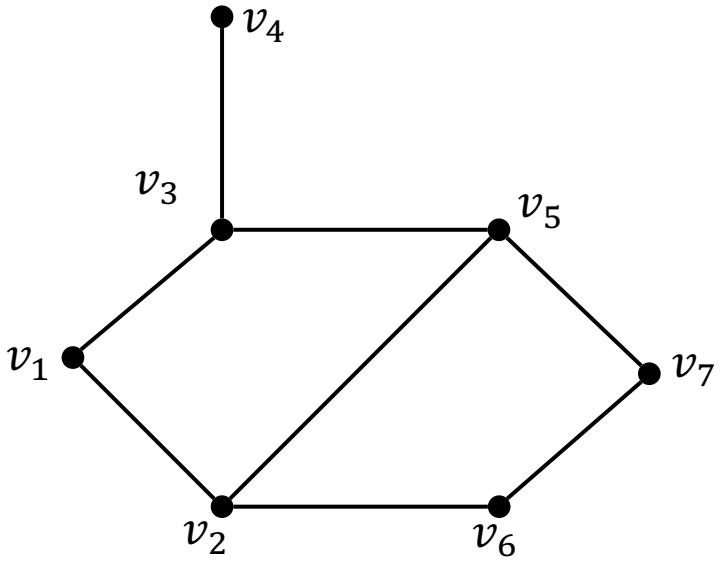


Resultierender Suchbaum:

← 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

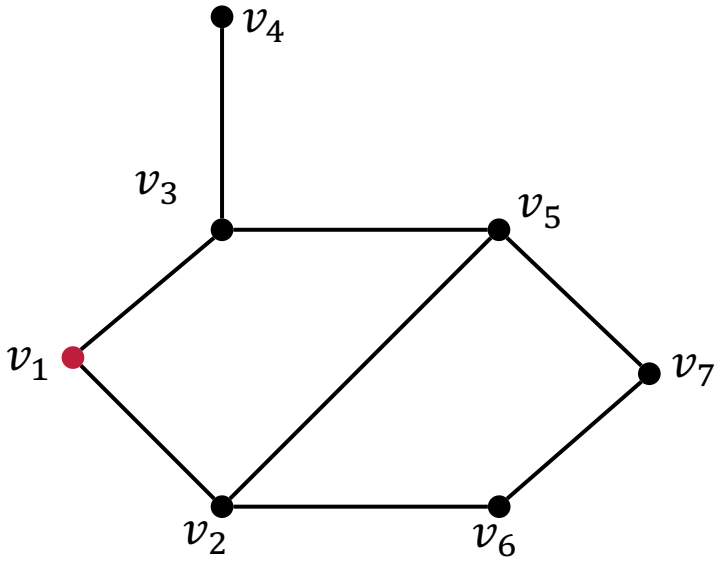
Benutze Stapel
Prinzip: Last-in-first-out



Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out

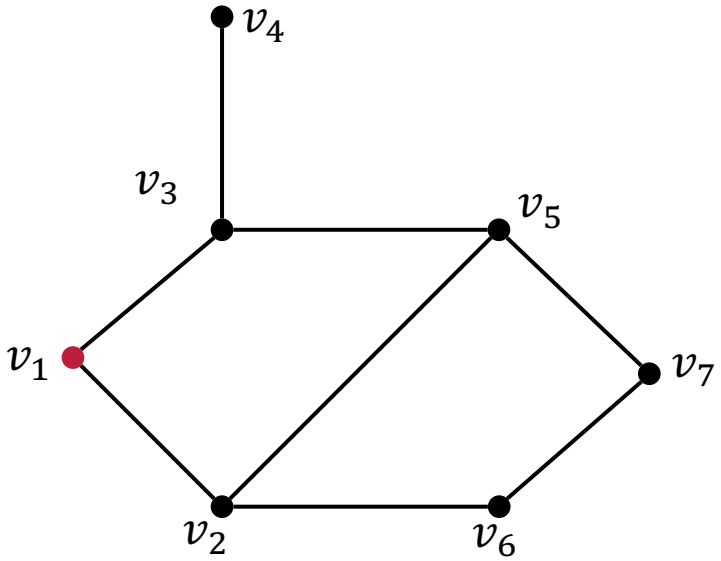
v_1



Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out

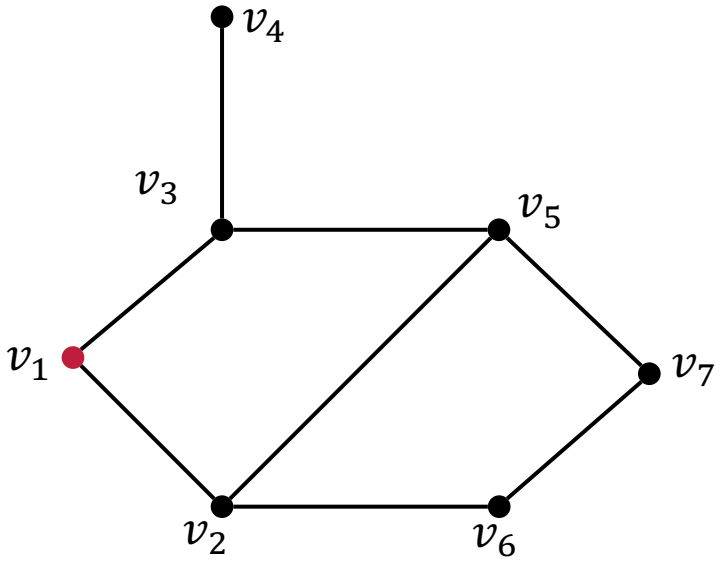
v_1 ← in



Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out

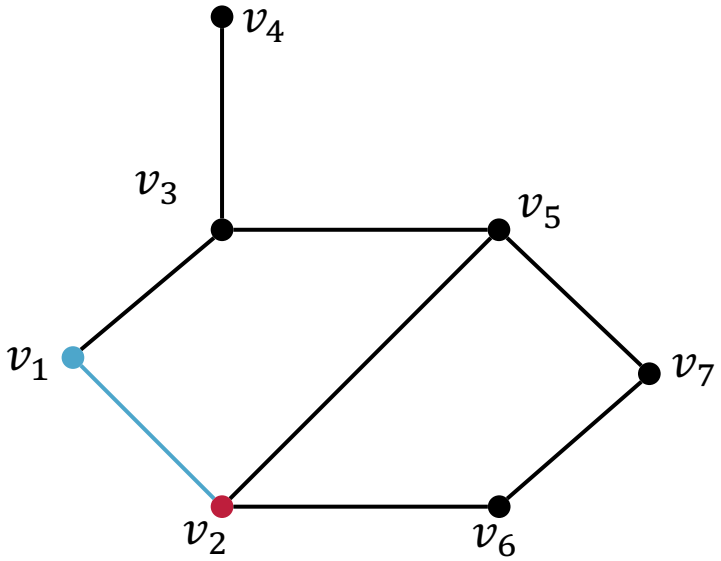
v_1 in out



Tiefensuche

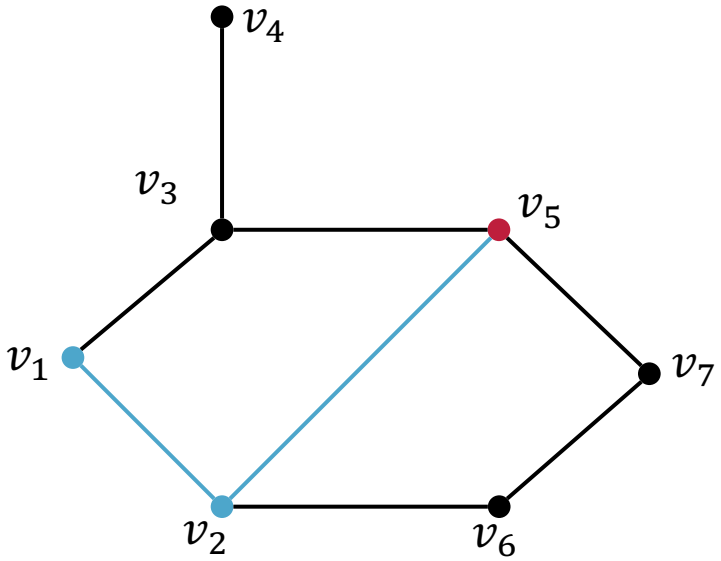
Benutze Stapel
Prinzip: Last-in-first-out

v_1 ← in
v_1, v_2 → out



Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out



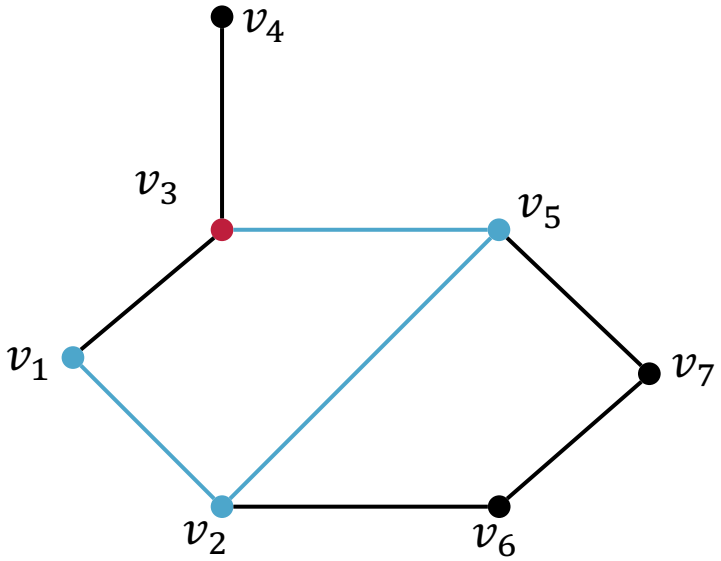
v_1 ← in
→ out

v_1, v_2

v_1, v_2, v_5

Tiefensuche

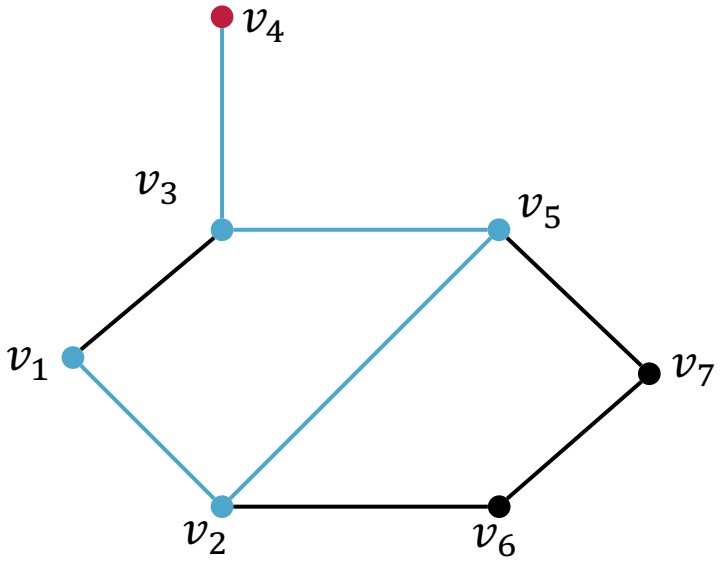
Benutze Stapel
Prinzip: Last-in-first-out



