

Du interessierst dich für die Arbeit der
Fachschaft und möchtest dich vielleicht
gerne selbst engagieren?

Wir zeigen dir, wie die Fachschaft
organisiert ist, was ihre Aufgaben sind
und wie du dich bei uns einbringen
kannst.

Essen / Snacks /
Getränke

Semester- auftakttreffen

Do. 25.04. 19:00 Uhr



Algorithmen 1

Übung 1 Asymptotik, Pseudocode & Amortisierte Analyse



Organisatorisches

Organisatorisches

Algo1-Team

Organisatorisches

Algo1-Team



Thomas



Jean-Pierre



Marcus



Wendy

Organisatorisches

Algo1-Team



Thomas
└──
Vorlesung



Jean-Pierre



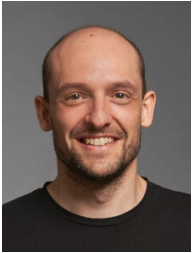
Marcus



Wendy

Organisatorisches

Algo1-Team



Thomas
└──
Vorlesung



Jean-Pierre



Marcus

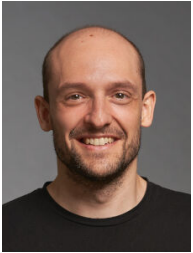


Wendy

└──
Übung

Organisatorisches

Algo1-Team



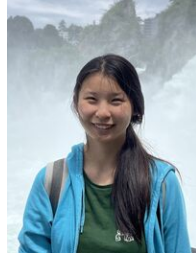
Thomas
└──
Vorlesung



Jean-Pierre



Marcus



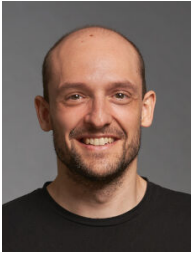
Wendy

└──
Übung

Übung?

Organisatorisches

Algo1-Team



Thomas

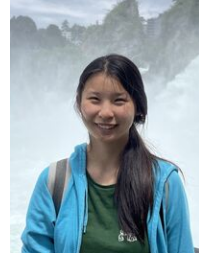
Vorlesung



Jean-Pierre



Marcus



Wendy

Übung

Übung?

- Wiederholung des Vorlesungsstoffes

Organisatorisches

Algo1-Team



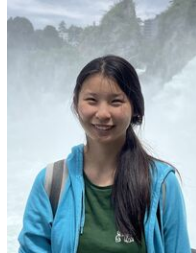
Thomas



Jean-Pierre



Marcus



Wendy

Vorlesung

Übung

Übung?

- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung

Organisatorisches

Algo1-Team



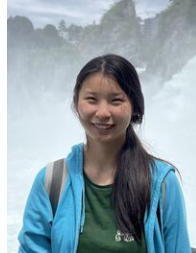
Thomas



Jean-Pierre



Marcus



Wendy

Vorlesung

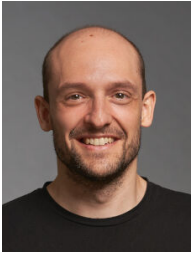
Übung

Übung?

- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel

Organisatorisches

Algo1-Team



Thomas

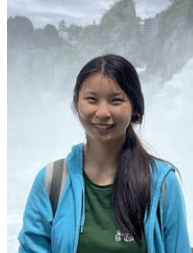
Vorlesung



Jean-Pierre



Marcus



Wendy

Übung

Übungsblätter

Übung?

- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel



Organisatorisches

Algo1-Team



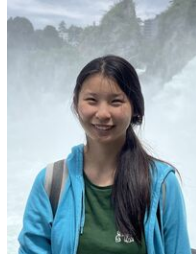
Thomas



Jean-Pierre



Marcus



Wendy

Vorlesung

Übung

Übung?

- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel

Übungsblätter

- werden **nicht** in der Übung besprochen

Organisatorisches

Algo1-Team



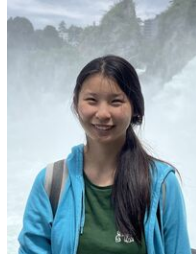
Thomas



Jean-Pierre



Marcus



Wendy

Vorlesung

Übung

Übung?

- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel
- Fragen!

Übungsblätter

- werden **nicht** in der Übung besprochen

Organisatorisches

Algo1-Team



Thomas

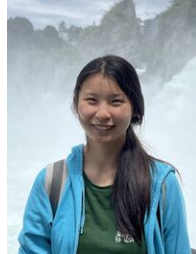
Vorlesung



Jean-Pierre



Marcus



Wendy

Übung

Übung?