v_1 ← in / → out
v_1, v_2
v_1, v_2, v_5
v_1, v_2, v_5, v_3

Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out



v_1 ← in
→ out

v_1, v_2

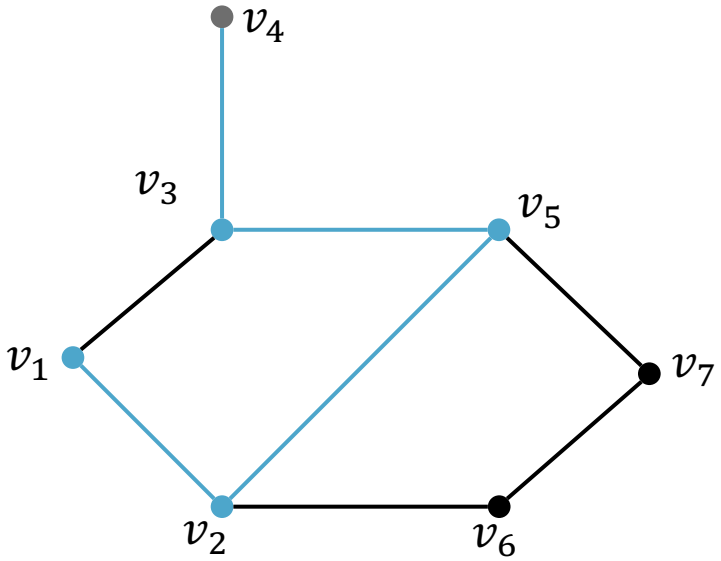
v_1, v_2, v_5

v_1, v_2, v_5, v_3

v_1, v_2, v_5, v_3, v_4

Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out



v_1 ← in
→ out

v_1, v_2

v_1, v_2, v_5

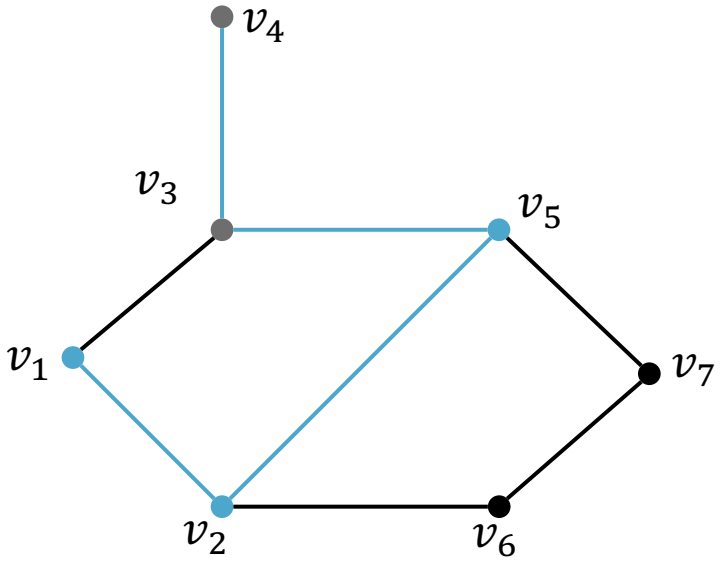
v_1, v_2, v_5, v_3

v_1, v_2, v_5, v_3, v_4

v_1, v_2, v_5, v_3

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

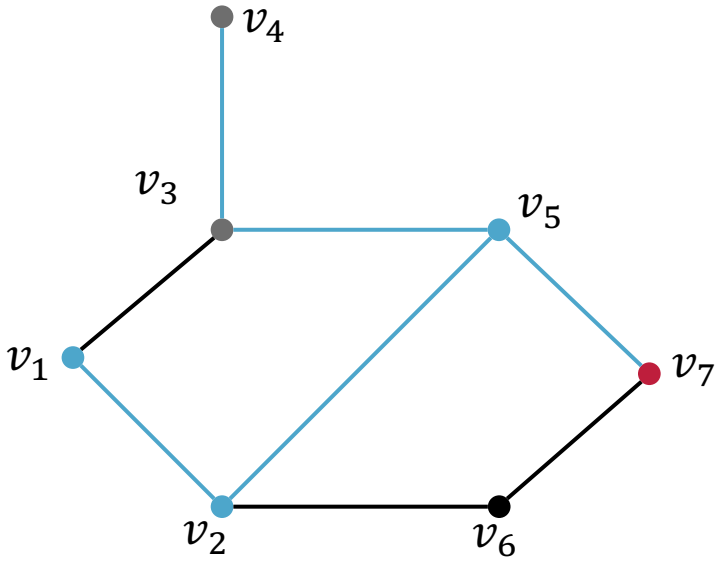
v_1, v_2, v_5, v_3, v_4

v_1, v_2, v_5, v_3

v_1, v_2, v_5

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

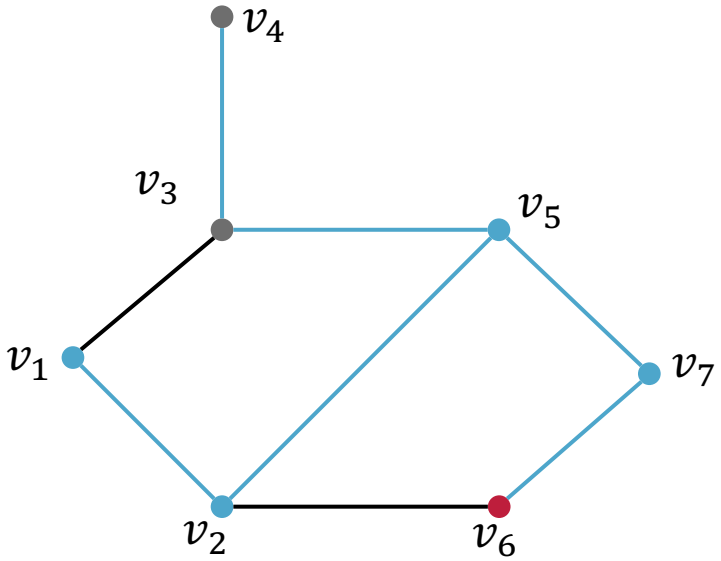
v_1, v_2, v_5, v_3

v_1, v_2, v_5

v_1, v_2, v_5, v_7

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

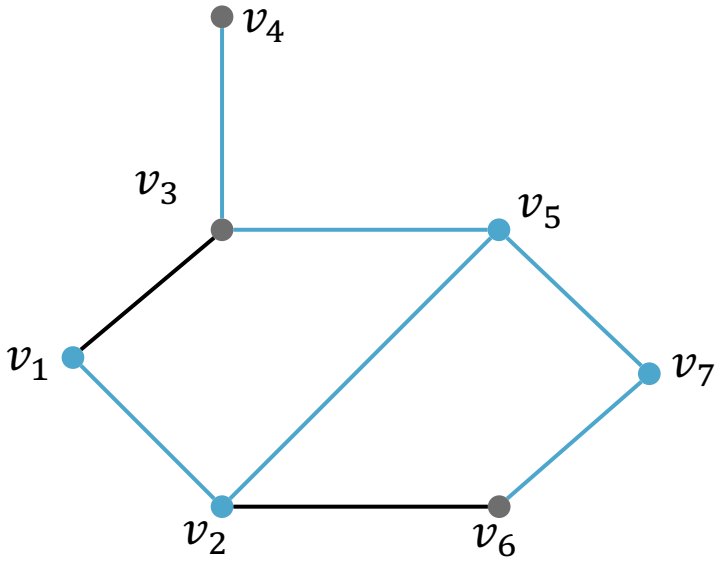
v_1, v_2, v_5

v_1, v_2, v_5, v_7

v_1, v_2, v_5, v_7, v_6

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

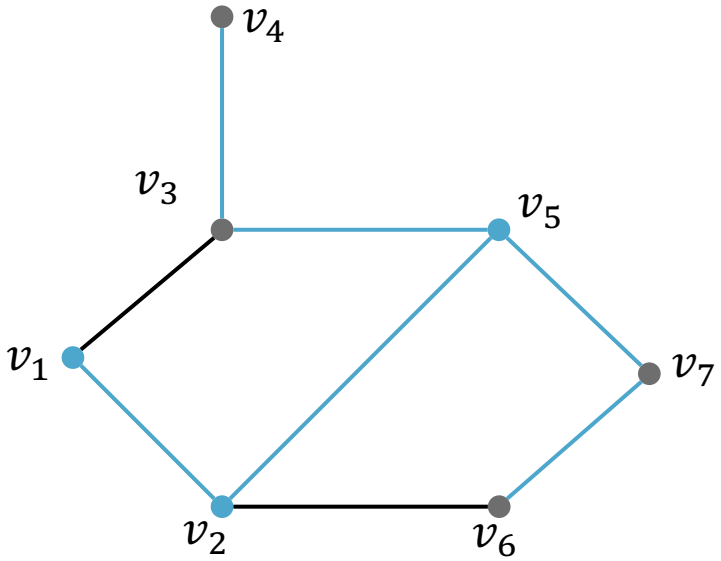
v_1, v_2, v_5, v_7

v_1, v_2, v_5, v_7, v_6

v_1, v_2, v_5, v_7

Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out

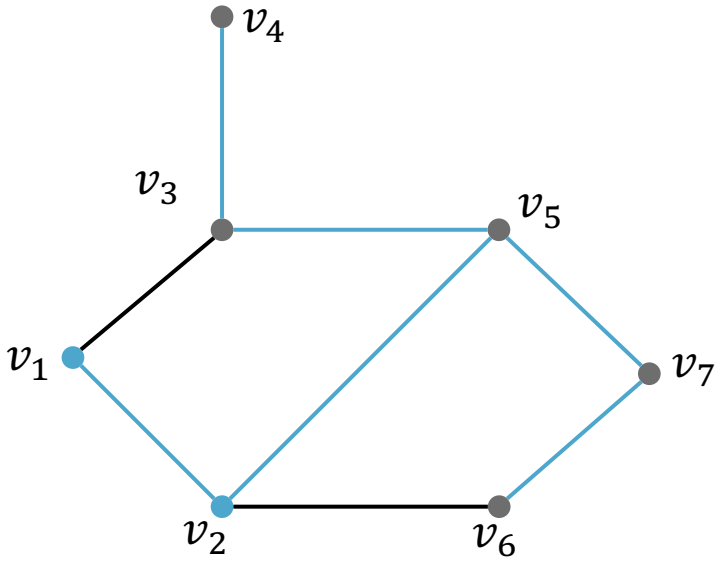


in
out

v_1
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

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

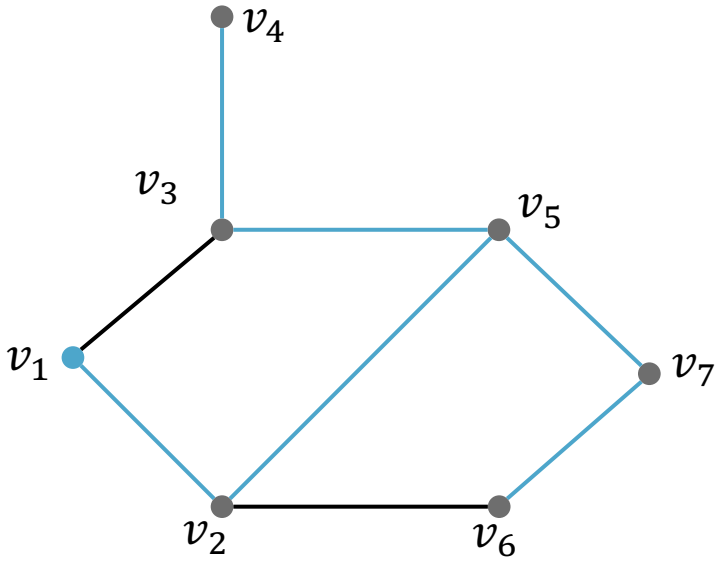
v_1, v_2, v_5, v_7

v_1, v_2, v_5

v_1, v_2

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

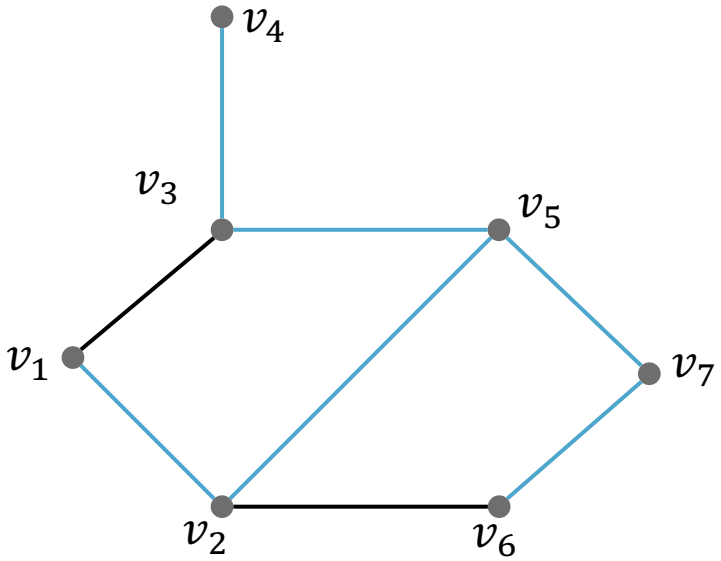
v_1, v_2, v_5

v_1, v_2

v_1

Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out



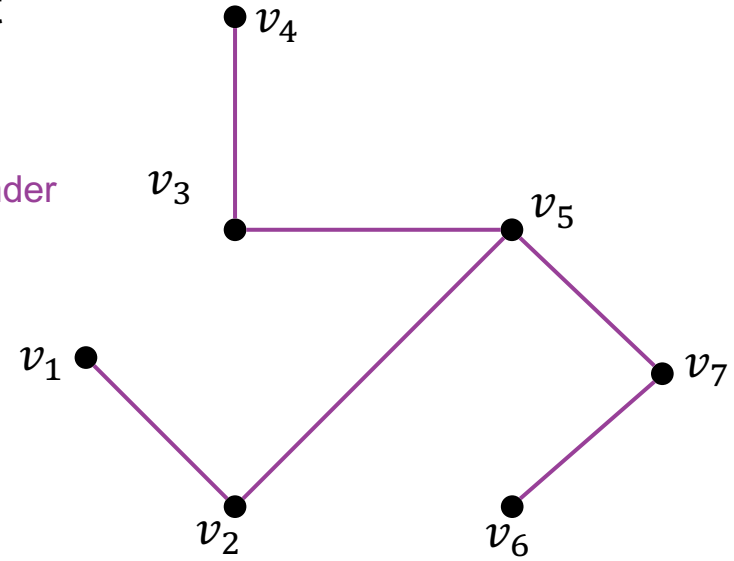
in
out

v_1
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

Resultierender Suchbaum:



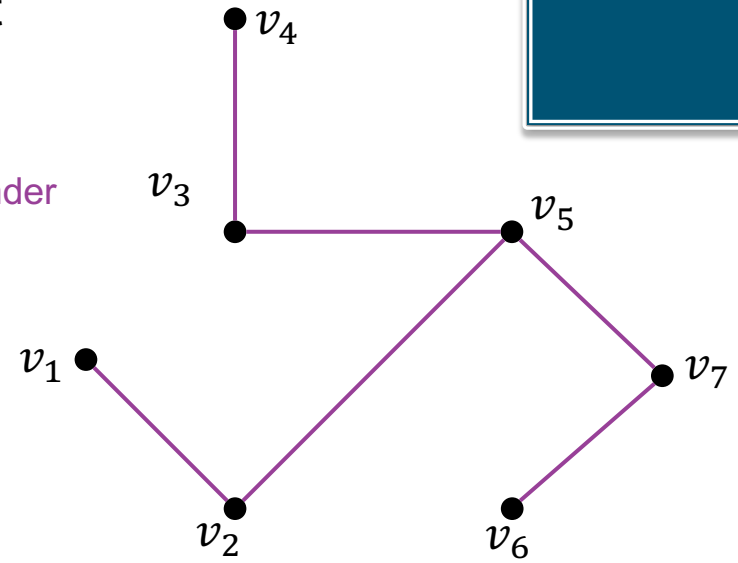
in
out

v_1
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

Resultierender Suchbaum:



Beliebte Fehler

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
$\emptyset$

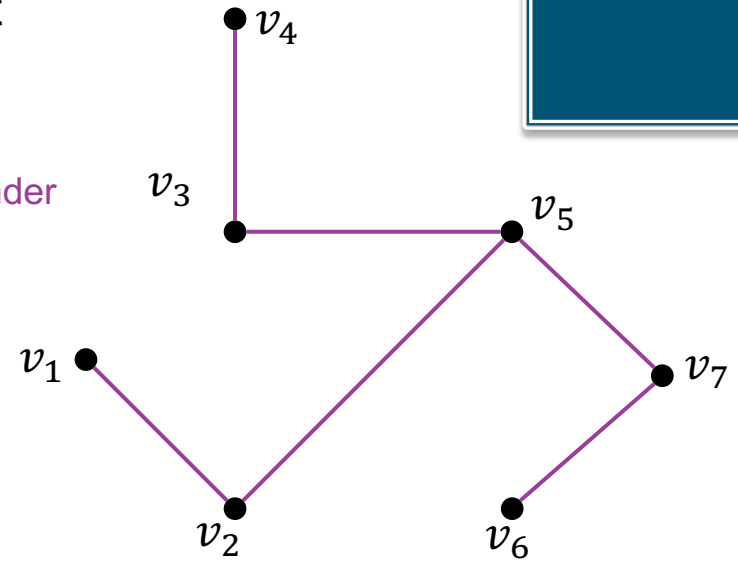
Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out

Beliebte Fehler

- Leere Menge am Ende vergessen

Resultierender Suchbaum:



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
$\emptyset$

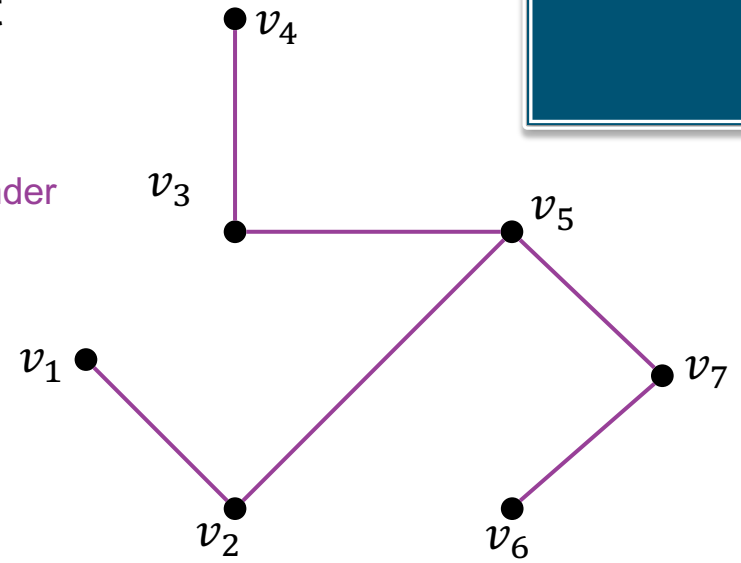
Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out

Beliebte Fehler

- Leere Menge am Ende vergessen
- Nicht jede Änderung angegeben

Resultierender Suchbaum:



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
$\emptyset$

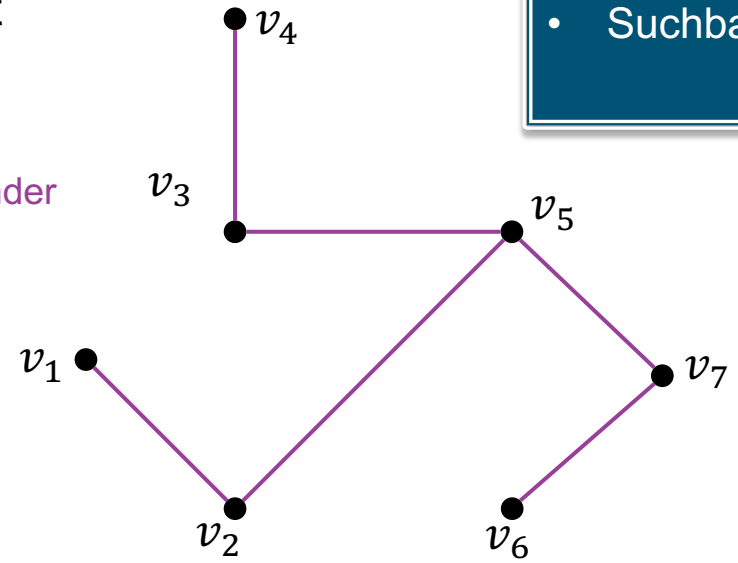
Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out

Beliebte Fehler

- Leere Menge am Ende vergessen
- Nicht jede Änderung angegeben
- Suchbaum nicht angegeben

Resultierender Suchbaum:



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
$\emptyset$

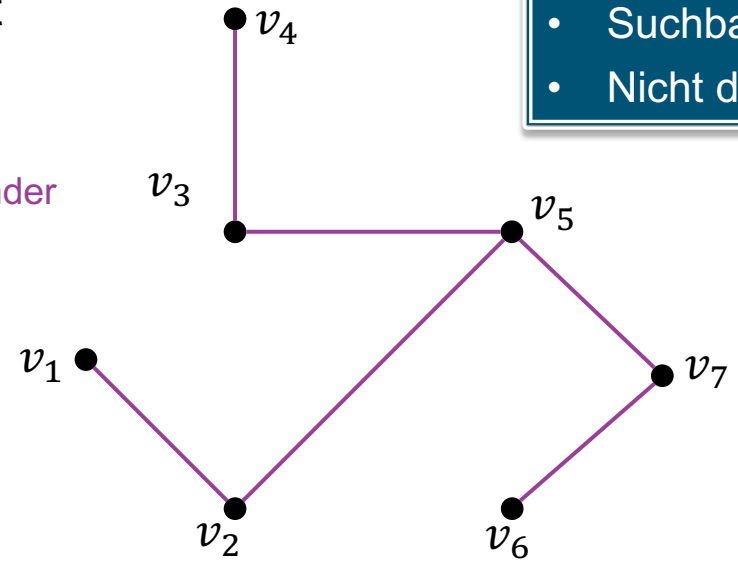
Tiefensuche

Benutze Stapel
Prinzip: Last-in-first-out

Beliebte Fehler

- Leere Menge am Ende vergessen
- Nicht jede Änderung angegeben
- Suchbaum nicht angegeben
- Nicht den kleinsten Index beachtet

Resultierender Suchbaum:

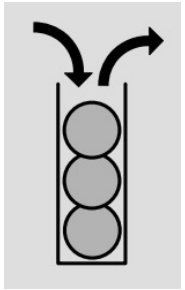


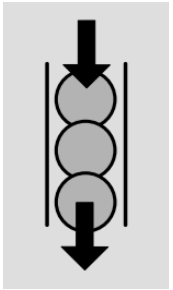
v_1, v_2, v_5, v_3, v_4
v_1, v_2, v_5, v_3
v_1, v_2, v_5
v_1, v_2, v_5, v_7
v_1, v_2, v_5, v_7, v_6
v_1, v_2, v_5, v_7
v_1, v_2, v_5
v_1, v_2
v_1
$\emptyset$

Laufzeiten
dynamische Datenstrukturen

Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

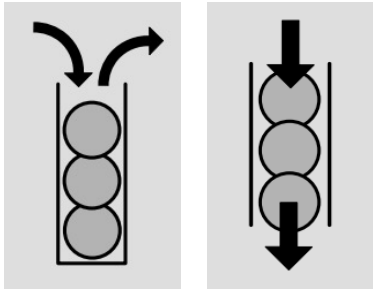
Datentyp	Stack	Queue
Einfügen		
Nächstes Element		
Löschen		





Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

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

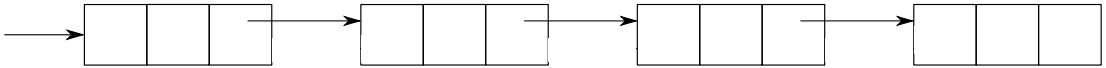


Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

Datentyp	Listen		
Subtyp	Einfach	Doppelt	Zyklisch
Suchen			
Einfügen			
Löschen			
Traversierung			

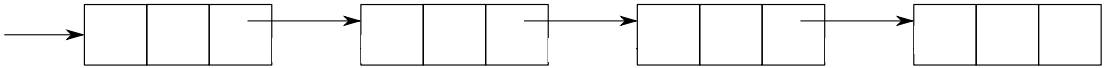
Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

Datentyp	Listen		
Subtyp	Einfach	Doppelt	Zyklisch
Suchen			
Einfügen			
Löschen			
Traversierung			



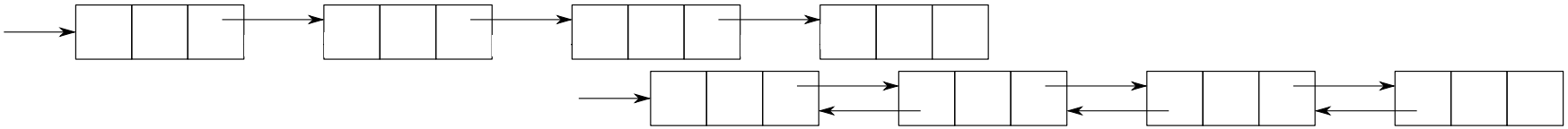
Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

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



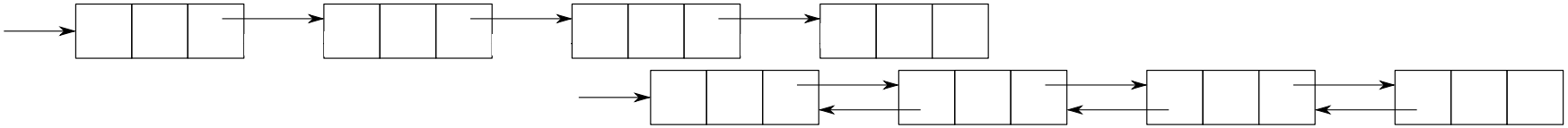
Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

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



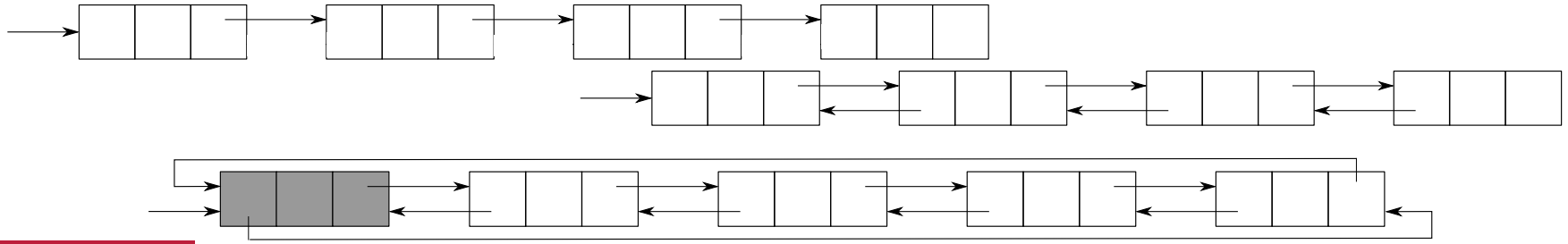
Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

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



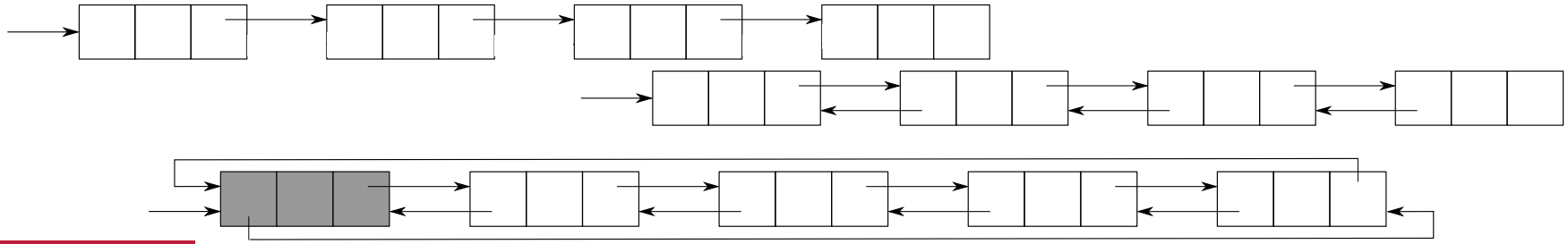
Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

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

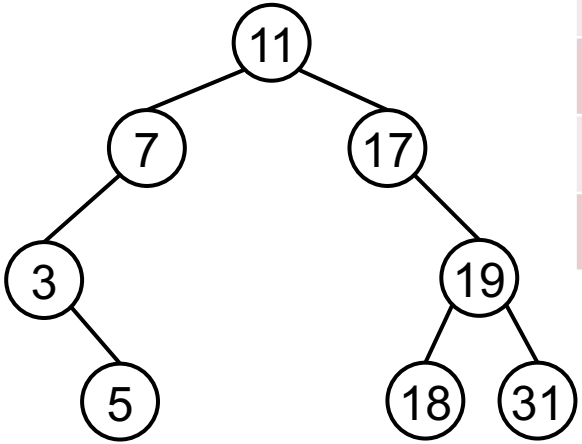


Laufzeiten – dynamische Datenstrukturen (ohne Sortierung)

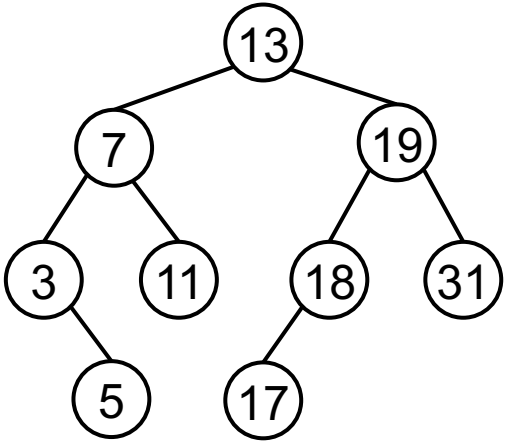
Datentyp	Listen		
Subtyp	Einfach	Doppelt	Zyklisch
Suchen	$O(n)$	$O(n)$	$O(n)$
Einfügen	$O(1)$	$O(1)$	$O(1)$
Löschen	$O(n)$	$O(1)$	$O(1)$
Traversierung	$O(n)$	$O(n)$	$O(n)$



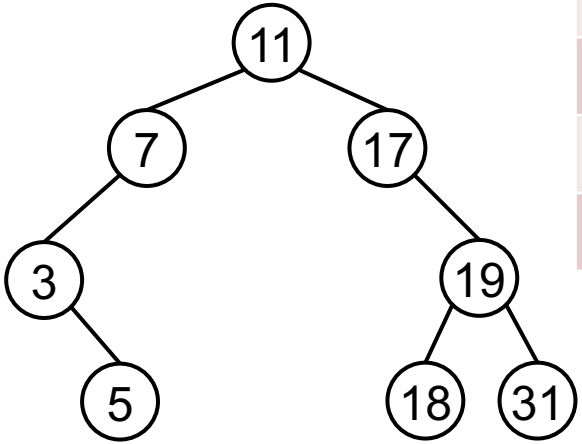
Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



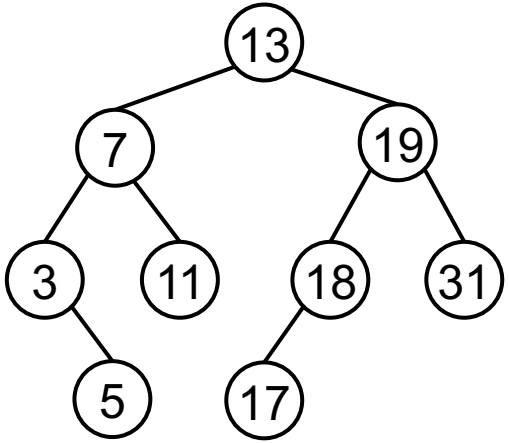
Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen		
Einfügen		
Löschen		
Traversierung		



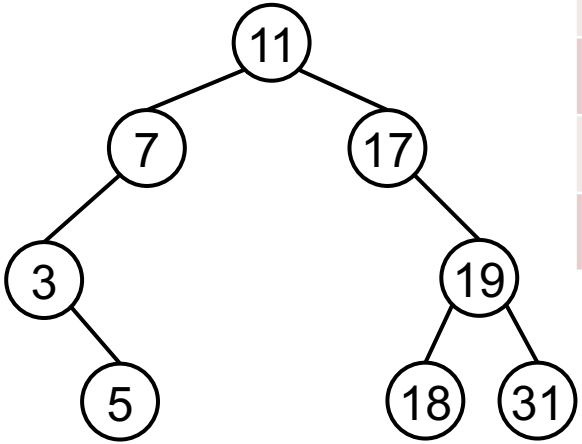
Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