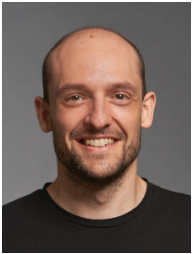
- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel
- Fragen!

Übungsblätter

- werden **nicht** in der Übung besprochen
- Mittwoch 15:30 Uhr – Freitag 18:00 Uhr
(eine Woche später)

Organisatorisches

Algo1-Team



Thomas

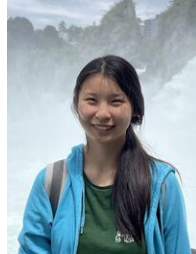
Vorlesung



Jean-Pierre



Marcus



Wendy

Übung

Übung?

- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel
- Fragen!

Übungsblätter

- werden **nicht** in der Übung besprochen
- Mittwoch 15:30 Uhr – Freitag 18:00 Uhr (eine Woche später)
- Abgabe zu zweit erlaubt (und erwünscht)

Organisatorisches

Algo1-Team



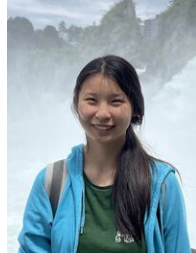
Thomas
 └──
 Vorlesung



Jean-Pierre



Marcus



Wendy

└──
 Übung

Übung?

- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel
- Fragen!

Übungsblätter

- werden **nicht** in der Übung besprochen
- Mittwoch 15:30 Uhr – Freitag 18:00 Uhr (eine Woche später)
- Abgabe zu zweit erlaubt (und erwünscht)
- Abschreiben \Rightarrow 0 Punkte auf das Blatt (nochmal: kein Bonus mehr)

Organisatorisches

Algo1-Team



Thomas
 └──
 Vorlesung



Jean-Pierre



Marcus



Wendy

└──
 Übung

Übung?

- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel
- Fragen!

Übungsblätter

- werden **nicht** in der Übung besprochen
- Mittwoch 15:30 Uhr – Freitag 18:00 Uhr (eine Woche später)
- Abgabe zu zweit erlaubt (und erwünscht)
- Abschreiben \Rightarrow 0 Punkte auf das Blatt (nochmal: kein Bonus mehr)

Literatur zur Vorlesung

- Folien sollten für Verständnis reichen

Organisatorisches

Algo1-Team



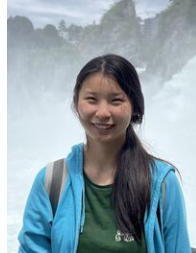
Thomas
 ┌
 Vorlesung



Jean-Pierre



Marcus



Wendy

└
 Übung

Übung?

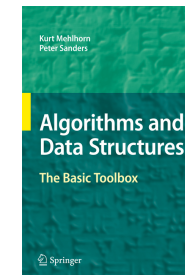
- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel
- Fragen!

Übungsblätter

- werden **nicht** in der Übung besprochen
- Mittwoch 15:30 Uhr – Freitag 18:00 Uhr (eine Woche später)
- Abgabe zu zweit erlaubt (und erwünscht)
- Abschreiben \Rightarrow 0 Punkte auf das Blatt (nochmal: kein Bonus mehr)

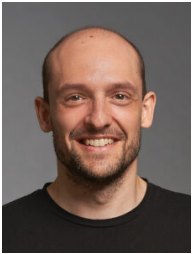
Literatur zur Vorlesung

- Folien sollten für Verständnis reichen
- Für weitergehende Recherche:



Organisatorisches

Algo1-Team



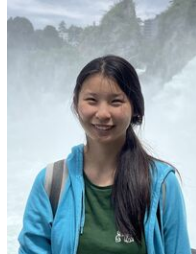
Thomas
 └──
 Vorlesung



Jean-Pierre



Marcus



Wendy

└──
 Übung

Übung?

- Wiederholung des Vorlesungsstoffes
- inkl. Vertiefung und Erweiterung
- manchmal anderer Blickwinkel
- Fragen!

Discord für Fragen/Tutorien:

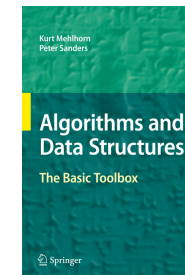
<https://scale.iti.kit.edu/teaching/2024ss/algo1/start>

Übungsblätter

- werden **nicht** in der Übung besprochen
- Mittwoch 15:30 Uhr – Freitag 18:00 Uhr (eine Woche später)
- Abgabe zu zweit erlaubt (und erwünscht)
- Abschreiben \Rightarrow 0 Punkte auf das Blatt (nochmal: kein Bonus mehr)

Literatur zur Vorlesung

- Folien sollten für Verständnis reichen
- Für weitergehende Recherche:



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails

Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, O -Notation, big- O notation



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, O -Notation, big- O notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, O -Notation, big- O notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- n bezeichnet die Größe der Eingabe



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, O -Notation, big- O notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- n bezeichnet die **Größe der Eingabe** ?

Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, *O*-Notation, big-*O* notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- n bezeichnet die **Größe der Eingabe** ?
 - Multiplikation von 2 Zahlen mit x Ziffern



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, *O*-Notation, big-*O* notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- n bezeichnet die **Größe der Eingabe** ?

- Multiplikation von 2 Zahlen mit x Ziffern $\rightarrow n = x$



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, *O*-Notation, big-*O* notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- n bezeichnet die **Größe der Eingabe** ?
 - Multiplikation von 2 Zahlen mit x Ziffern $\rightarrow n = x$
 - Suche in einer Menge von k Zahlen die je maximal x Ziffern haben



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, *O*-Notation, big-*O* notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- n bezeichnet die **Größe der Eingabe** ?

- Multiplikation von 2 Zahlen mit x Ziffern $\rightarrow n = x$

Abstraktionsniveau! ■ Suche in einer Menge von k Zahlen die je maximal x Ziffern haben



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, *O*-Notation, big-*O* notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- n bezeichnet die **Größe der Eingabe** ?

- Multiplikation von 2 Zahlen mit x Ziffern $\rightarrow n = x$

- Abstraktionsniveau!
- Suche in einer Menge von k Zahlen die je maximal x Ziffern haben
 $\rightarrow n = k$



Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, *O*-Notation, big-*O* notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- n bezeichnet die Größe der Eingabe
- Intuitiv: Wenn die Eingabe ausreichend groß ist (mindestens n_0), dann braucht ein Algorithmus mit Laufzeit $f(n)$ (ungefähr) höchstens so lang wie einer mit Laufzeit $g(n)$.



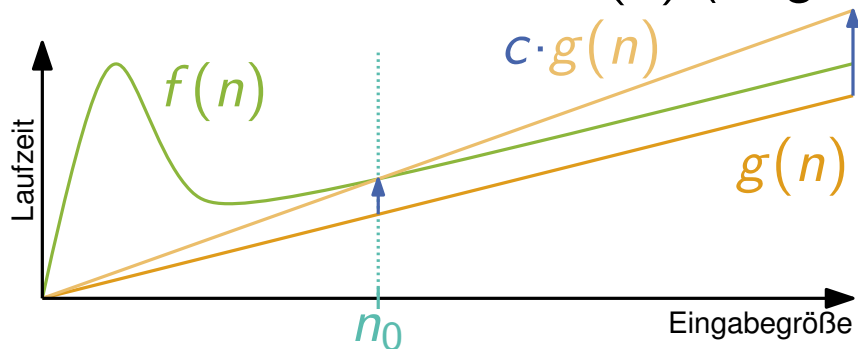
Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, O -Notation, big- O notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

- n bezeichnet die Größe der Eingabe
- Intuitiv: Wenn die Eingabe ausreichend groß ist (mindestens n_0), dann braucht ein Algorithmus mit Laufzeit $f(n)$ (ungefähr) höchstens so lang wie einer mit Laufzeit $g(n)$.



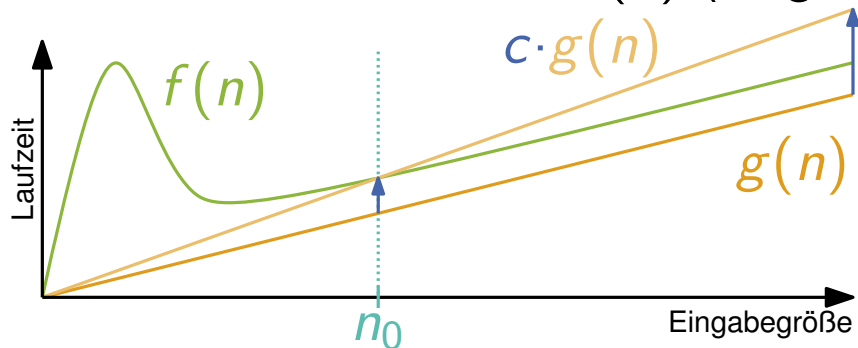
Asymptotik

- Werkzeug zur Klassifizierung der Laufzeit von Algorithmen
- Abstrahieren von Implementierungs- bzw. Hardwaredetails
- Vereinfacht die Analyse