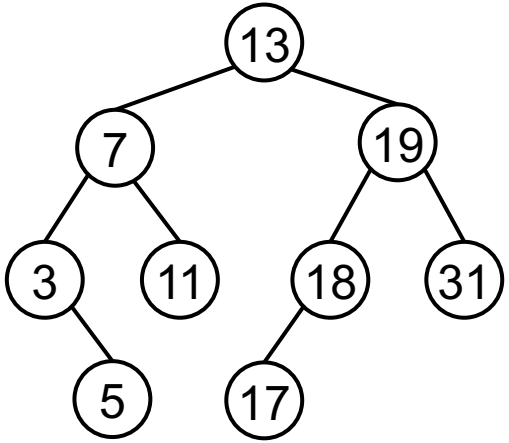
Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen	$O(h)$	
Einfügen		
Löschen		
Traversierung		



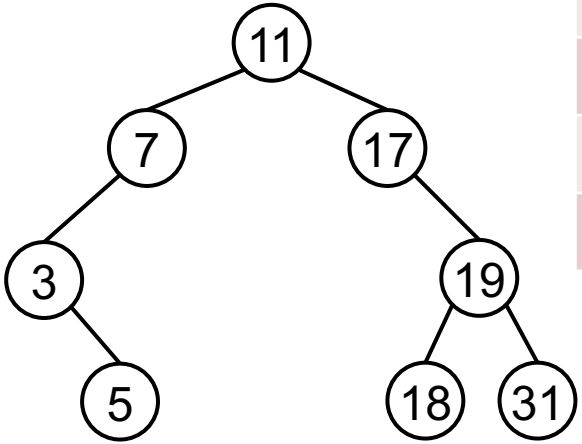
Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



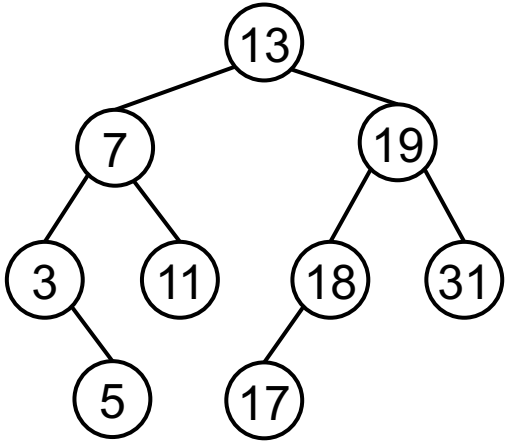
Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen	$O(h)$	
Einfügen	$O(h)$	
Löschen		
Traversierung		



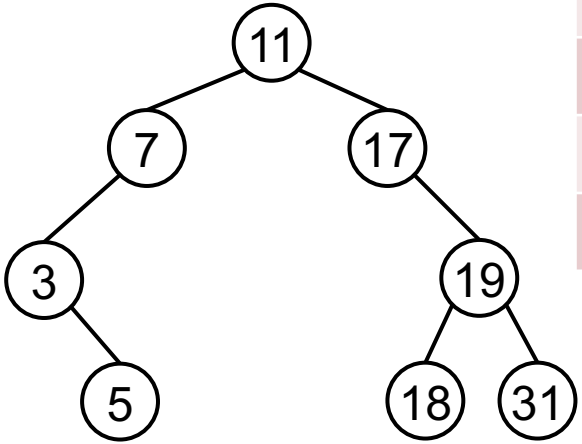
Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



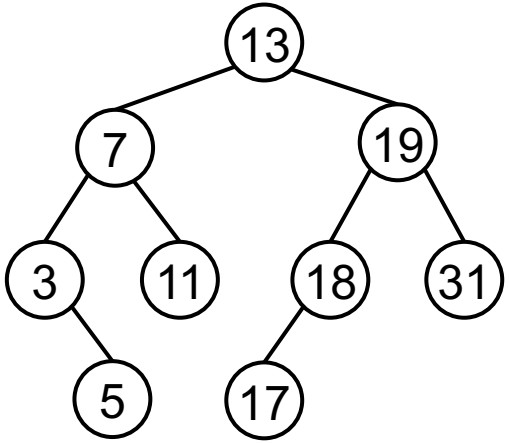
Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen	$O(h)$	
Einfügen	$O(h)$	
Löschen	$O(h)$	
Traversierung		



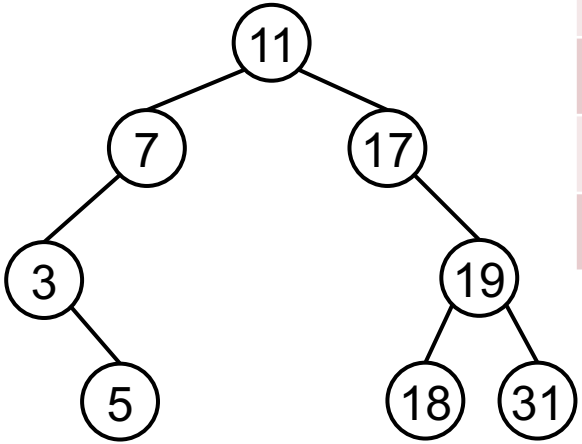
Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



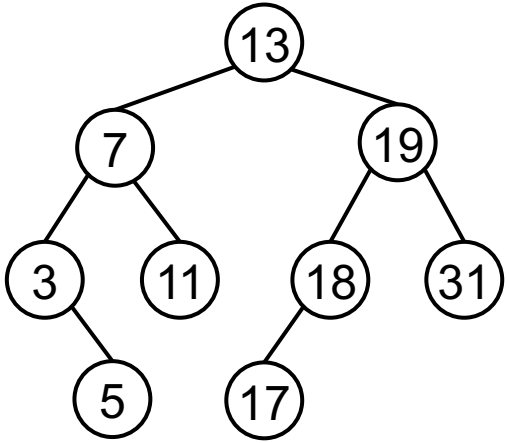
Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen	$O(h)$	
Einfügen	$O(h)$	
Löschen	$O(h)$	
Traversierung	$O(n)$	



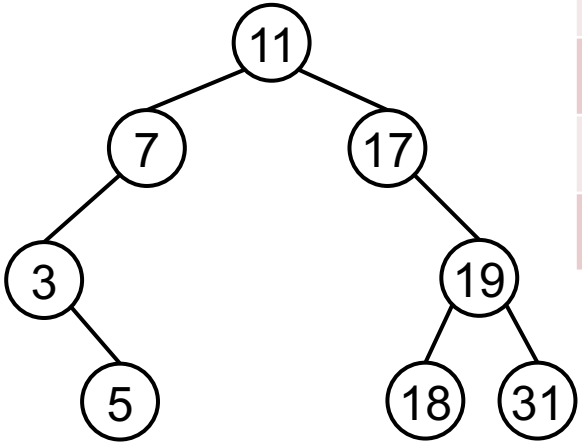
Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



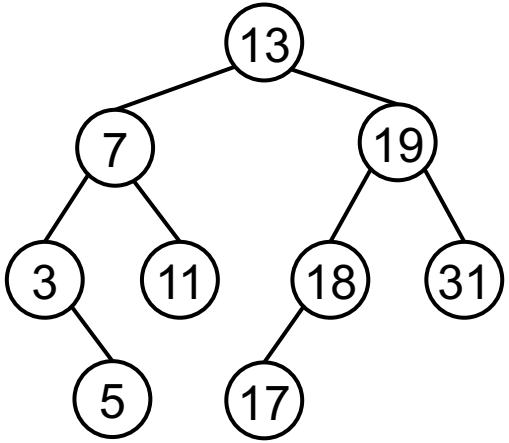
Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen	$O(h)$	$O(\log n)$
Einfügen	$O(h)$	
Löschen	$O(h)$	
Traversierung	$O(n)$	



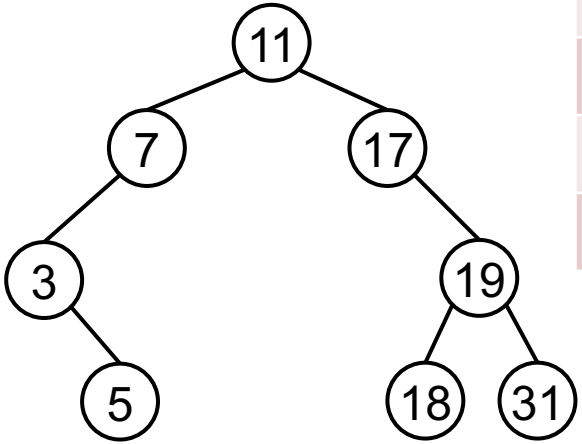
Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



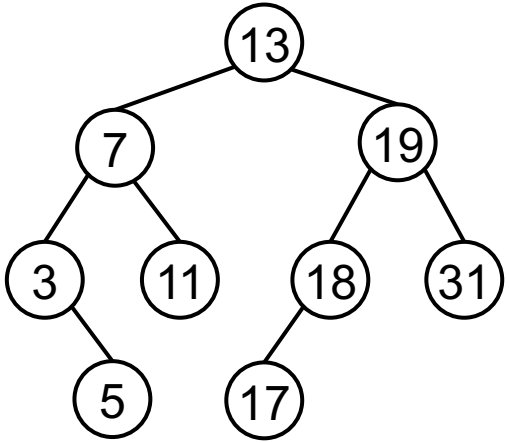
Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen	$O(h)$	$O(\log n)$
Einfügen	$O(h)$	$O(\log n)$
Löschen	$O(h)$	
Traversierung	$O(n)$	



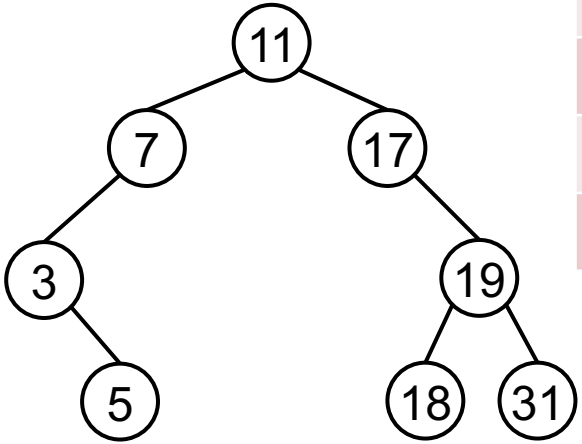
Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



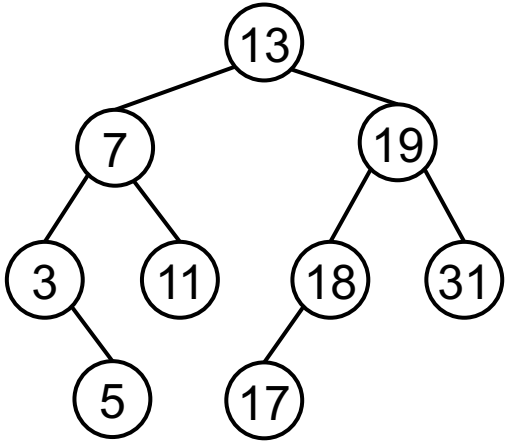
Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen	$O(h)$	$O(\log n)$
Einfügen	$O(h)$	$O(\log n)$
Löschen	$O(h)$	$O(\log n)$
Traversierung	$O(n)$	



Laufzeiten – dynamische Datenstrukturen (mit Sortierung)



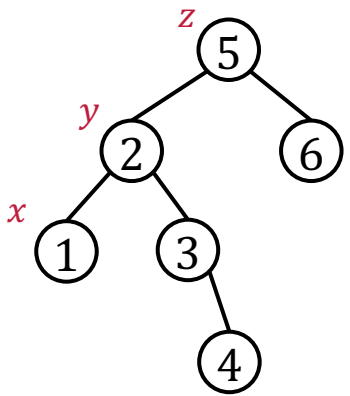
Datentyp	Suchbäume	
Subtyp	-	AVL
Suchen	$O(h)$	$O(\log n)$
Einfügen	$O(h)$	$O(\log n)$
Löschen	$O(h)$	$O(\log n)$
Traversierung	$O(n)$	$O(n)$



AVL-Bäume – Restructure

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?

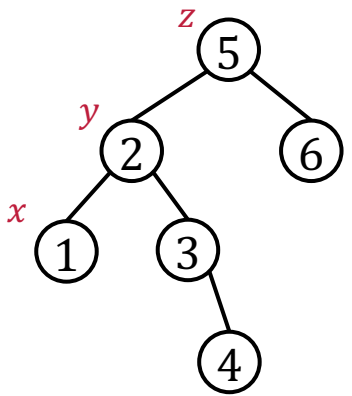


AVL-Bäume – Restructure

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.

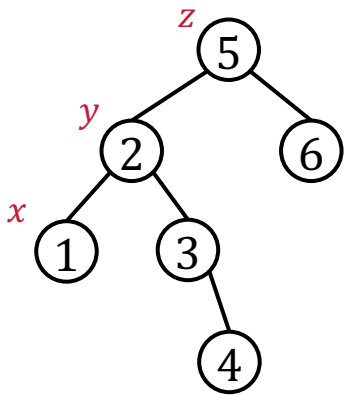


AVL-Bäume – Restructure

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.

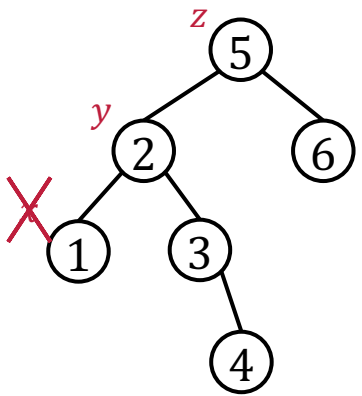


AVL-Bäume – Restructure

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.

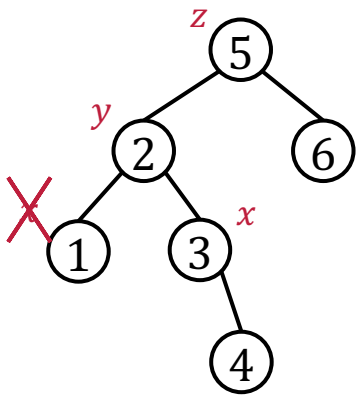


AVL-Bäume – Restructure

Bei Insert und Delete stellen sich nun folgende Fragen:

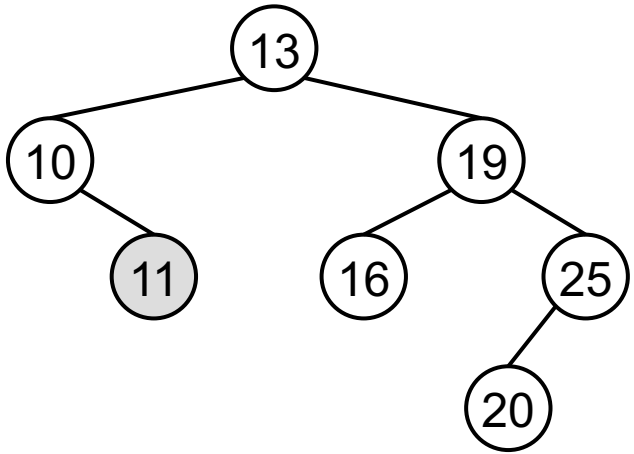
1. Welche Knoten werden unbalanciert?
2. Wie stellt man die Balance wieder her?
3. Welche Regeln sollte man berücksichtigen?