Landau-Notation, O -Notation, big- O notation

$f(n) \in O(g(n))$ $f(n)$ wächst max. so schnell wie $g(n)$ $\exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

- n bezeichnet die Größe der Eingabe
- Intuitiv: Wenn die Eingabe ausreichend groß ist (mindestens n_0), dann braucht ein Algorithmus mit Laufzeit $f(n)$ (ungefähr) höchstens so lang wie einer mit Laufzeit $g(n)$.



$f(n) \in \omega(g(n))$ $f(n)$ wächst schneller als $g(n)$

$f(n) \in \Omega(g(n))$ $f(n)$ wächst min. so schnell wie $g(n)$

$f(n) \in \Theta(g(n))$ $f(n)$ und $g(n)$ wachsen gleich schnell

$f(n) \in o(g(n))$ $f(n)$ wächst langsamer als $g(n)$

Asymptotik – Beispiele

Was wächst schneller?

Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$\blacksquare f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$$

Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$5n^2 \leq c \cdot 2n^3$$

$$\blacksquare f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$$



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$5n^2 \leq c \cdot 2n^3$$

$$n^2 \leq \frac{2c}{5} \cdot n^3$$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$\begin{aligned} 5n^2 &\leq c \cdot 2n^3 \\ n^2 &\leq \frac{2c}{5} \cdot n^3 \quad c = \frac{5}{2} \end{aligned}$$

$$\blacksquare f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$$



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$5n^2 \leq c \cdot 2n^3$$

$$n^2 \leq \frac{2c}{5} \cdot n^3 \quad c = \frac{5}{2}$$

$$n^2 \leq n^3$$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$\begin{array}{l}
 5n^2 \leq c \cdot 2n^3 \\
 n^2 \leq \frac{2c}{5} \cdot n^3 \quad c = \frac{5}{2} \\
 n^2 \leq n^3 \\
 1 \leq n
 \end{array}$$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$\begin{array}{l}
 5n^2 \leq c \cdot 2n^3 \\
 n^2 \leq \frac{2c}{5} \cdot n^3 \quad c = \frac{5}{2} \\
 n^2 \leq n^3 \\
 1 \leq n \quad n_0 = 1
 \end{array}$$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$\begin{array}{l}
 5n^2 \leq c \cdot 2n^3 \\
 n^2 \leq \frac{2c}{5} \cdot n^3 \quad c = \frac{5}{2} \\
 n^2 \leq n^3 \\
 1 \leq n \quad n_0 = 1
 \end{array}$$

Für $c = \frac{5}{2}$ gilt für alle $n > 1$: $5n^2 \leq c \cdot 2n^3$.

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10}n^2 \text{ oder } \frac{1}{10^{10}}n^3$$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung	$5n^2 \leq c \cdot 2n^3$	
	$n^2 \leq \frac{2c}{5} \cdot n^3$	$c = \frac{5}{2}$
	$n^2 \leq n^3$	
	$1 \leq n$	$n_0 = 1$

Für $c = \frac{5}{2}$ gilt für alle $n > 1$: $5n^2 \leq c \cdot 2n^3$.

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10}n^2 \text{ oder } \frac{1}{10^{10}}n^3$$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$\begin{array}{l} 5n^2 \leq c \cdot 2n^3 \\ n^2 \leq \frac{2c}{5} \cdot n^3 \quad c = \frac{5}{2} \\ n^2 \leq n^3 \\ 1 \leq n \quad n_0 = 1 \end{array}$$

Für $c = \frac{5}{2}$ gilt für alle $n > 1$: $\cancel{5}n^2 \leq c \cdot \cancel{2}n^3$.

$$\blacksquare f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10}n^2 \text{ oder } \frac{1}{10^{10}}n^3$$

Zeige $5n^2 \in O(2n^3)$:

Nebenrechnung

$$\begin{aligned} 5n^2 &\leq c \cdot 2n^3 \\ n^2 &\leq \frac{2c}{5} \cdot n^3 & c &= \frac{5}{2} \\ n^2 &\leq n^3 \\ 1 &\leq n & n_0 &= 1 \end{aligned}$$

Für $c = \frac{10^{10}}{2}$ gilt für alle $n > 1$: $\frac{10^{10}}{2}n^2 \leq \frac{10^{-10}}{2}n^3$.

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$
- kontante Faktoren weglassen



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$
- kontante Faktoren weglassen

Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

ist beides in $2^{\Theta(n)}$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$
- kontante Faktoren weglassen

Asymptotik – Beispiele

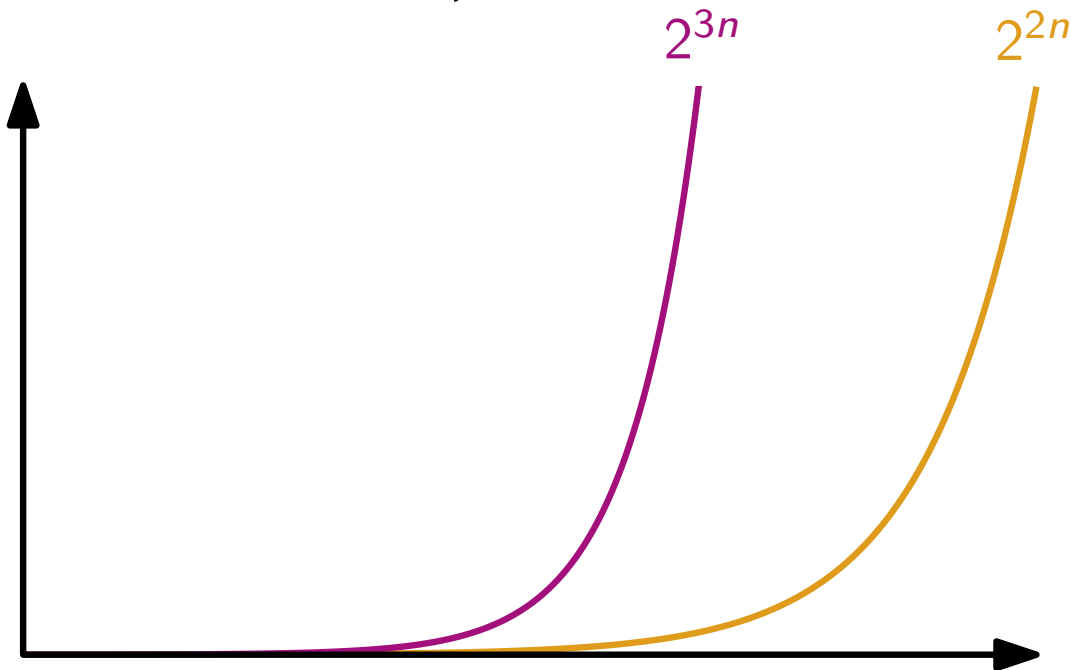
Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

ist beides in $2^{\Theta(n)}$, aber:



- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$
- kontante Faktoren weglassen

Asymptotik – Beispiele

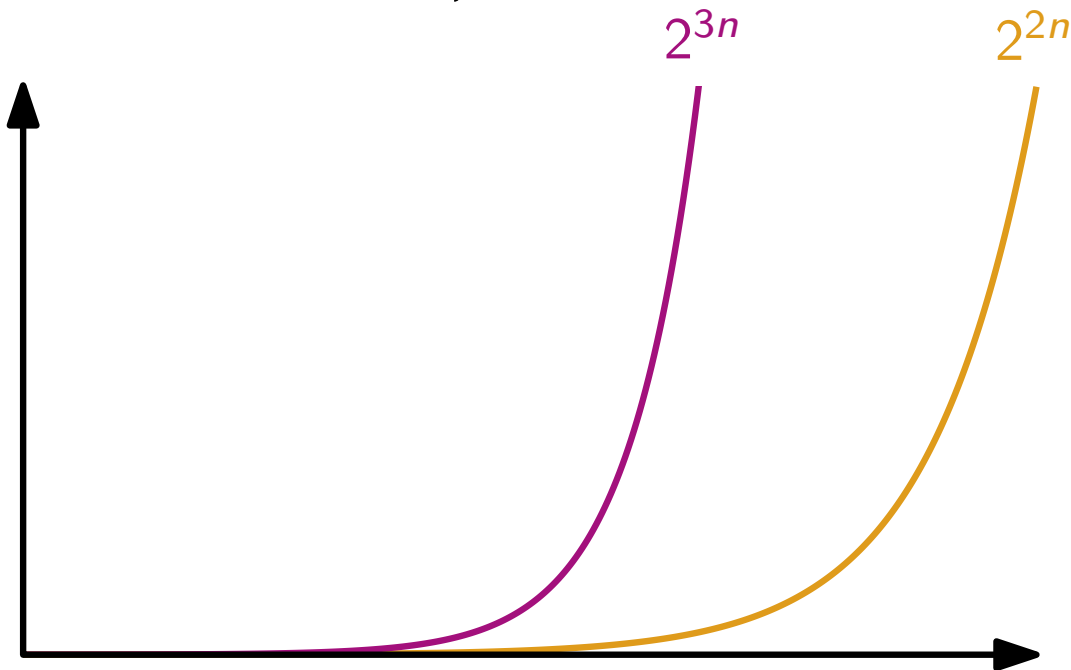
Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