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



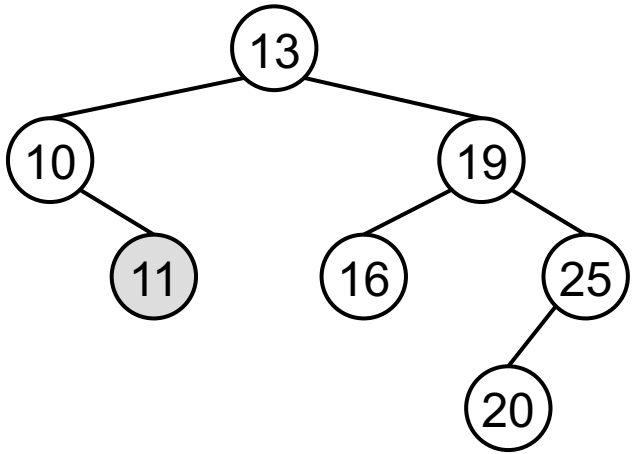
AVL-Bäume – Beispiele

AVL-Bäume – Beispiele



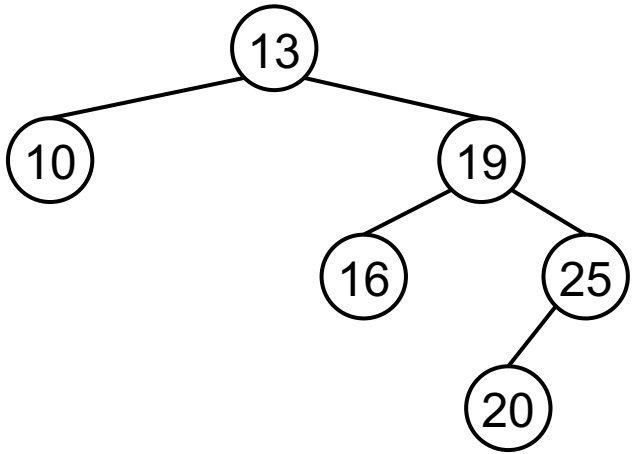
AVL-Bäume – Beispiele

$\text{DELETE}(T, 11)$



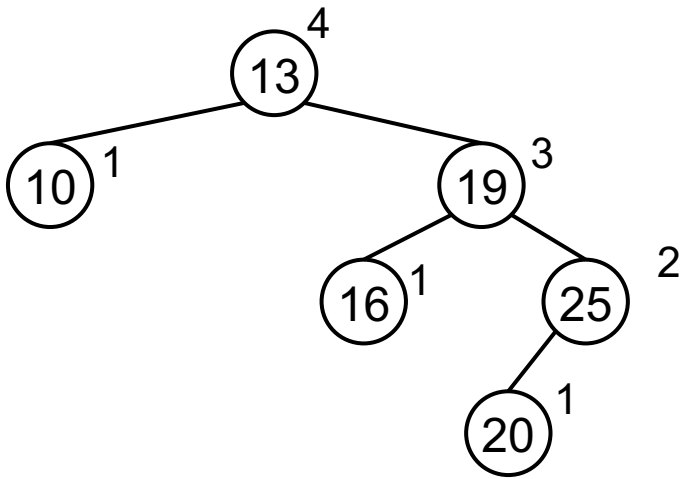
AVL-Bäume – Beispiele

$\text{DELETE}(T, 11)$



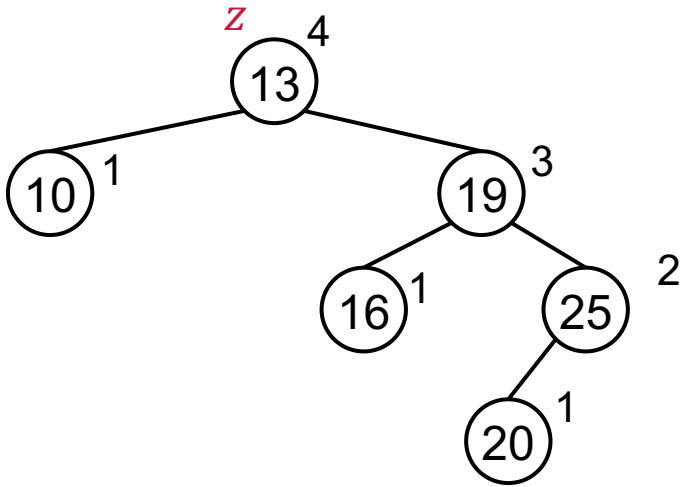
AVL-Bäume – Beispiele

$\text{DELETE}(T, 11)$



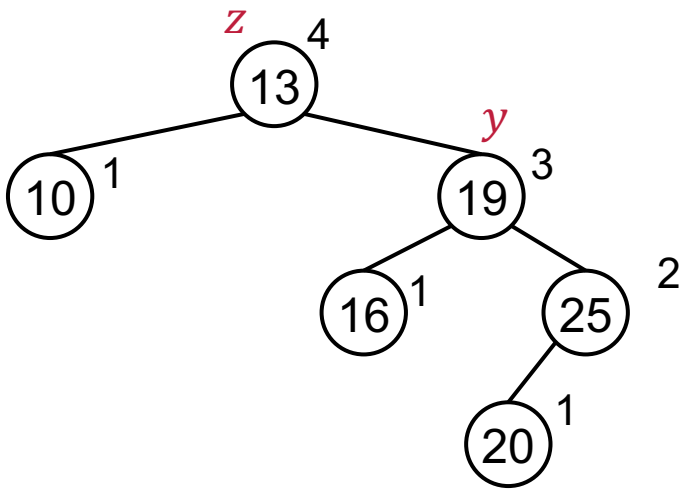
AVL-Bäume – Beispiele

$\text{DELETE}(T, 11)$



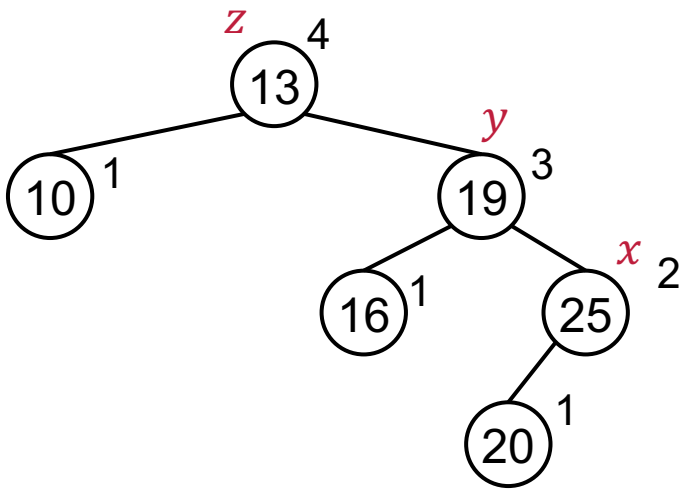
AVL-Bäume – Beispiele

$\text{DELETE}(T, 11)$



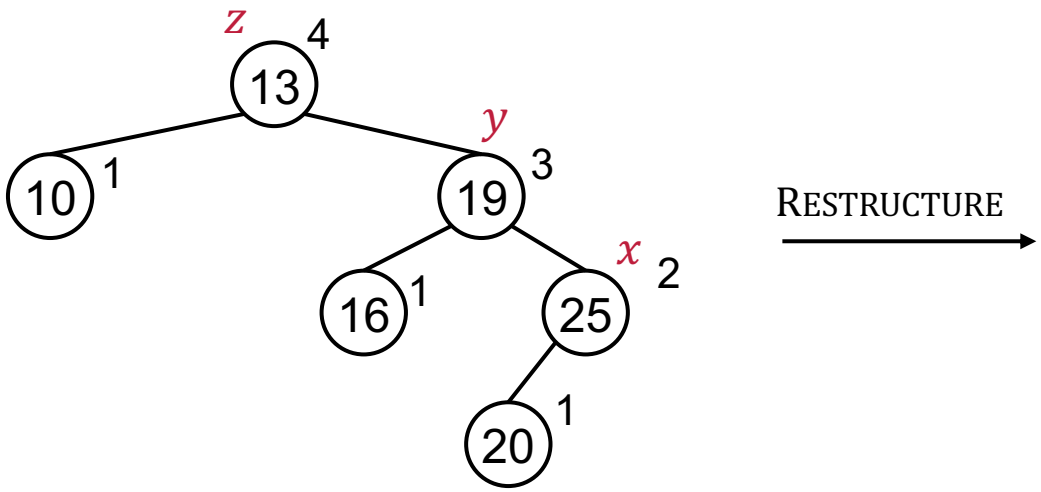
AVL-Bäume – Beispiele

$\text{DELETE}(T, 11)$



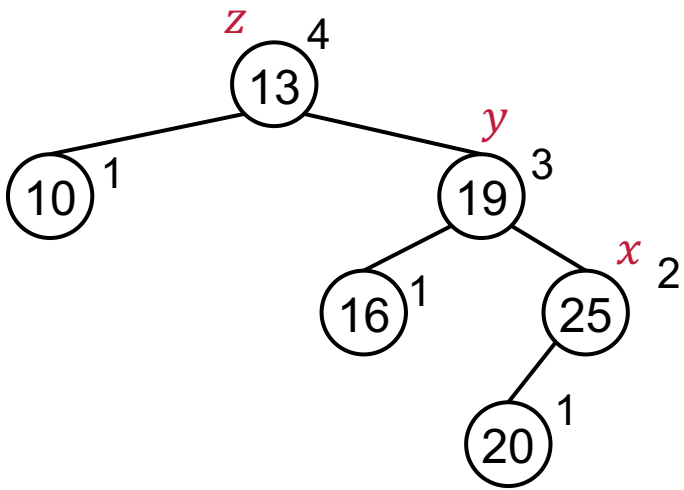
AVL-Bäume – Beispiele

DELETE$(T, 11)$

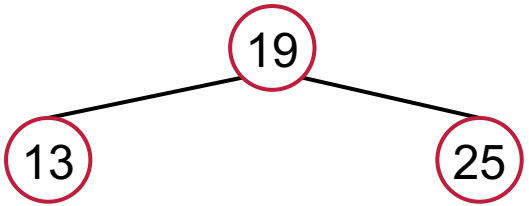


AVL-Bäume – Beispiele

DELETE$(T, 11)$

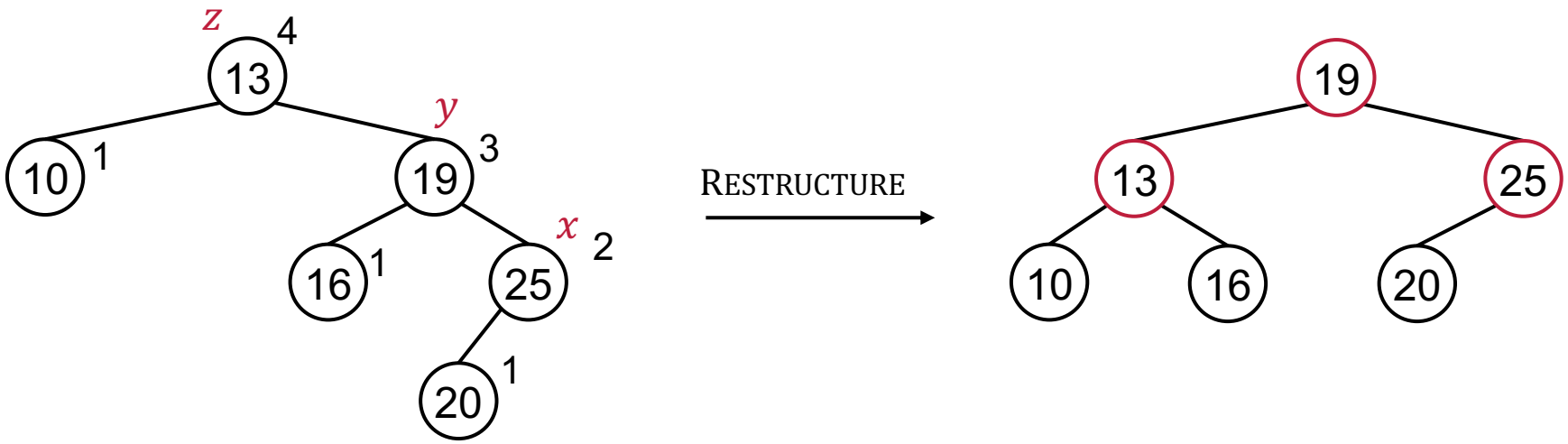


RESTRUCTURE



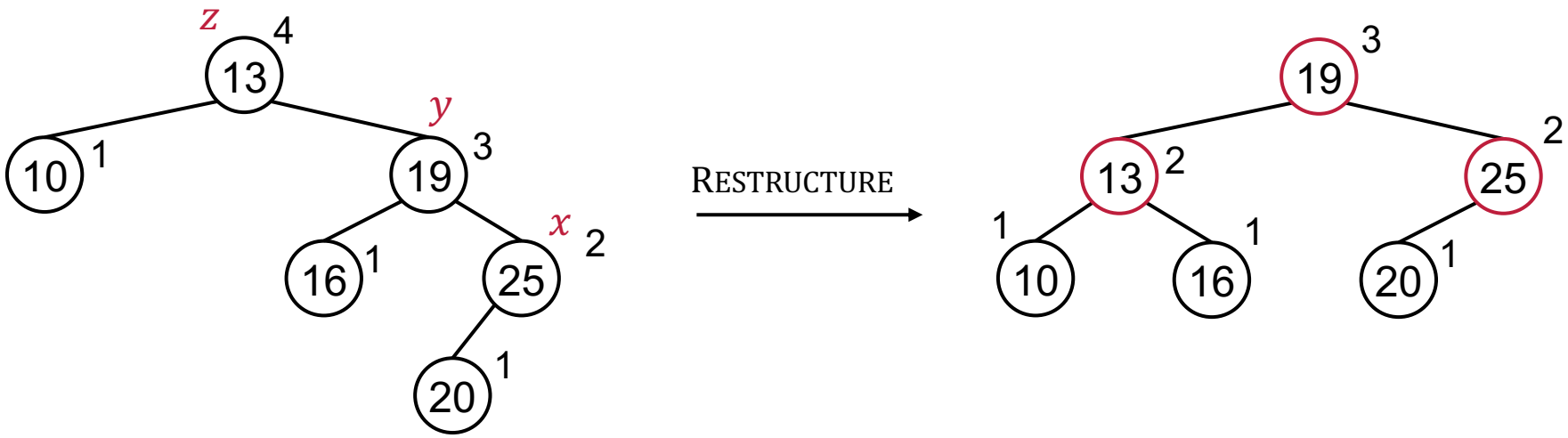
AVL-Bäume – Beispiele

DELETE$(T, 11)$



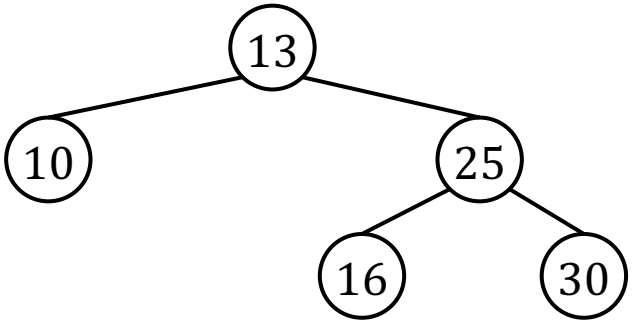
AVL-Bäume – Beispiele

DELETE$(T, 11)$



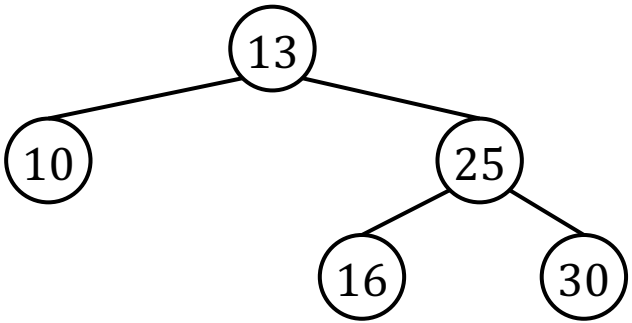
AVL-Bäume – Beispiele

AVL-Bäume – Beispiele



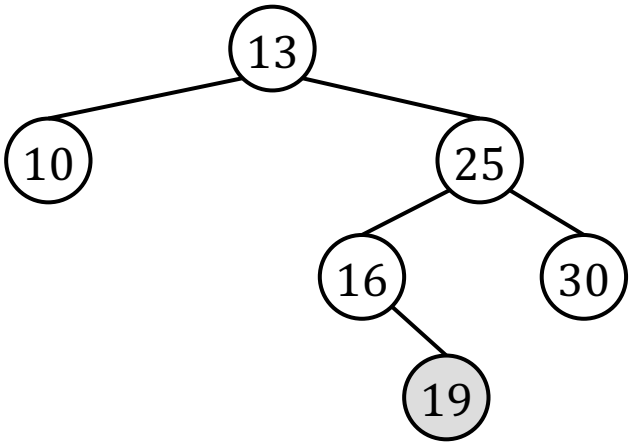
AVL-Bäume – Beispiele

INSERT$(T, 19)$



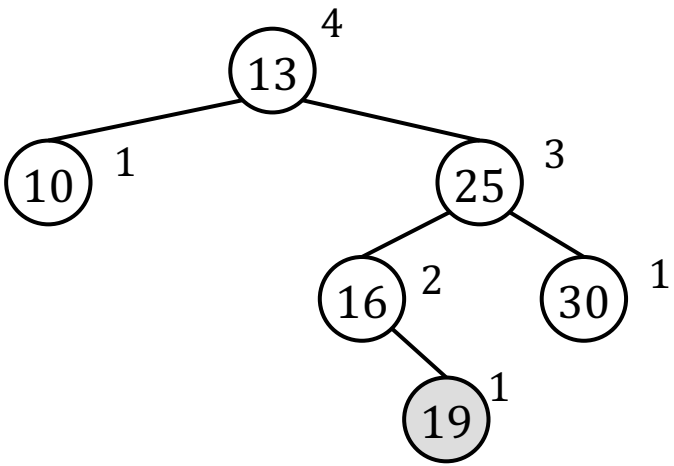
AVL-Bäume – Beispiele

$\text{INSERT}(T, 19)$



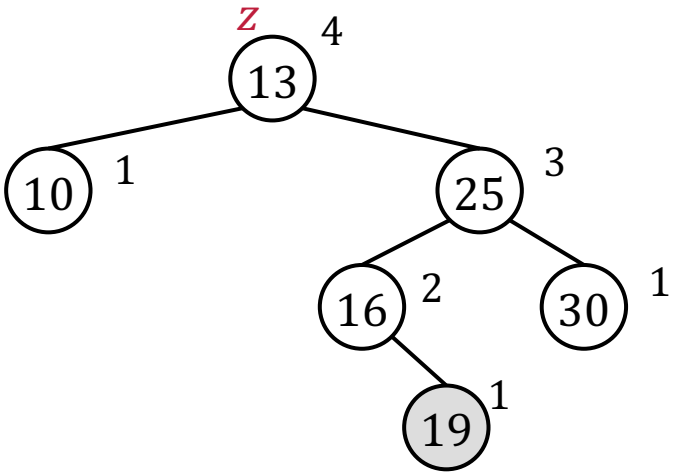
AVL-Bäume – Beispiele

INSERT$(T, 19)$



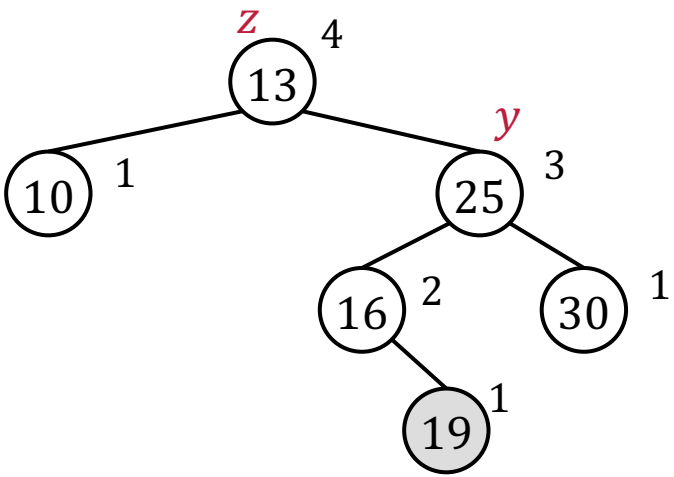
AVL-Bäume – Beispiele

INSERT$(T, 19)$



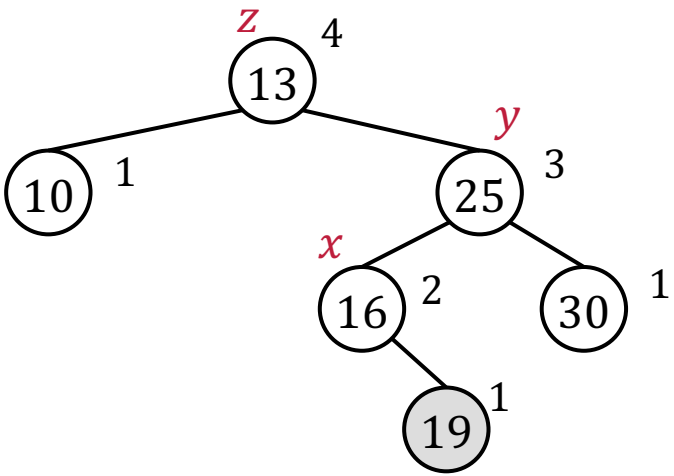
AVL-Bäume – Beispiele

INSERT$(T, 19)$



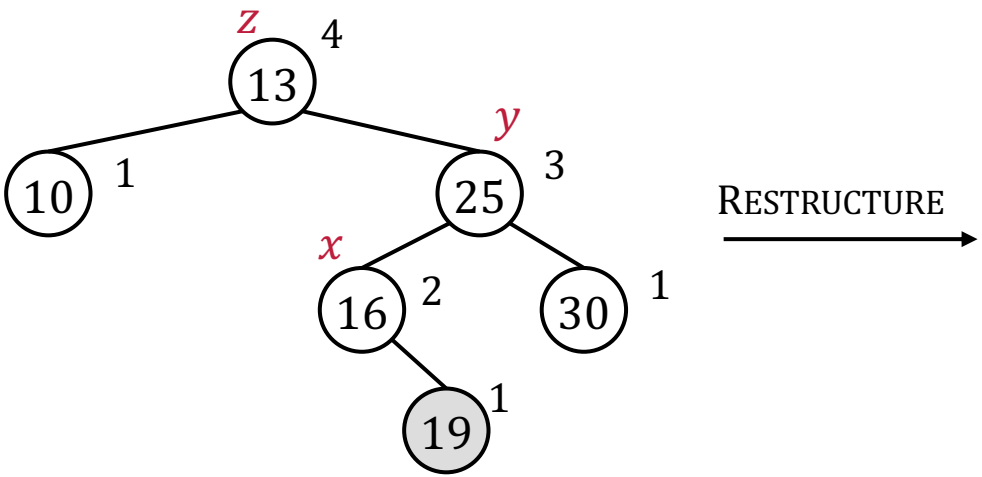
AVL-Bäume – Beispiele

INSERT$(T, 19)$



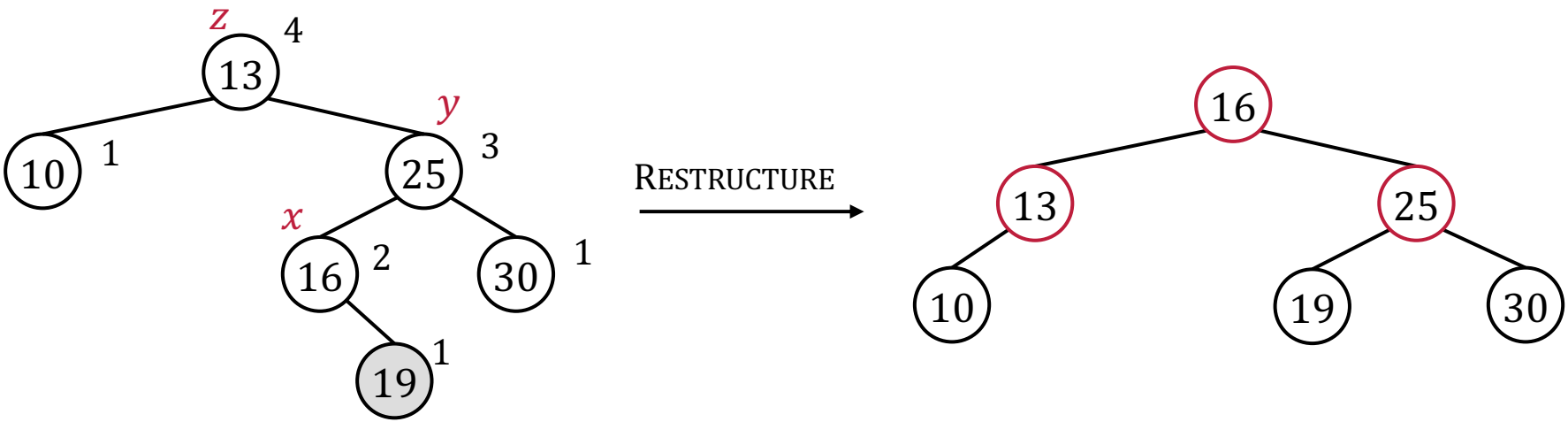
AVL-Bäume – Beispiele

INSERT$(T, 19)$



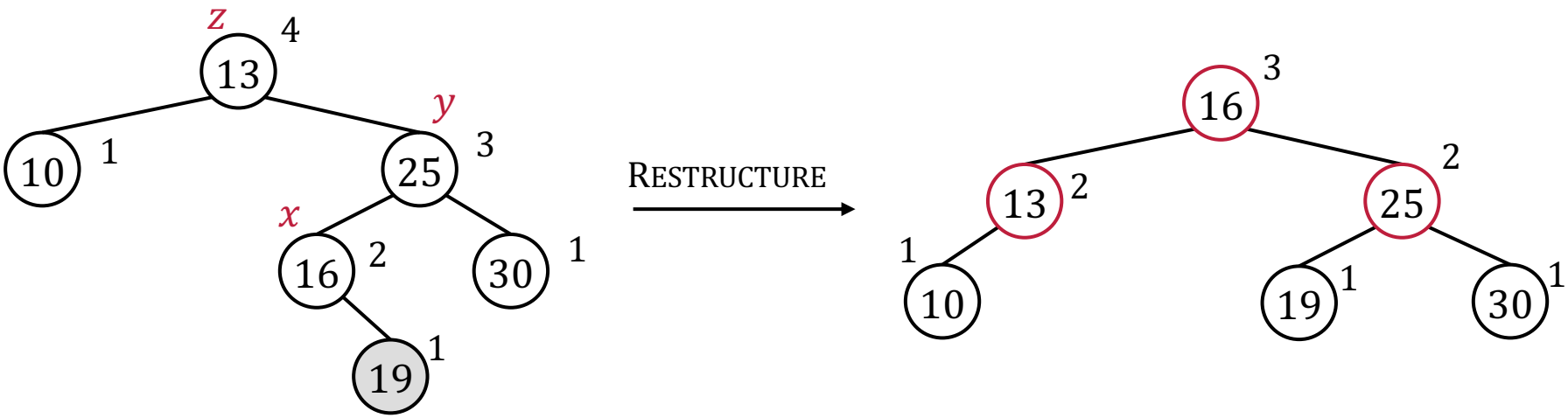
AVL-Bäume – Beispiele

INSERT$(T, 19)$



AVL-Bäume – Beispiele

INSERT$(T, 19)$



AVL Restructure - Zusammenfassung

Merkblatt!

AVL Restructure - Zusammenfassung

Merkblatt!

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

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.

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.

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.

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:

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.

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.

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.

AVL Restructure - Zusammenfassung

Merkblatt!

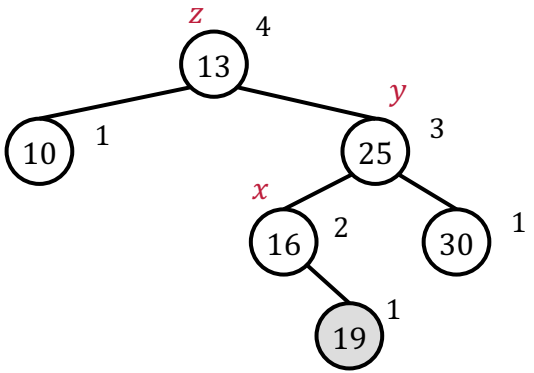
Ist nach INSERT oder DELETE die AVL-Eigenschaft verletzt, kann diese mit RESTRUCTURE wiederhergestellt werden.

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

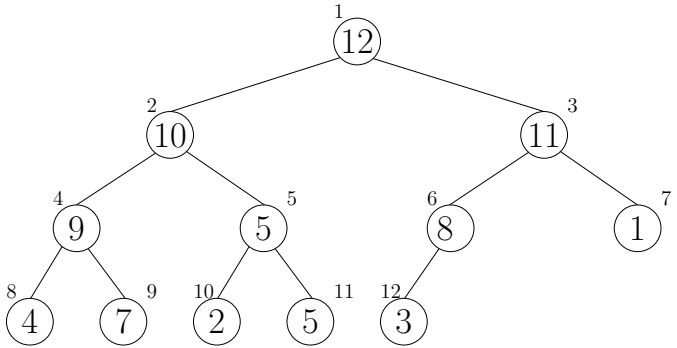
Nach DELETE können *mehrere* (genauer gesagt $O(\log n)$ viele) Aufrufe von RESTRUCTURE nötig sein.