ist beides in $2^{\Theta(n)}$, aber:



■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen

nicht im Exponenten!

■ Plots können helfen

Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen

nicht im Exponenten!

■ Plots können helfen

Asymptotik – Beispiele

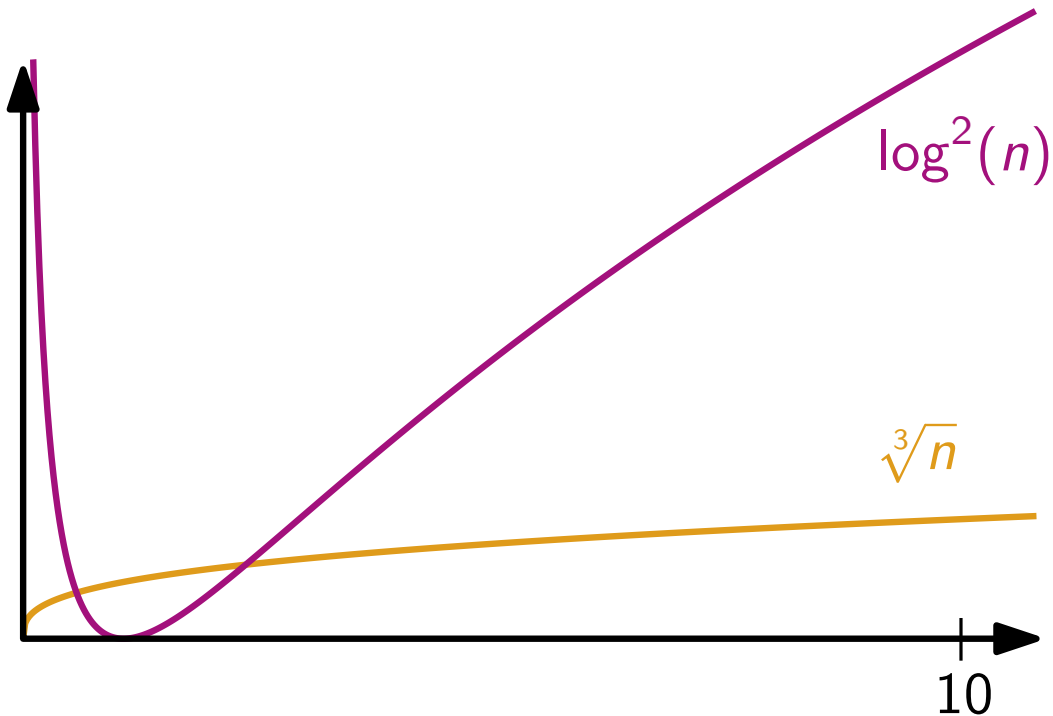
Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$



■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen

nicht im Exponenten!

■ Plots können helfen

Asymptotik – Beispiele

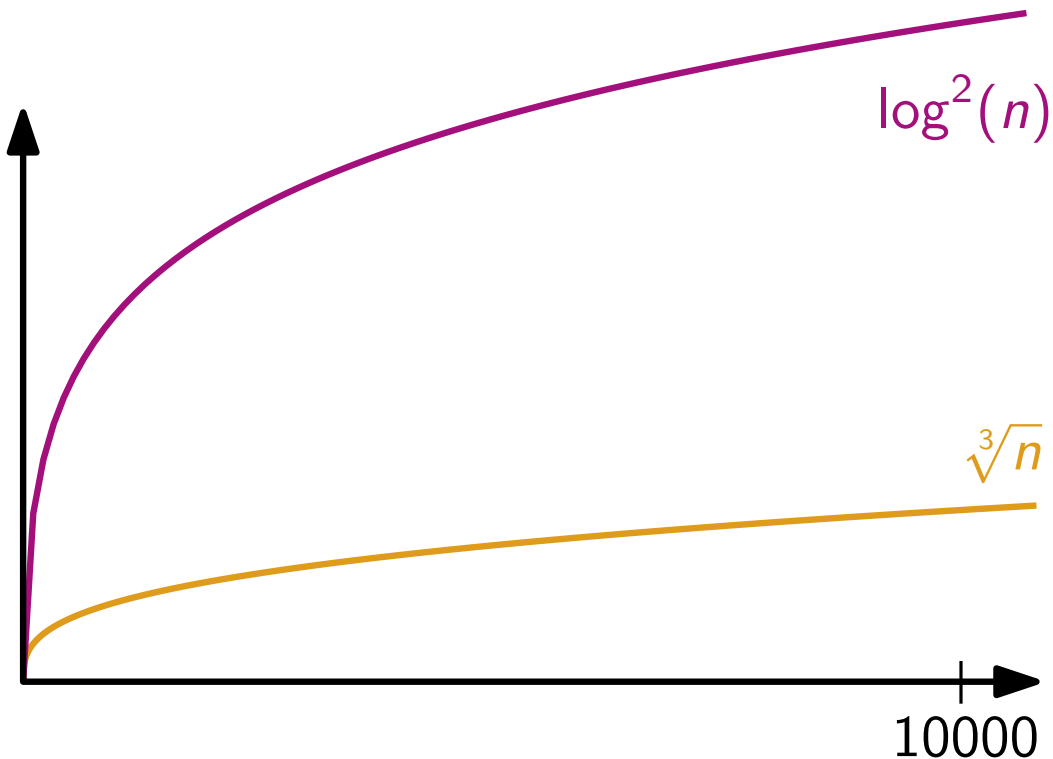
Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$



■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen

nicht im Exponenten!

■ Plots können helfen

Asymptotik – Beispiele

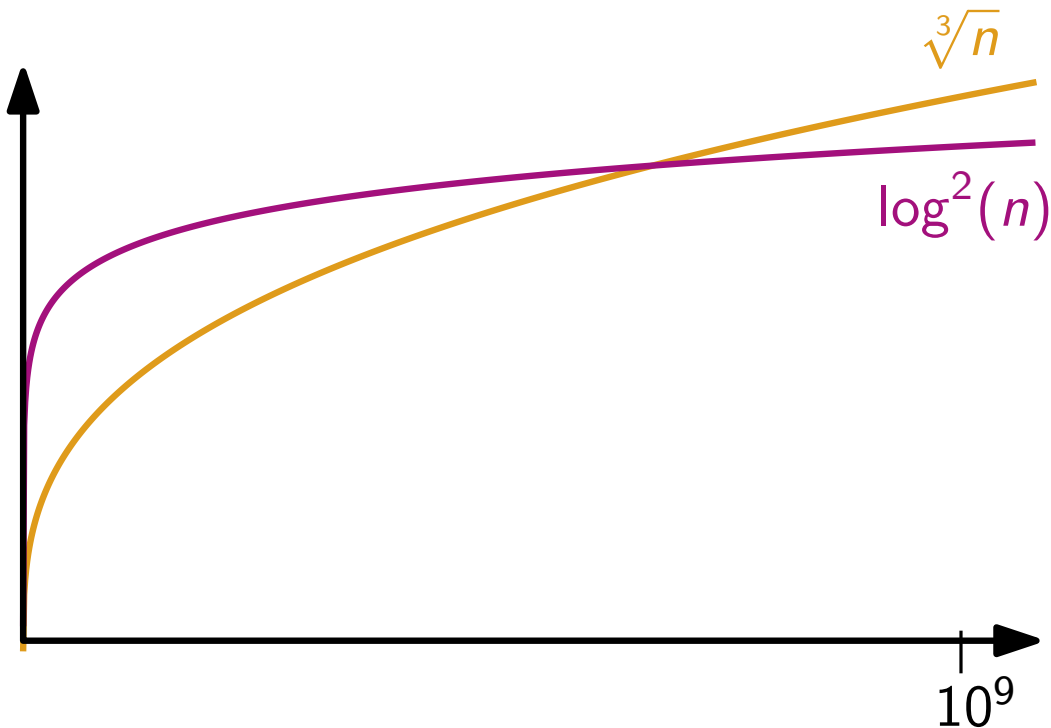
Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$



■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen

nicht im Exponenten!