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.



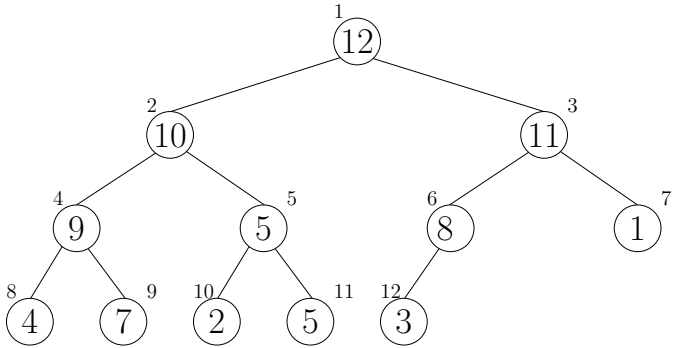
Laufzeiten – dynamische Datenstrukturen (partielle Sortierung)



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

Laufzeiten – dynamische Datenstrukturen (partielle Sortierung)

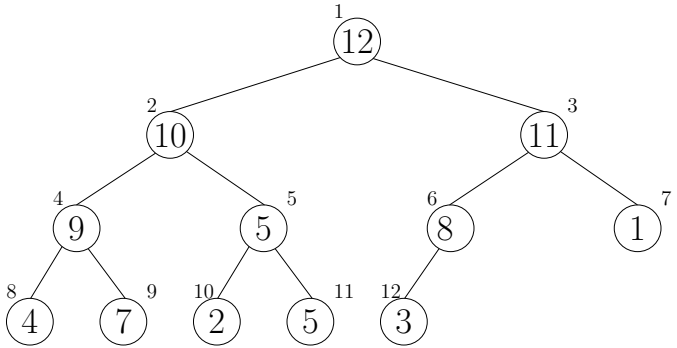
Datentyp	(Max)Heaps
Einfügen	
Löschen	
Minimum/Maximum	
Extrahiere Min/Max	



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

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

Algorithm 1: Bubblesort(A, n)

```
begin
    for j := n − 1 DOWNTO 1 do
        for i := 1 TO j do
            if A[i] < A[i + 1] then
                vertausche (A[i], A[i + 1])
```

Klausur WiSe 11/12

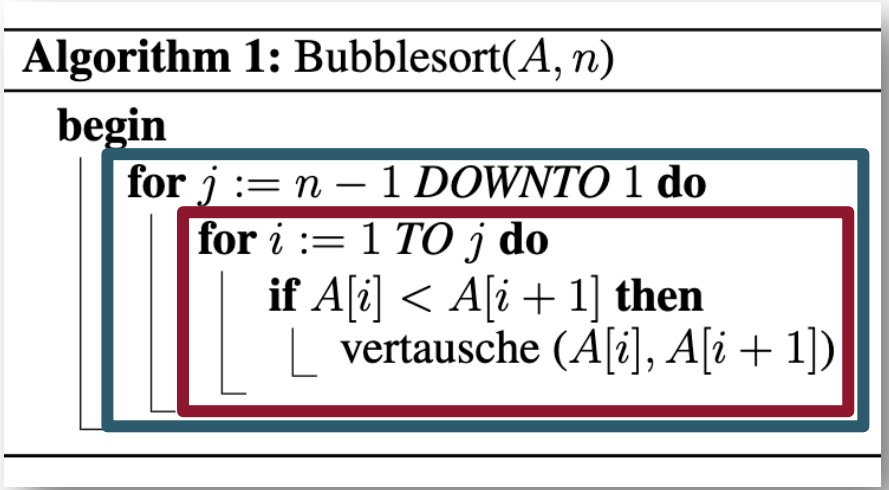
Sortieren mit Bubblesort

Algorithm 1: Bubblesort(A, n)

```
begin
    for j := n − 1 DOWNTO 1 do
        for i := 1 TO j do
            if A[i] < A[i + 1] then
                vertausche (A[i], A[i + 1])
```

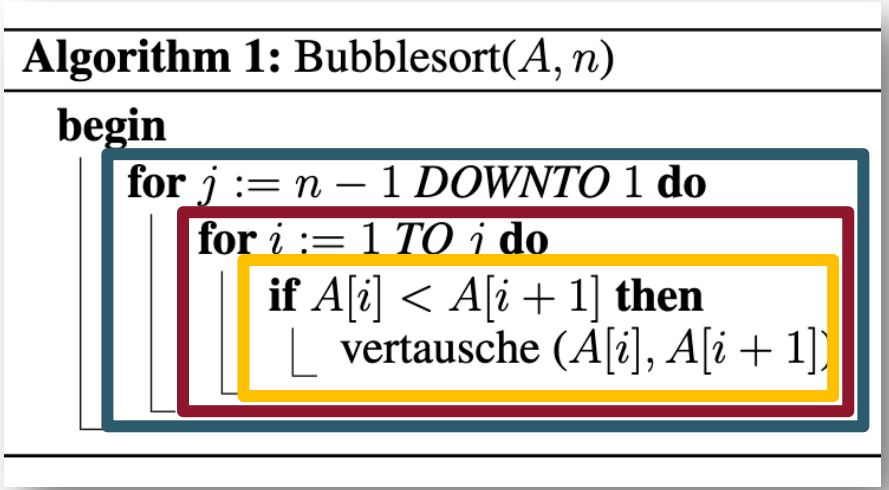
Klausur WiSe 11/12

Sortieren mit Bubblesort



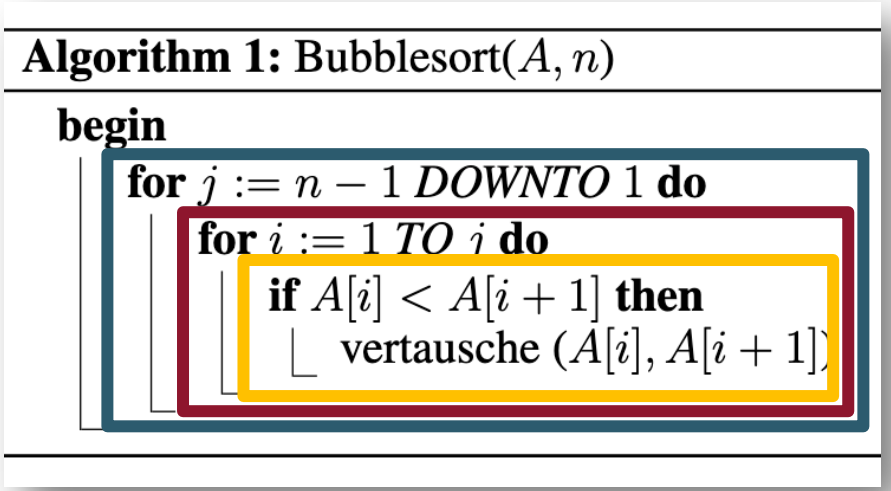
Klausur WiSe 11/12

Sortieren mit Bubblesort



Klausur WiSe 11/12

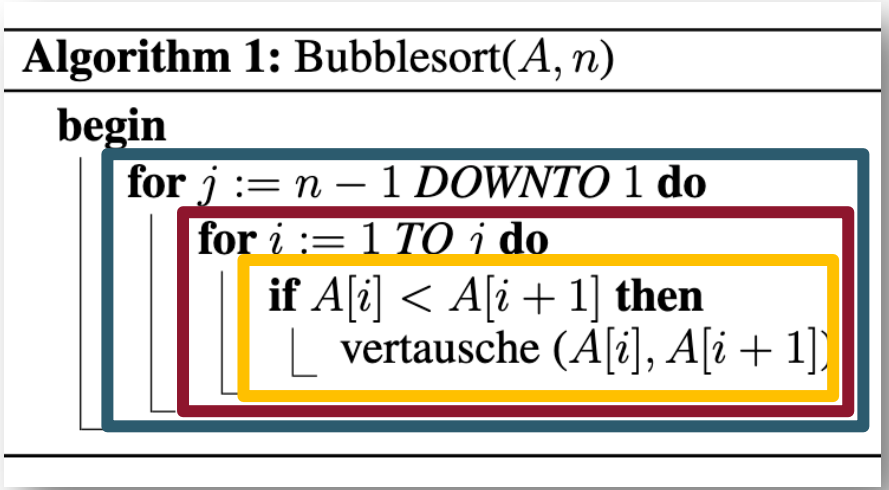
Sortieren mit Bubblesort



Laufzeit:

Klausur WiSe 11/12

Sortieren mit Bubblesort

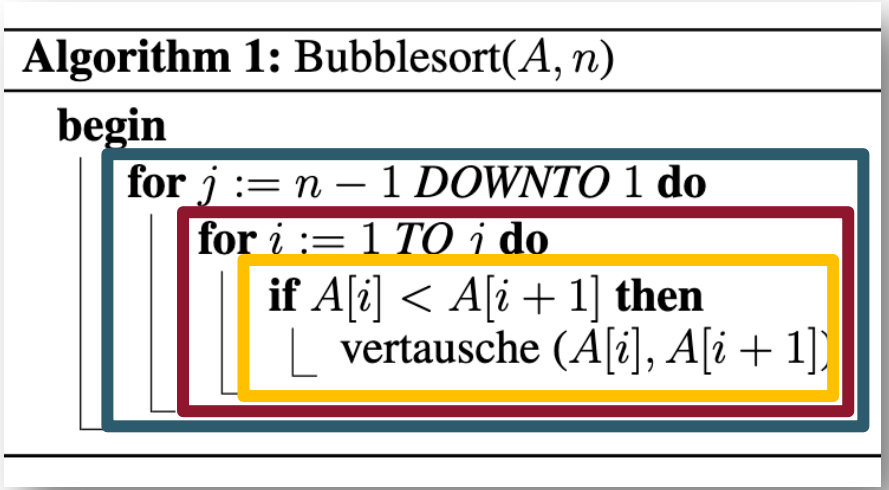


Klausur WiSe 11/12

Laufzeit:

$$\#Iterationen * \#Iterationen * Box$$

Sortieren mit Bubblesort

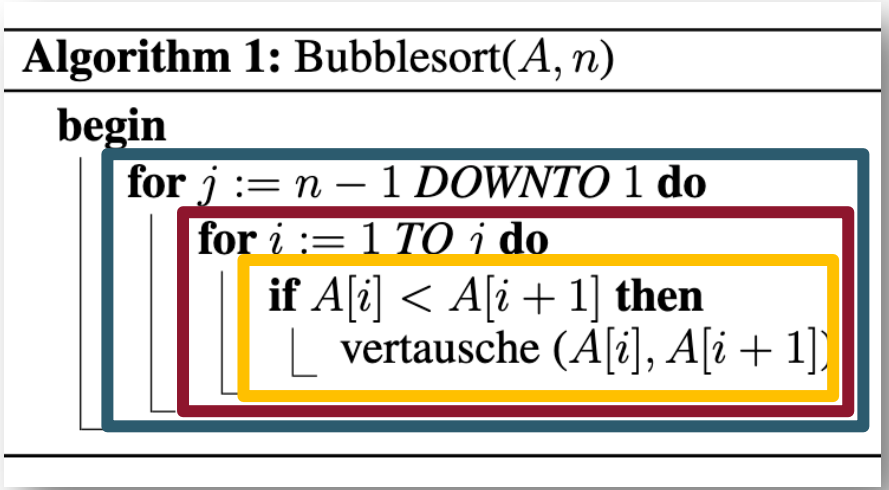


Klausur WiSe 11/12

Laufzeit:

$$\#Iterationen * \#Iterationen * Box$$
$$\#Iterationen * \#Iterationen * O(1)$$

Sortieren mit Bubblesort

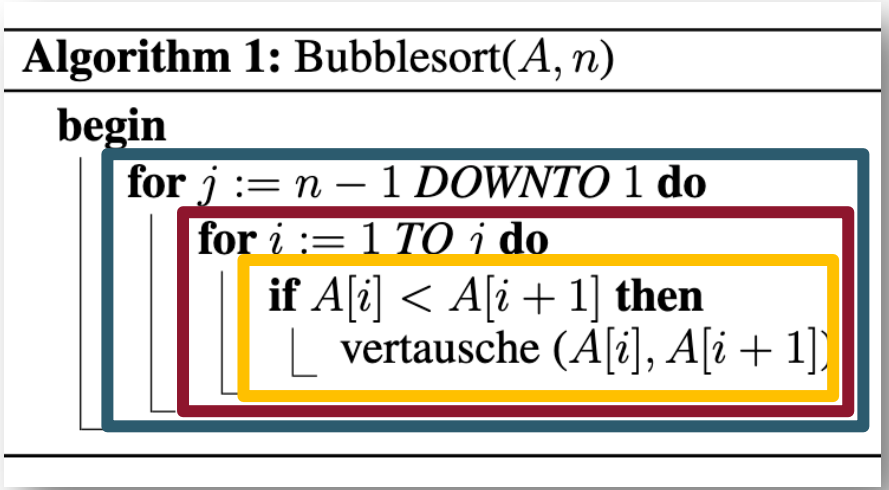


Klausur WiSe 11/12

Laufzeit:

$$\#Iterationen * \#Iterationen * Box$$
$$\#Iterationen * \#Iterationen * O(1)$$
$$\#Iterationen * O(n) * O(1)$$

Sortieren mit Bubblesort

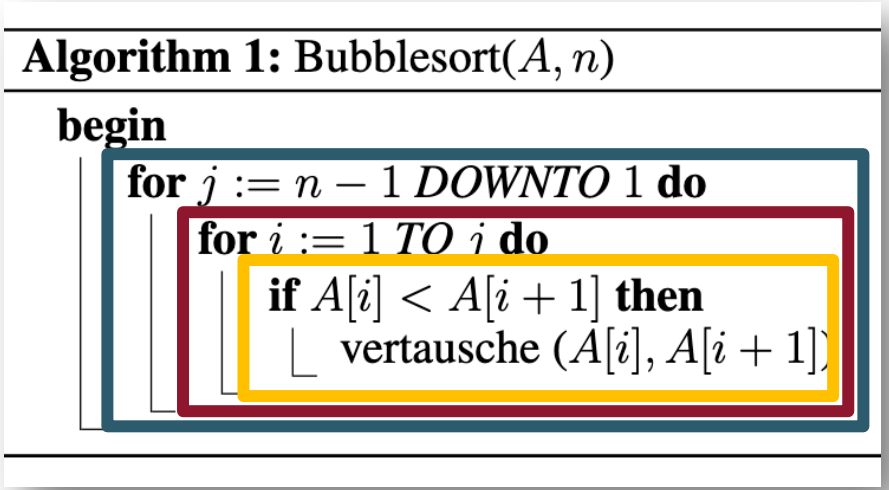


Klausur WiSe 11/12

Laufzeit:

$$\#Iterationen * \#Iterationen * Box$$
$$\#Iterationen * \#Iterationen * O(1)$$
$$\#Iterationen * O(n) * O(1)$$
$$O(n) * O(n) * O(1)$$

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

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:

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

$$\Leftrightarrow$$

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

Ω-Notation

Definition:

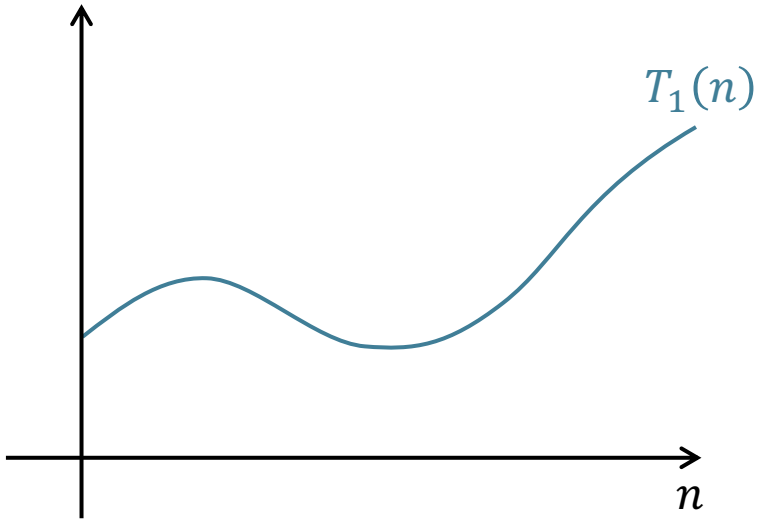
Es gibt Konstanten $n_0 \in \mathbb{N}$ und $c_2 \in \mathbb{R}^+$,

sodass für alle $n \geq n_0$ gilt:

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

$$\Leftrightarrow$$

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



Ω-Notation

Definition:

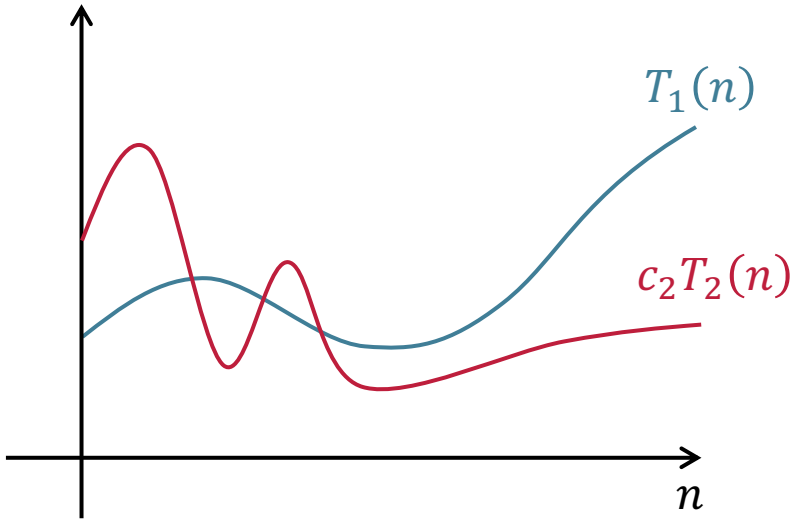
Es gibt Konstanten $n_0 \in \mathbb{N}$ und $c_2 \in \mathbb{R}^+$,

sodass für alle $n \geq n_0$ gilt:

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

$$\Leftrightarrow$$

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



Ω-Notation

Definition:

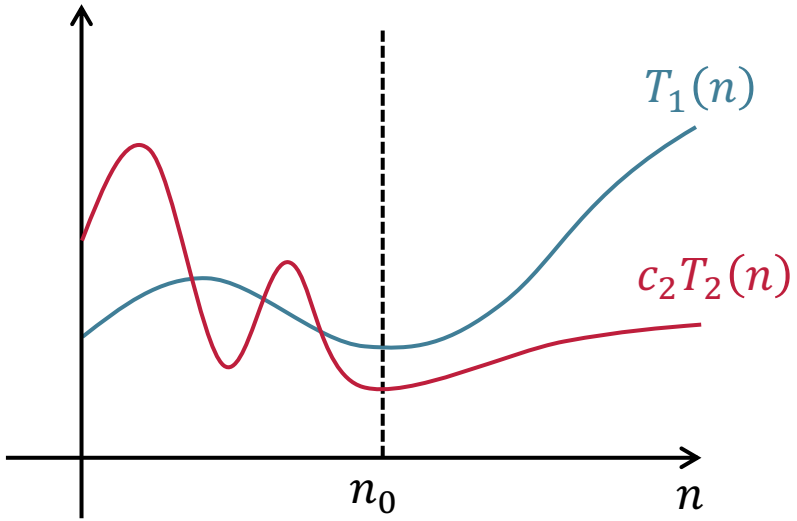
Es gibt Konstanten $n_0 \in \mathbb{N}$ und $c_2 \in \mathbb{R}^+$,

sodass für alle $n \geq n_0$ gilt:

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

$$\Leftrightarrow$$

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



Ω-Notation

Definition:

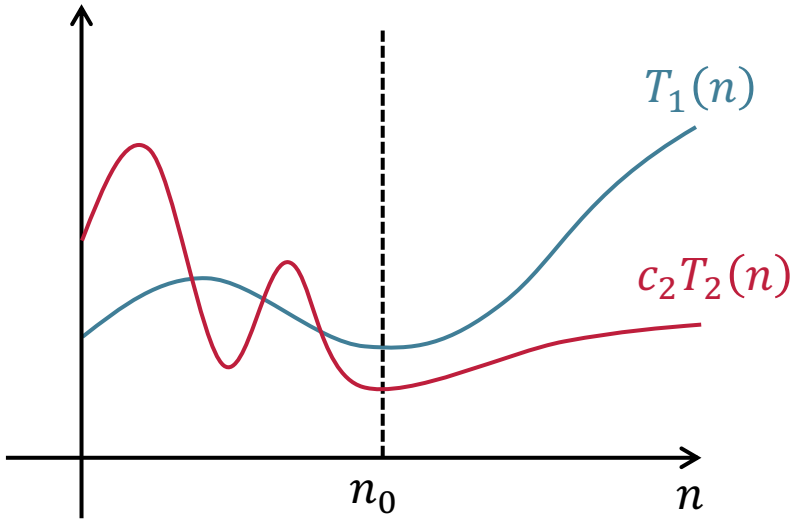
Es gibt Konstanten $n_0 \in \mathbb{N}$ und $c_2 \in \mathbb{R}^+$,

sodass für alle $n \geq n_0$ gilt:

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

$$\Leftrightarrow$$

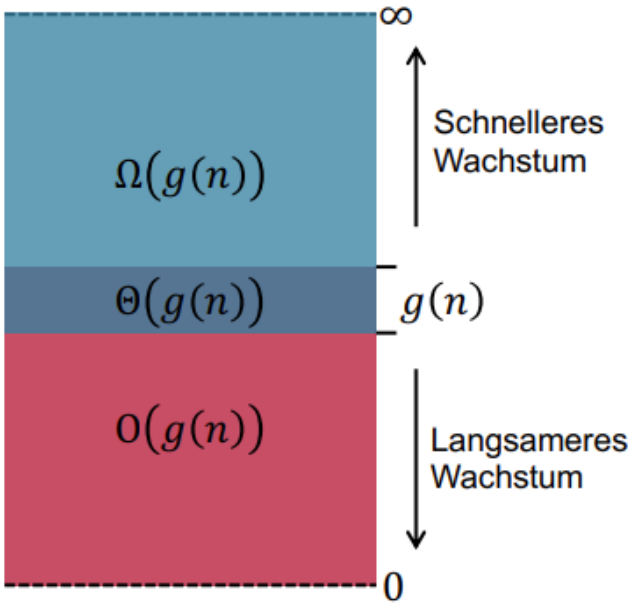
$$T_1(n) \in \Omega\big(T_2(n)\big)$$



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

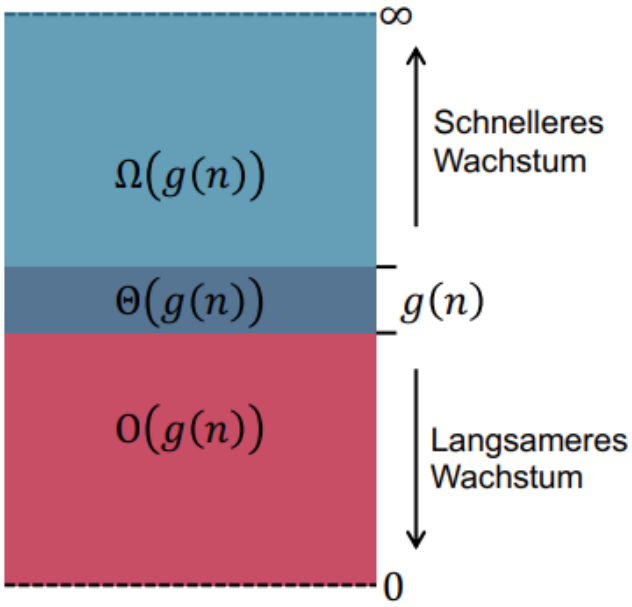
Wachstum von Funktionen

Vergleichen von Klassen



Wachstum von Funktionen

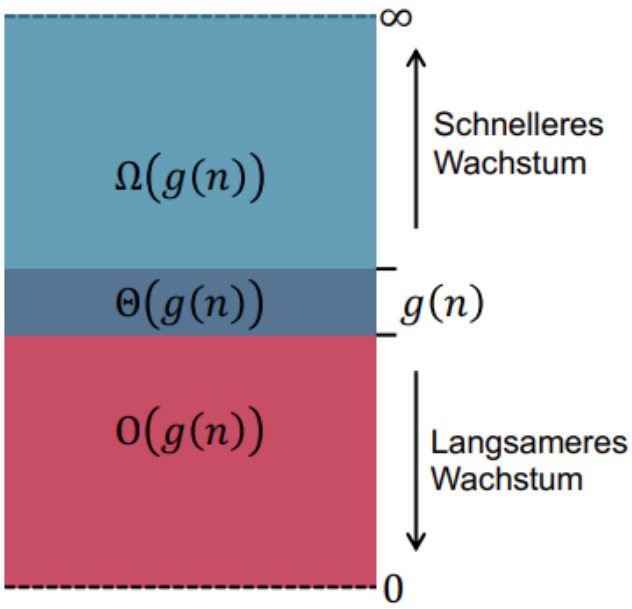
Vergleichen von Klassen



Hierarchie-Ausschnitt:

Wachstum von Funktionen

Vergleichen von Klassen

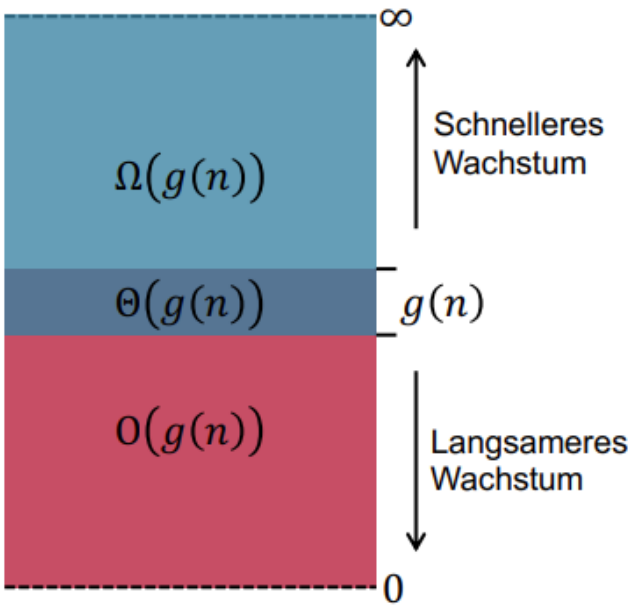


Hierarchie-Ausschnitt:

$O(1)$

Wachstum von Funktionen

Vergleichen von Klassen

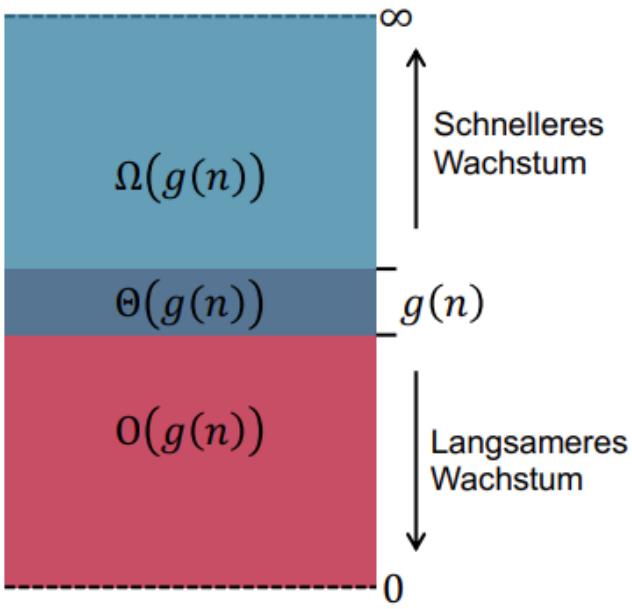


Hierarchie-Ausschnitt:

$O(1) \subset O(\log^a n)$

Wachstum von Funktionen

Vergleichen von Klassen

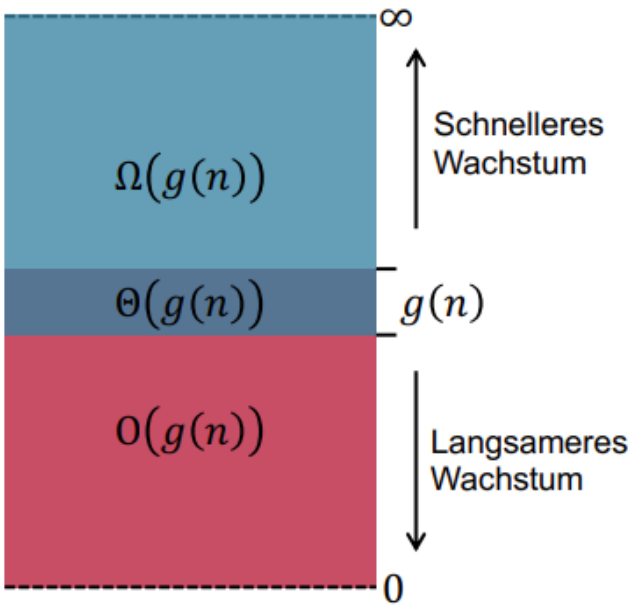


Hierarchie-Ausschnitt:

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

Wachstum von Funktionen

Vergleichen von Klassen

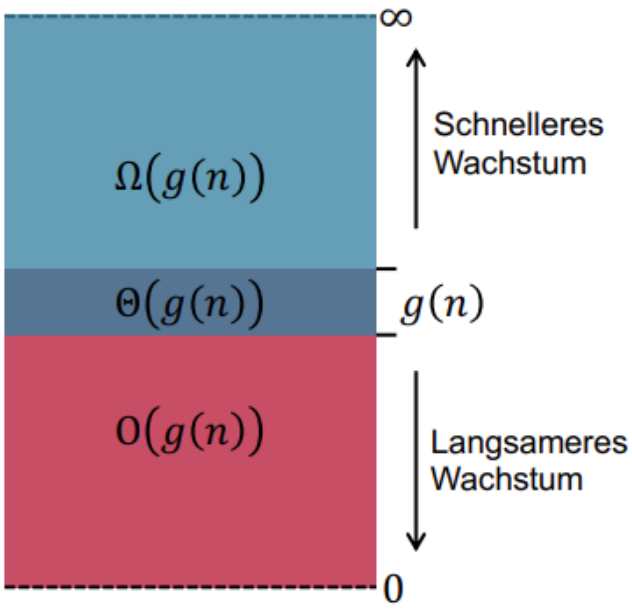


Hierarchie-Ausschnitt:

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

Wachstum von Funktionen

Vergleichen von Klassen

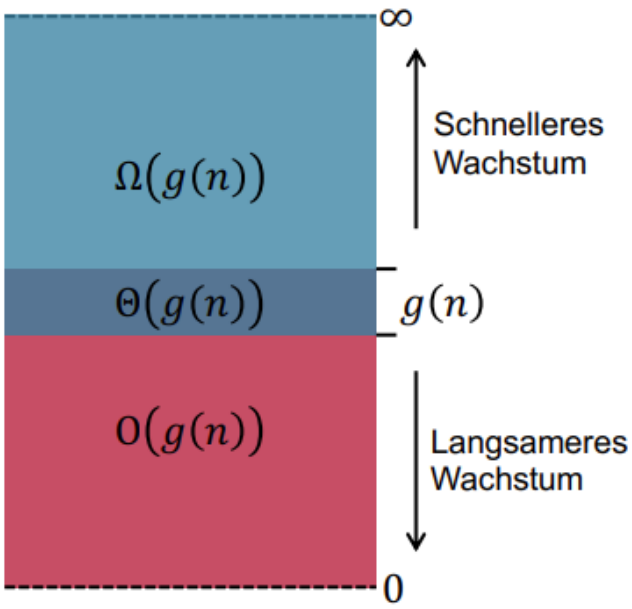


Hierarchie-Ausschnitt:

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

Wachstum von Funktionen

Vergleichen von Klassen

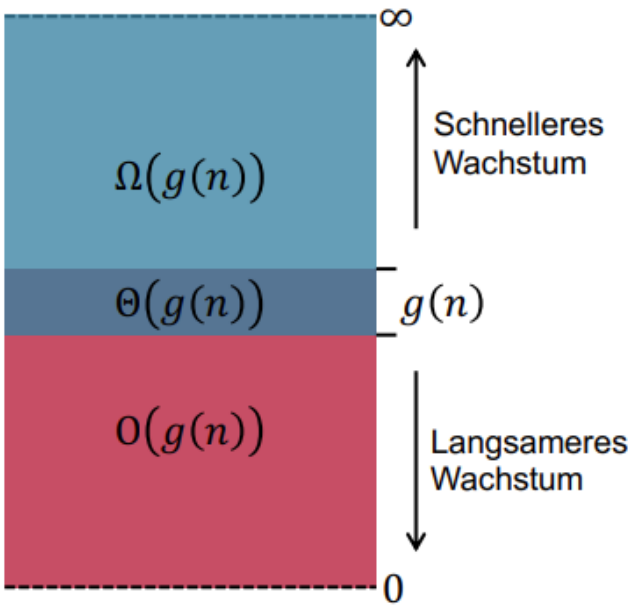


Hierarchie-Ausschnitt:

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

Wachstum von Funktionen

Vergleichen von Klassen



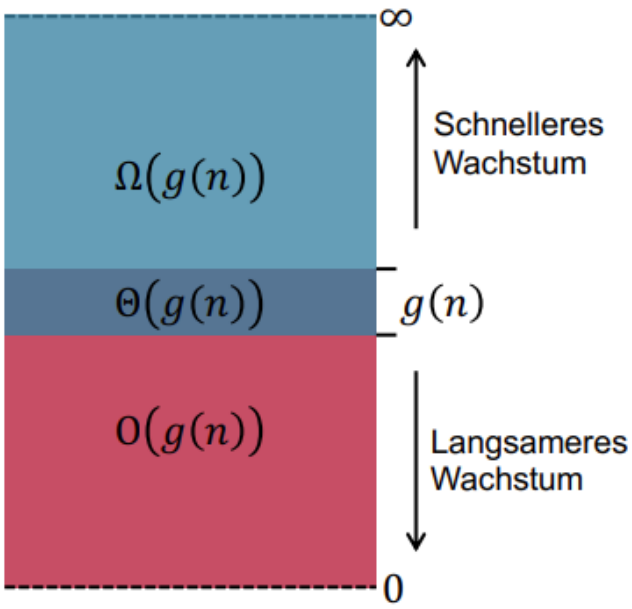
Hierarchie-Ausschnitt:

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

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

Wachstum von Funktionen

Vergleichen von Klassen



Hierarchie-Ausschnitt:

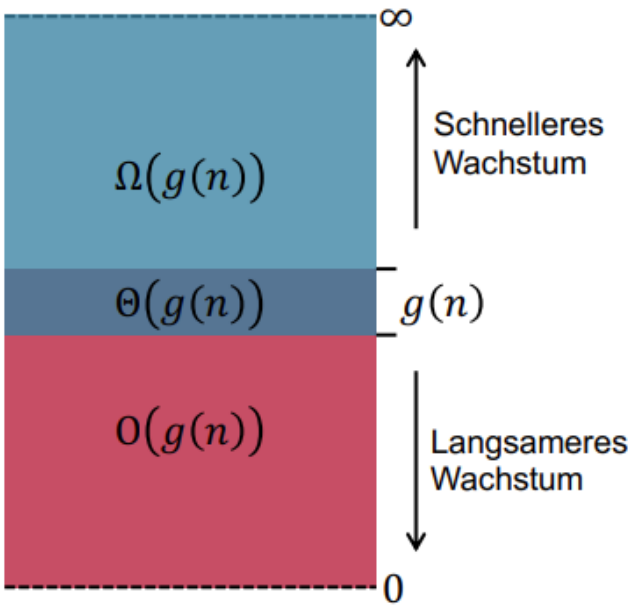
$$O(1) \subset O(\log^a n) \subset O(n^b) \subset O(c^n) \subset O(n!) \subset O(n^n)$$

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

Bei Ω dreht sich das Inklusionszeichen um!

Wachstum von Funktionen

Vergleichen von Klassen



Hierarchie-Ausschnitt:

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

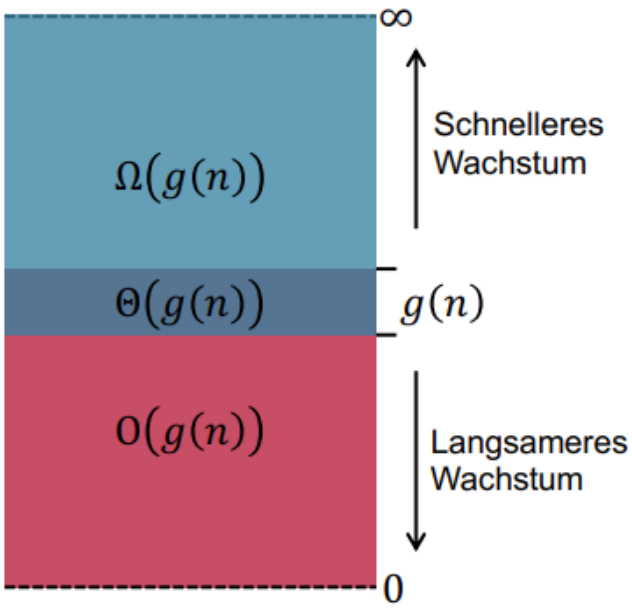
$$a, b > 0, c > 1$$

Bei Ω dreht sich das Inklusionszeichen um!

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

Wachstum von Funktionen

Vergleichen von Klassen



Hierarchie-Ausschnitt:

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

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

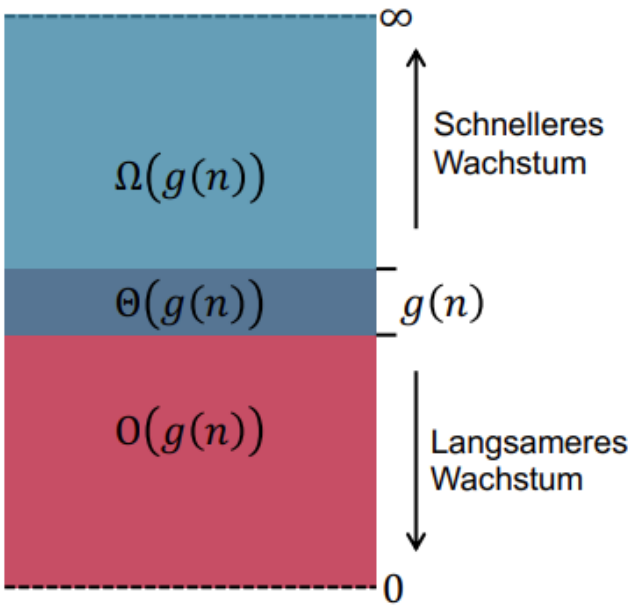
Bei Ω dreht sich das Inklusionszeichen um!

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

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

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?

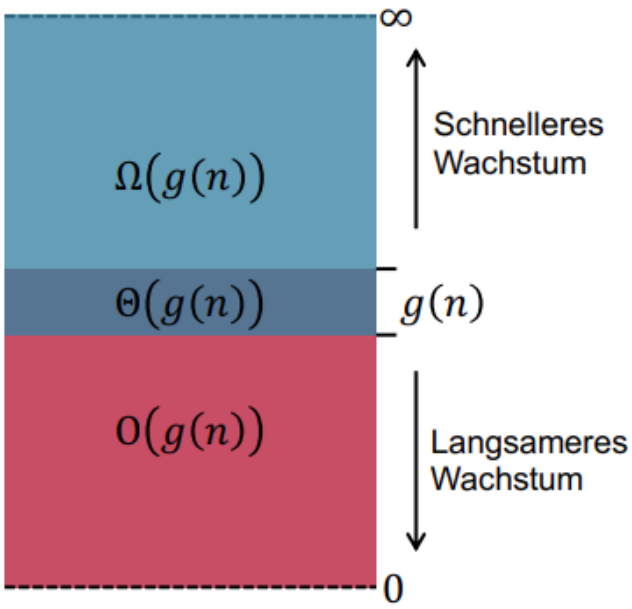
Wie steht das zu $O(n \log \log n)$?

Wir wissen:

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

Wachstum von Funktionen

Vergleichen von Klassen



Hierarchie-Ausschnitt:

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

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

Bei Ω dreht sich das Inklusionszeichen um!

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

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

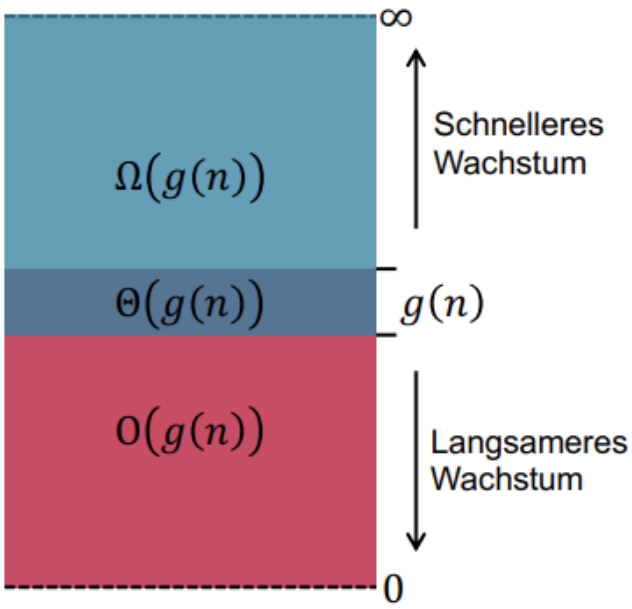
Wir wissen:

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

Also muss gelten:

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

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

Beweis (Ω-Notation)

Wachstum von Funktionen

Bestimmen von Klassen

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

Beweis (Ω-Notation)

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

Wachstum von Funktionen

Bestimmen von Klassen

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

Beweis (Ω-Notation)

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

Wachstum von Funktionen

Bestimmen von Klassen

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

Beweis (Ω-Notation)

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

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

Wachstum von Funktionen

Bestimmen von Klassen

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

Beweis (Ω-Notation)

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

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

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

Wachstum von Funktionen

Bestimmen von Klassen

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

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! →

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

Wachstum von Funktionen

Bestimmen von Klassen

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

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! →

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

Wachstum von Funktionen

Bestimmen von Klassen

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

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! →

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

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

Wachstum von Funktionen

Bestimmen von Klassen

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

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! →

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

Wachstum von Funktionen

Bestimmen von Klassen

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

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

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“:

Tipps für Wachstumsanalyse

Mit „Abuse of Notation“:

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

Tipps für Wachstumsanalyse

Mit „Abuse of Notation“:

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

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

Tipps für Wachstumsanalyse

Mit „Abuse of Notation“:

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

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

Tipps für Wachstumsanalyse

Mit „Abuse of Notation“:

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

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

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

Tipps für Wachstumsanalyse

Mit „Abuse of Notation“:

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

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

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

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

Tipps für Wachstumsanalyse

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

Tipps für Wachstumsanalyse

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

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

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:

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:

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

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:

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

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

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:

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

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

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

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

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

Wachstum von Funktionen (Beweise)

Satz 3.12

Satz 3.12

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

Satz 3.12

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

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

Satz 3.12

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

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

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

Satz 3.12

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

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

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

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

Satz 3.12

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

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

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

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

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

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 $c' = \frac{1}{c}$)

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!

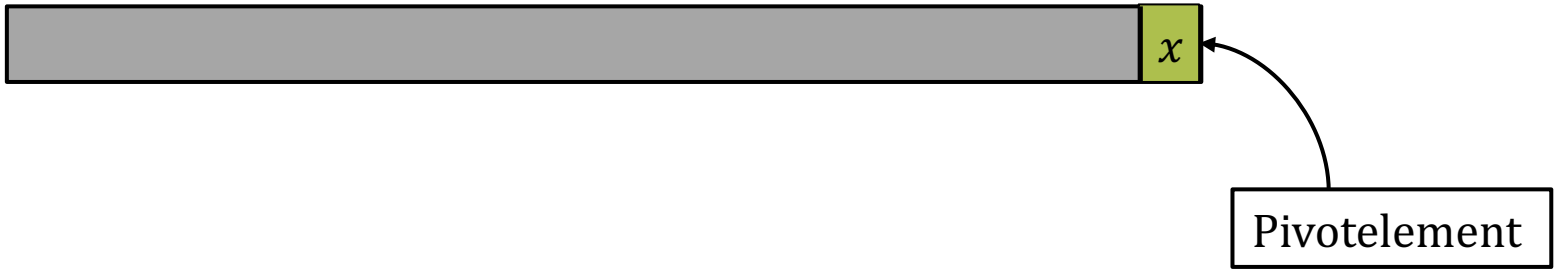
Quicksort

Quicksort – Prinzip

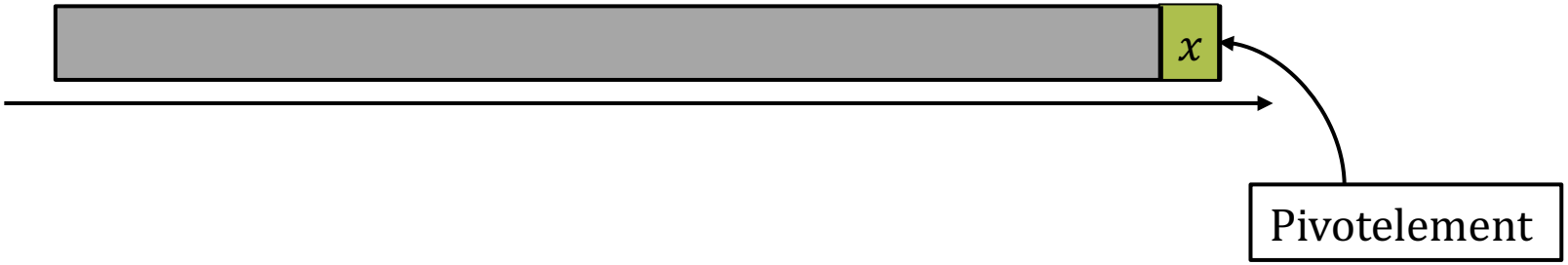
Quicksort – Prinzip

Quicksort – Prinzip

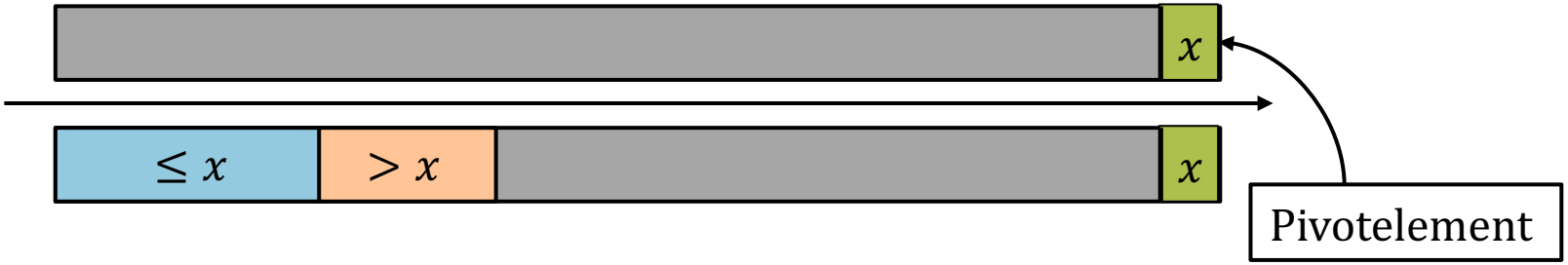
Quicksort – Prinzip



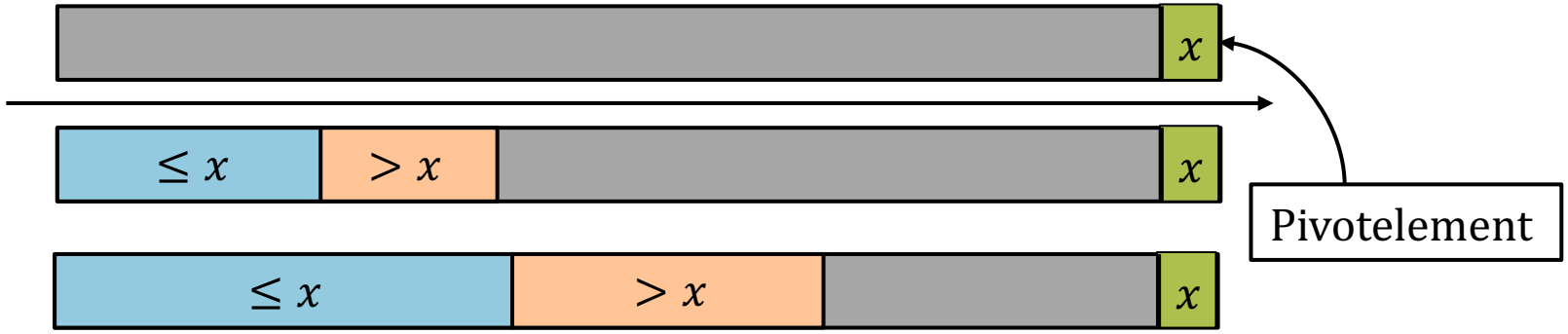
Quicksort – Prinzip



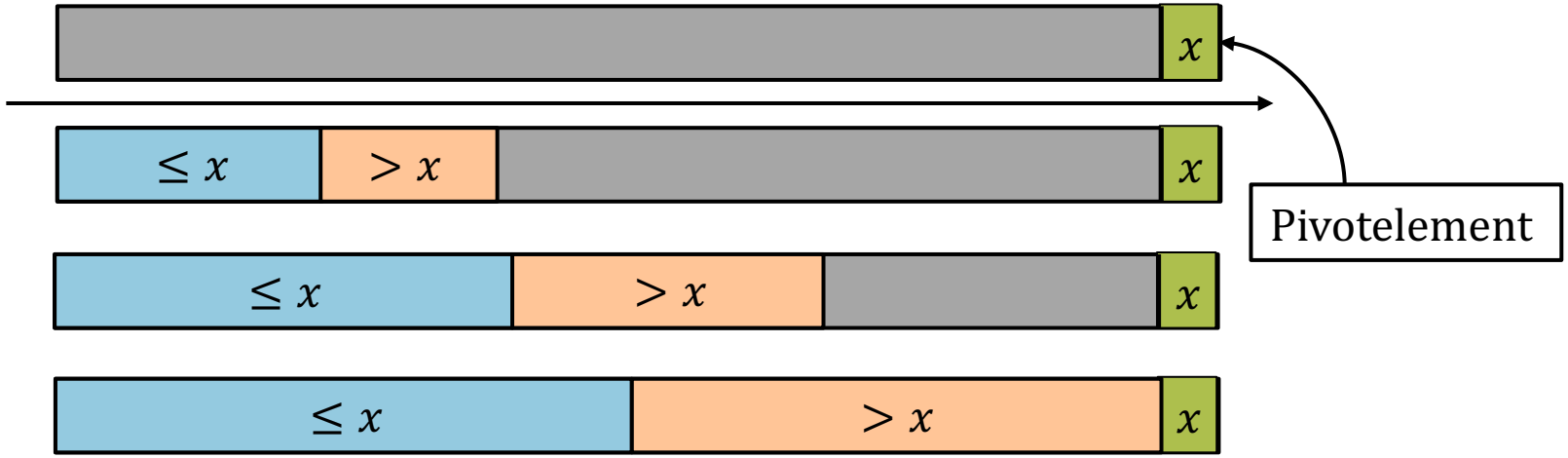
Quicksort – Prinzip



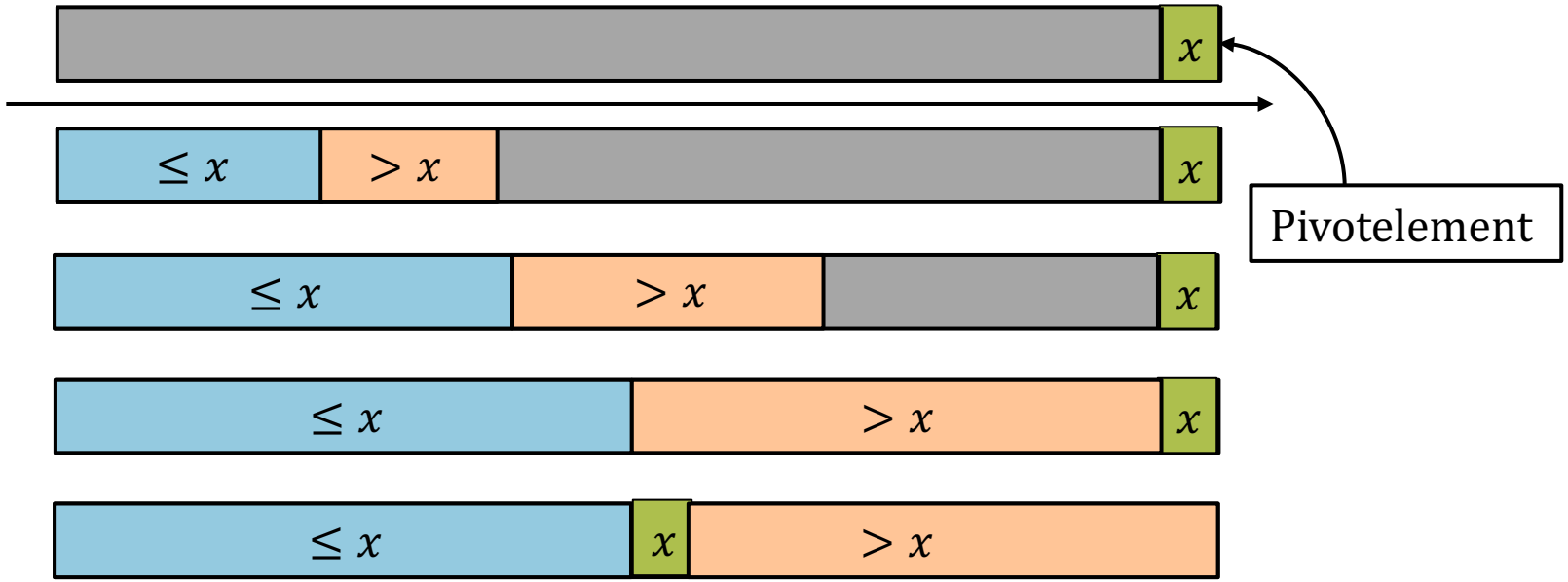
Quicksort – Prinzip



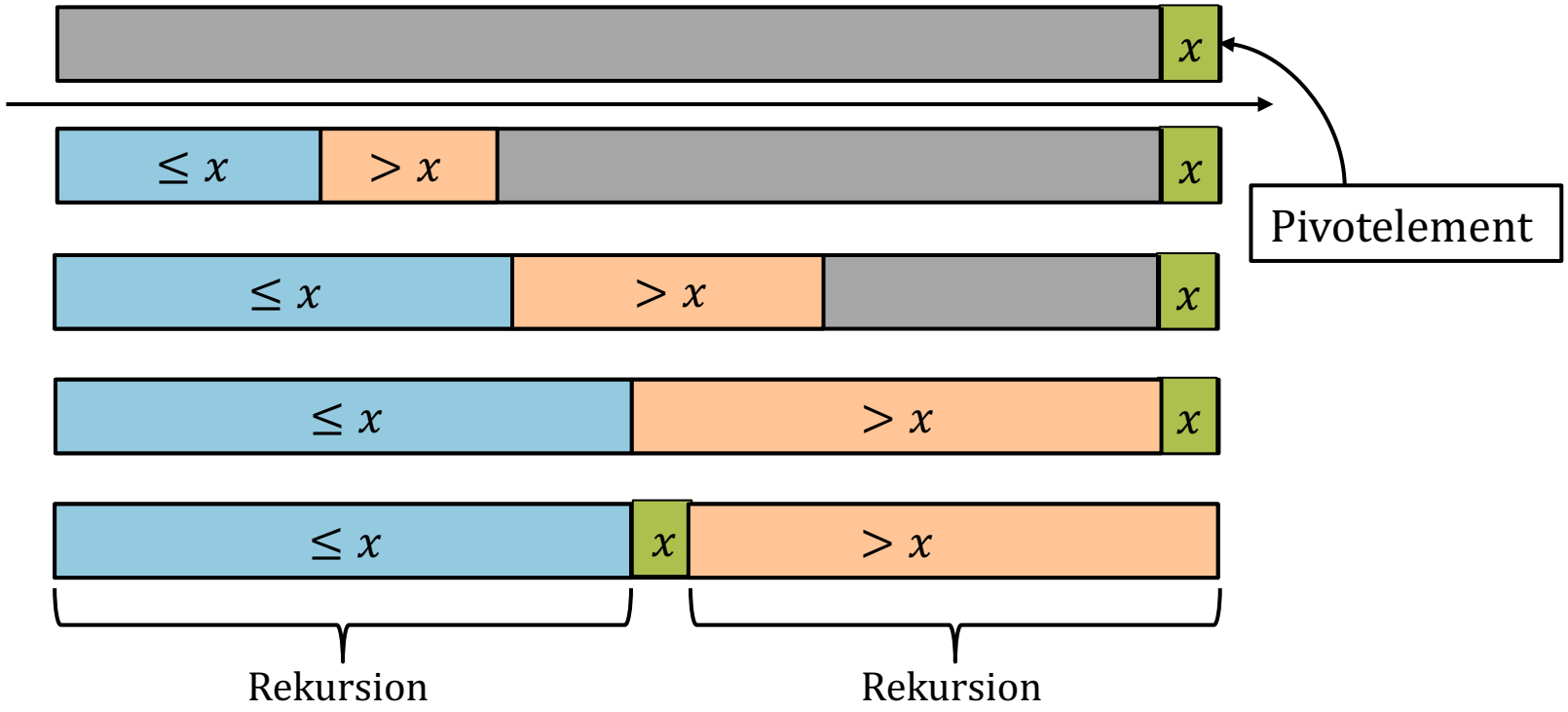
Quicksort – Prinzip



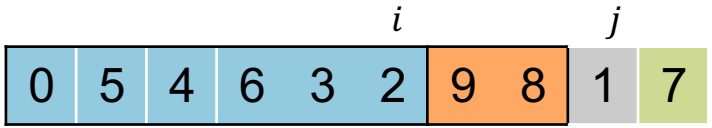
Quicksort – Prinzip



Quicksort – Prinzip

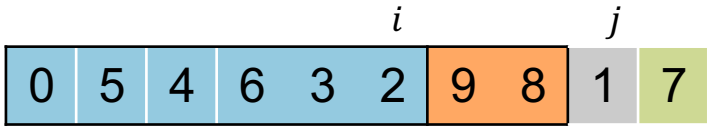


Quicksort – Details



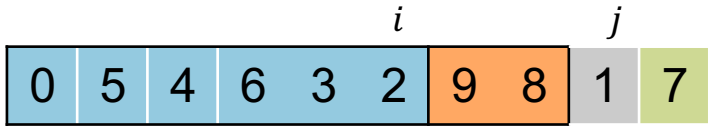
Quicksort – Details

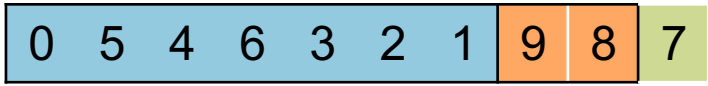
- Innerhalb der rot und blau markierten Bereiche gibt es *keine* Sortierung! Und auch keine Garantie, dass die Elemente während des Partition-Aufrufs diese Reihenfolge behalten.



Quicksort – Details

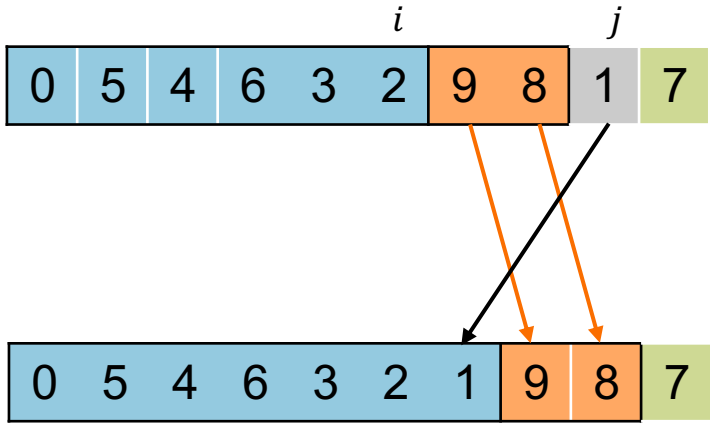
- Innerhalb der rot und blau markierten Bereiche gibt es *keine* Sortierung! Und auch keine Garantie, dass die Elemente während des Partition-Aufrufs diese Reihenfolge behalten.





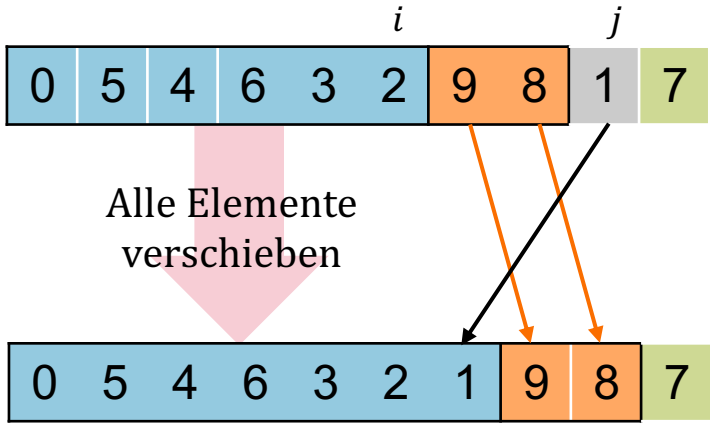
Quicksort – Details

- 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.



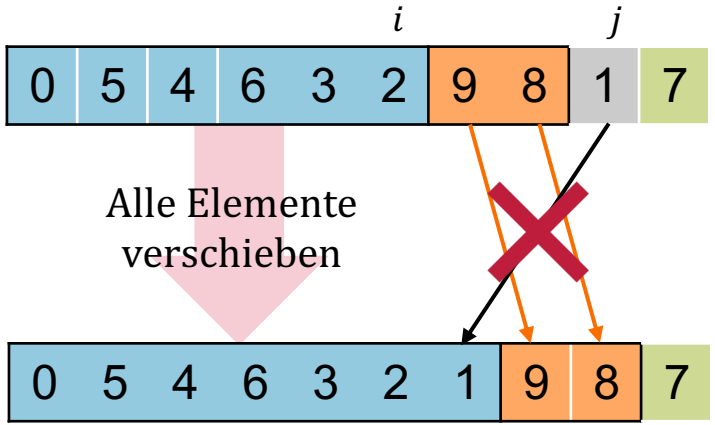
Quicksort – Details

- 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.



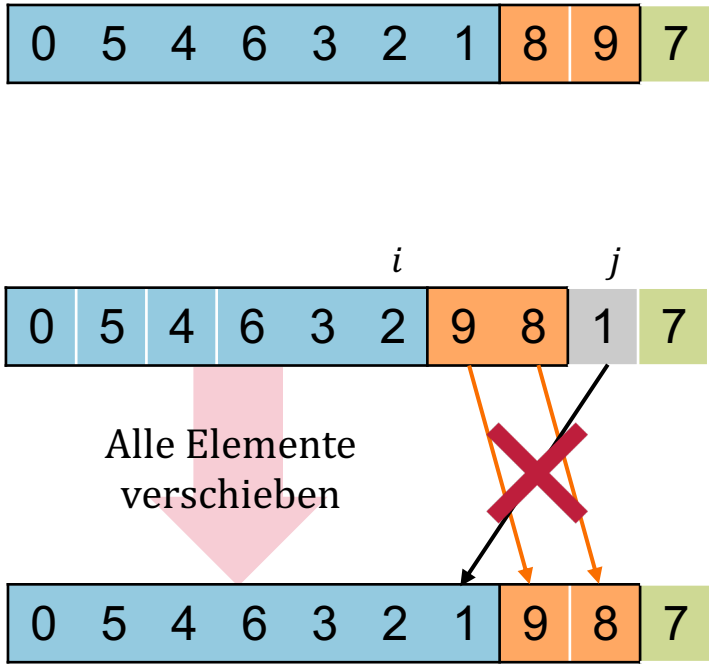
Quicksort – Details

- 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.



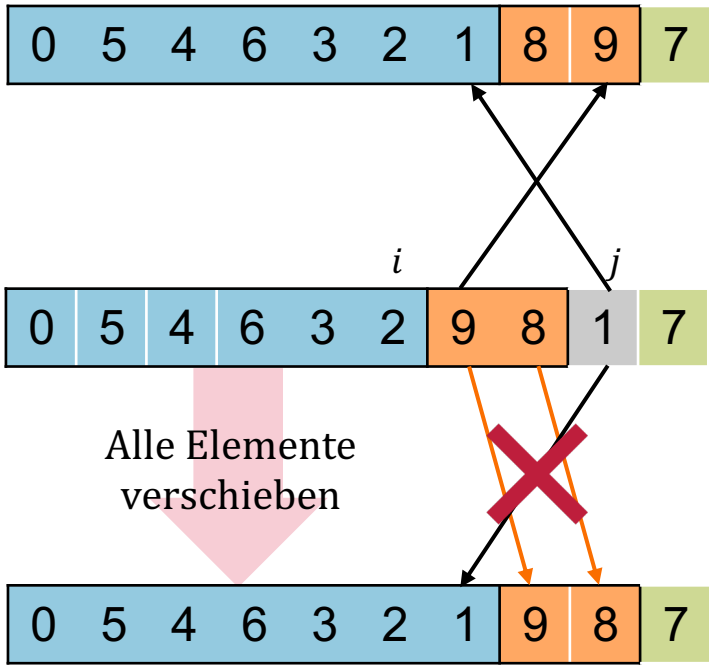
Quicksort – Details

- 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.



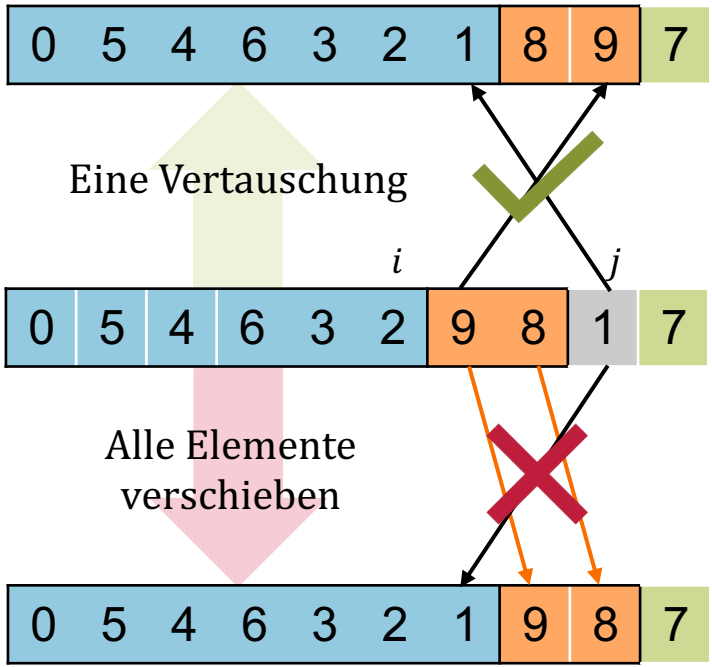
Quicksort – Details

- 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.



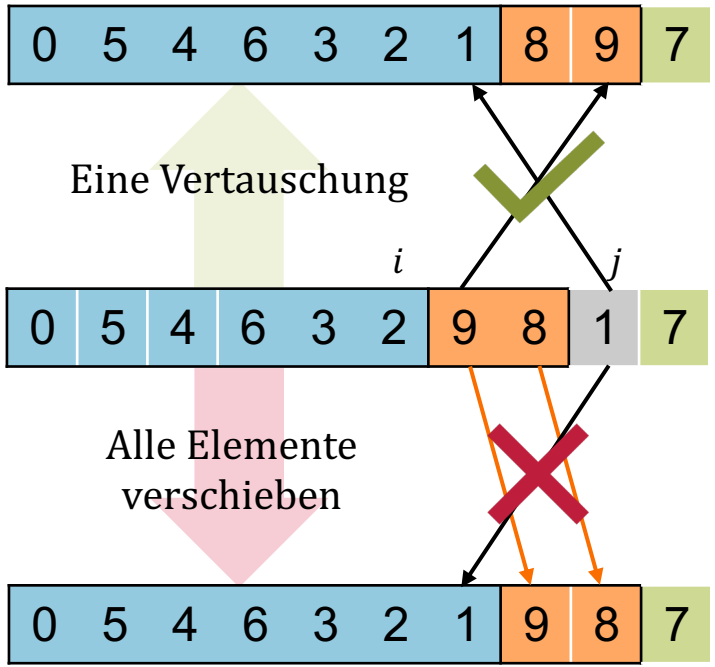
Quicksort – Details

- 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.



Quicksort – Details

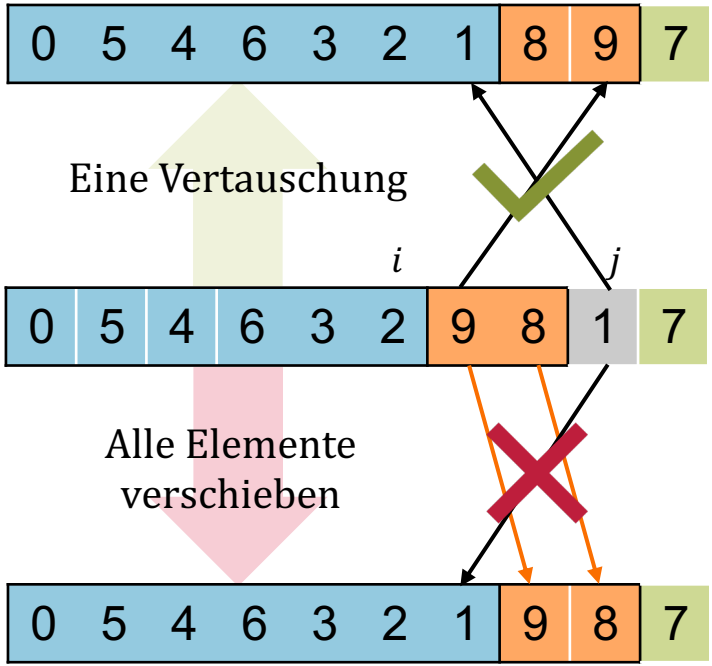
- Innerhalb der rot und blau markierten Bereiche gibt es *keine* Sortierung! Und auch keine Garantie, dass die Elemente während des Partition-Aufrufs diese Reihenfolge behalten.
- Wir “verschieben” nicht den ganzen roten Bereich, sondern tauschen einzelne Paare, um lineare Laufzeit zu erreichen!



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!

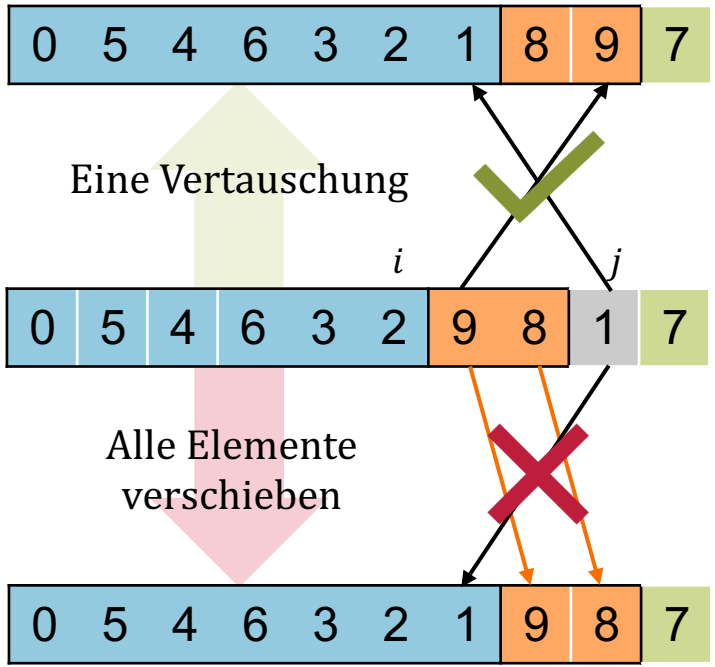


Quicksort – Details

- 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.



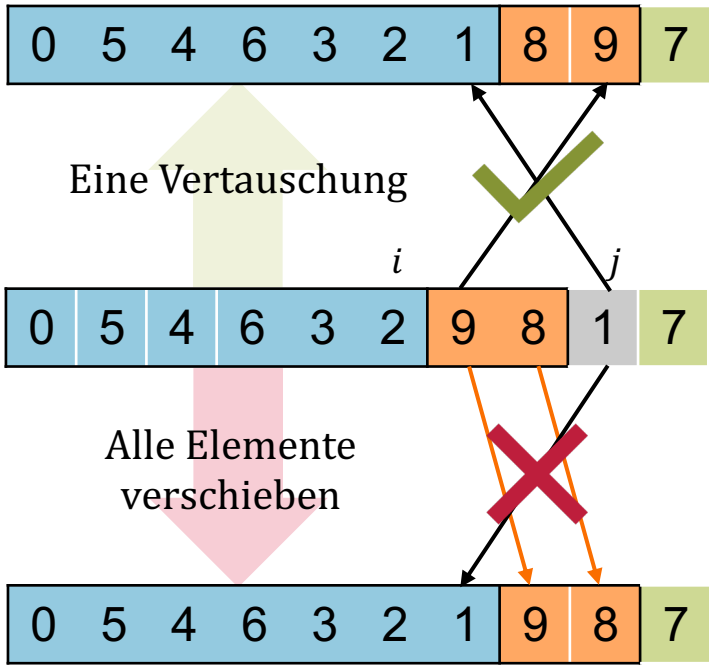
Quicksort – Details

- 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.



Beispiel

Beispiel

Sortiere das folgende Array mit Quicksort:

Beispiel

Sortiere das folgende Array mit Quicksort:

$$4, 1, 5, 13, 2, 13, 9, 3, 8$$

Mediane

Mediane – Definition

Rang-k Element m in X:

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

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

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

Mediane – Definition

Rang-k Element m in X:

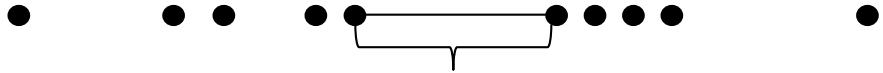
$$|\{x \in X : x \leq m\}| \geq k$$

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!

Mediane – Definition

Rang-k Element m in X:

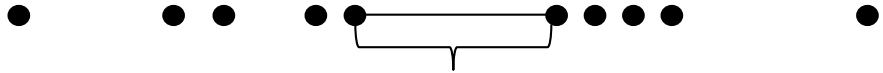
$$|\{x \in X : x \leq m\}| \geq k$$

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.

```
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

```
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

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

```
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

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

$k = 2$

```
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

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

$k = 2$

```
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

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

$k = 2$

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.

Induktionsbeweise

Handshake-Lemma

Handshake-Lemma

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

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.

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:

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).

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:

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 $m \in \mathbb{N}$.

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).

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:

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.

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 $e = \{u, v\}$.

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.

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).

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

Checkliste: Induktion

- IA?

Checkliste: Induktion

- IA?
  - Muss man mehrere Startfälle abdecken?

Checkliste: Induktion

- IA?
  - Muss man mehrere Startfälle abdecken?
- IV?

Checkliste: Induktion

- IA?
  - Muss man mehrere Startfälle abdecken?
- IV?
  - Starke Induktion?

Checkliste: Induktion

- IA?
  - Muss man mehrere Startfälle abdecken?
- IV?
  - Starke Induktion?
- IS?

Checkliste: Induktion

- IA?
  - Muss man mehrere Startfälle abdecken?
- IV?
  - Starke Induktion?
- IS?
  - Starte bei $n + 1$

Checkliste: Induktion

- IA?
  - Muss man mehrere Startfälle abdecken?
- IV?
  - Starke Induktion?
- IS?
  - Starte bei $n + 1$
  - Komme irgendwie auf n

Checkliste: Induktion

- IA?
  - Muss man mehrere Startfälle abdecken?
- IV?
  - Starke Induktion?
- IS?
  - Starte bei $n + 1$
  - Komme irgendwie auf n
  - Setze IV ein (dort gilt unsere Aussage, und die nutzen wir)

Checkliste: Induktion

- IA?
  - Muss man mehrere Startfälle abdecken?
- IV?
  - 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

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Infos zur Klausur

28. Januar 2025 | Allgemein

Hey zusammen!

Die Klausur für Algorithmen und Datenstrukturen findet am **Mittwoch, den 12. Februar 2025** von **07:45-10:15** statt. Die Raumaufteilung werden wir spätestens am 11. Februar auf dieser Seite und per Mailingliste bekannt geben. Die möglichen Räume sind ZI 24.1, ZI 24.2, IZ 24.3 und SN 19.1.

Bitte beachtet die folgenden Punkte:

- Mitbringen:
  - Studierendenausweis (plus Lichtbildausweis, falls kein Foto auf dem Studiausweis vorhanden),
  - dokumentenechter Stift (kein Bleistift, kein rot!)
  - Kein Tipex!
  - Wörterbuch, falls ihr das benötigt
- Weitere Unterlagen sind *nicht* erlaubt!
- Eigenes Papier ist *nicht* erlaubt, wir stellen Papier.
- Die Klausur startet 08:00 und dauert 2 Stunden.

Wir wünschen euch ganz viel Erfolg!!! 🙂

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Vorbereitung

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Vorbereitung

- Hausaufgaben

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Vorbereitung

- Hausaufgaben
- Tutorien (Tutoren fragen)

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Vorbereitung

- Hausaufgaben
- Tutorien (Tutoren fragen)
- Vorlesungsvideos & Folien

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Vorbereitung

- Hausaufgaben
- Tutorien (Tutoren fragen)
- Vorlesungsvideos & Folien
- Skript

Klausur

Eckdaten

- 12.02.2025 von 07:45 Uhr bis 10:15
- Klausur dauert 2h
- Raumaufteilung wird kurz vorher bekanntgegeben (Mailingliste & Webseite)
- Mehr Infos: Webseite!

Vorbereitung

- Hausaufgaben
- Tutorien (Tutoren fragen)
- Vorlesungsvideos & Folien
- Skript
- Alte Klausuren

Bei Fragen …

Bei Fragen …

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

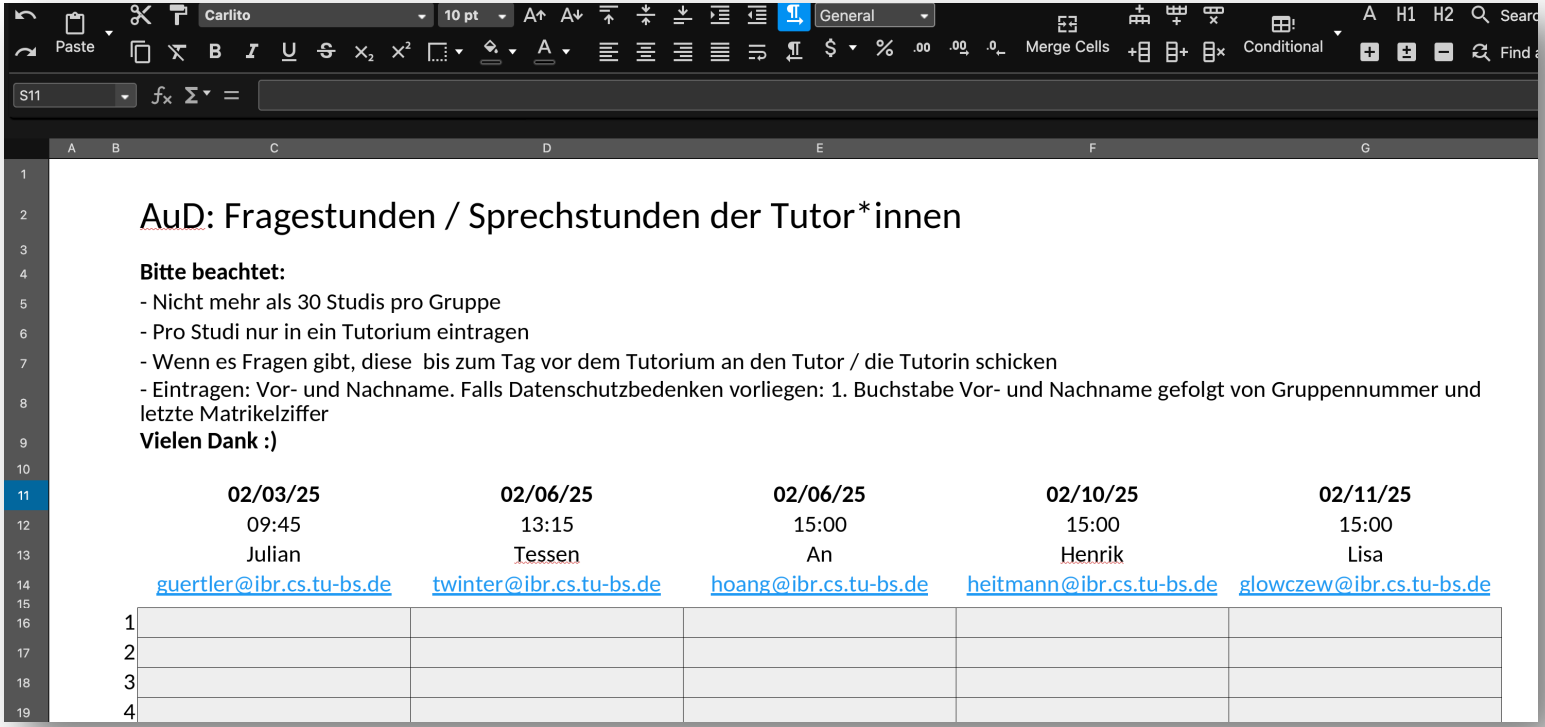
Bei Fragen …

Bei Fragen per Mail:
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!

Sprechstunden

Sprechstunden



AuD: Fragestunden / Sprechstunden der Tutor*innen

Bitte beachtet:

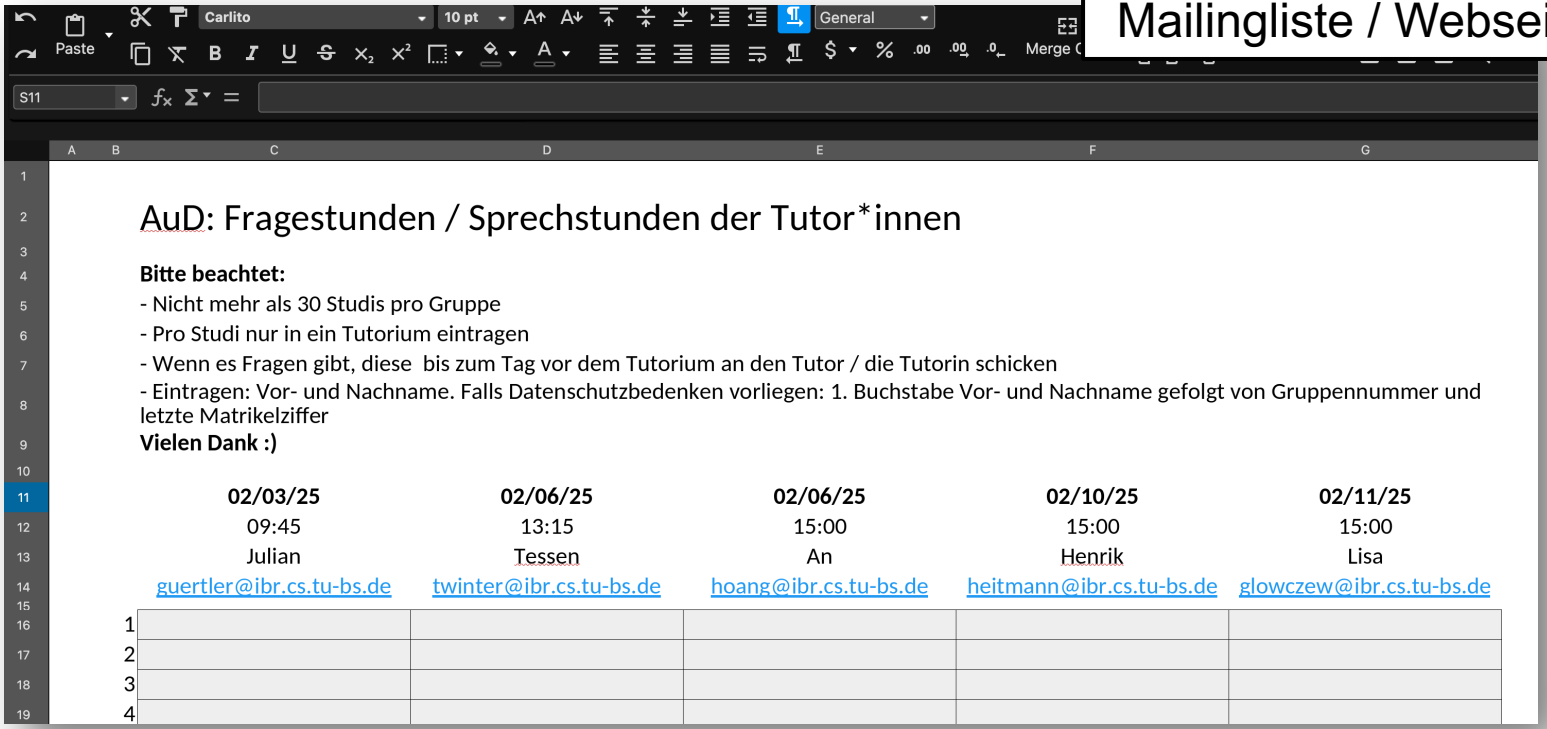
- Nicht mehr als 30 Studis pro Gruppe
- Pro Studi nur in ein Tutorium eintragen
- Wenn es Fragen gibt, diese bis zum Tag vor dem Tutorium an den Tutor / die Tutorin schicken
- Eintragen: Vor- und Nachname. Falls Datenschutzbedenken vorliegen: 1. Buchstabe Vor- und Nachname gefolgt von Gruppennummer und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

Vielen Dank :)

	02/03/25	**02/06/25**	**02/06/25**	**02/10/25**	**02/11/25**
	09:45	13:15	15:00	15:00	15:00
	Julian	Tessen	An	Henrik	Lisa
	guertler@ibr.cs.tu-bs.de	twinter@ibr.cs.tu-bs.de	hoang@ibr.cs.tu-bs.de	heitmann@ibr.cs.tu-bs.de	glowczew@ibr.cs.tu-bs.de
1					
2					
3					
4					

Sprechstunden

Link zur Anmeldung per Mailingliste / Webseite!



AuD: Fragestunden / Sprechstunden der Tutor*innen

Bitte beachtet:

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

Vielen Dank :)

	02/03/25	**02/06/25**	**02/06/25**	**02/10/25**	**02/11/25**
	09:45	13:15	15:00	15:00	15:00
	Julian	Tessen	An	Henrik	Lisa
	guertler@ibr.cs.tu-bs.de	twinter@ibr.cs.tu-bs.de	hoang@ibr.cs.tu-bs.de	heitmann@ibr.cs.tu-bs.de	glowczew@ibr.cs.tu-bs.de
1					
2					
3					
4					

Veranstaltungen im Sommer

Veranstaltung	Empfehlung	Inhalte
Theoretische Informatik 2	4. Semester Theo Inf 1 bereits bestanden	Turingmaschinen Berechnbarkeit Komplexitätsklassen
Algorithmen und Datenstrukturen 2 (Wahlpflicht Informatik / Wirtschaftsinformatik)	2. Semester AuD bereits bestanden	Optimierungsprobleme Heuristiken Approximationen Exakte Algorithmen Hashing
Einführung in algorithmische Geometrie (Neu! Wahlpflicht Informatik)	AuD bereits bestanden (Theo Inf 2 bereits bestanden)	Range-Queries Lokalisierungsprobleme Motion Planning Art-Gallery-Problem

Falls ihr noch ausgedruckte Altklausuren braucht…

(Alle Altklausuren gibt es natürlich auch auf der Webseite, aber die muss man dann mühsam ausdrucken.)

Fragerunde


Viel Erfolg!!!