■ Plots können helfen



Asymptotik – Beispiele

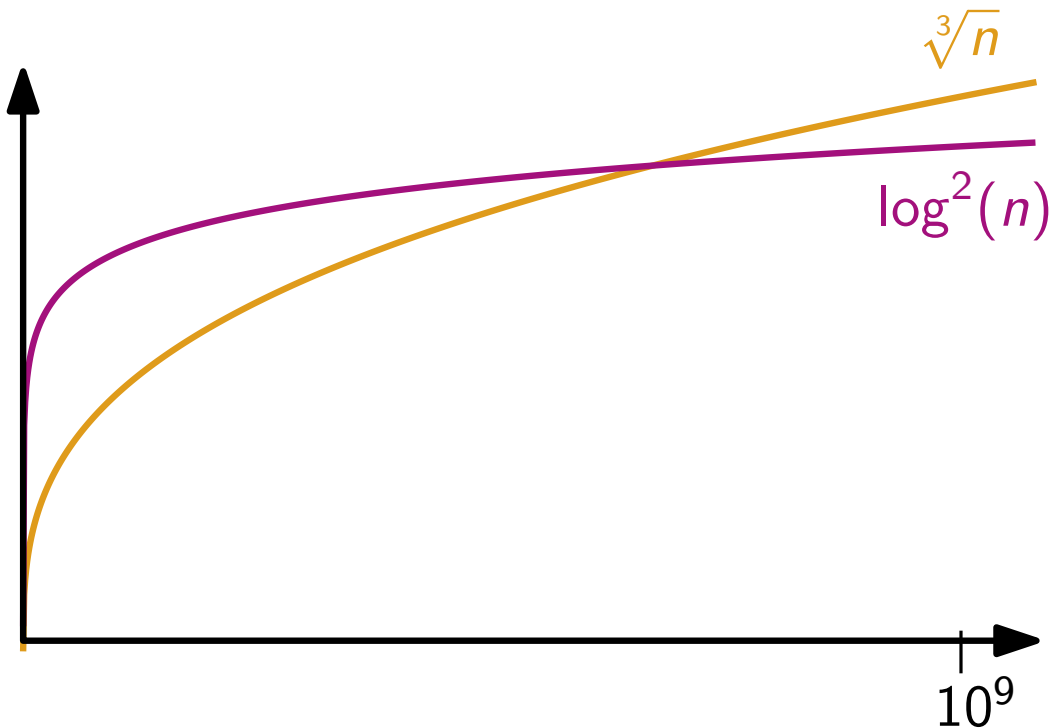
Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$



■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen

nicht im Exponenten!

■ Plots können helfen

nicht immer!



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen

nicht im Exponenten!

■ Plots können helfen

nicht immer!

■ $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10} n^2 \text{ oder } \frac{1}{10^{10}} n^3$$

$$2^{3n} \text{ oder } 2^{2n}$$

$$\log^2(n) \text{ oder } \sqrt[3]{n}$$

$$\lim_{n \rightarrow \infty} \frac{\log^2(n)}{\sqrt[3]{n}}$$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

- kontante Faktoren weglassen

nicht im Exponenten!

- Plots können helfen

nicht immer!

- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10} n^2 \text{ oder } \frac{1}{10^{10}} n^3$$

$$2^{3n} \text{ oder } 2^{2n}$$

$$\log^2(n) \text{ oder } \sqrt[3]{n}$$

$$\lim_{n \rightarrow \infty} \frac{\log^2(n) \xrightarrow{n \rightarrow \infty} \infty}{\sqrt[3]{n} \xrightarrow{n \rightarrow \infty} \infty}$$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

- kontante Faktoren weglassen

nicht im Exponenten!

- Plots können helfen

nicht immer!

- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10} n^2 \text{ oder } \frac{1}{10^{10}} n^3$$

$$2^{3n} \text{ oder } 2^{2n}$$

$$\log^2(n) \text{ oder } \sqrt[3]{n}$$

$$\lim_{n \rightarrow \infty} \frac{\log^2(n) \xrightarrow{n \rightarrow \infty} \infty}{\sqrt[3]{n} \xrightarrow{n \rightarrow \infty} \infty}$$

beides ableiten (L'Hôpital)

$$= \lim_{n \rightarrow \infty} \frac{2 \log(n) \cdot \frac{1}{n}}{\frac{1}{3} n^{-2/3}}$$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

- kontante Faktoren weglassen **nicht im Exponenten!**

- Plots können helfen **nicht immer!**

- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10} n^2 \text{ oder } \frac{1}{10^{10}} n^3$$

$$2^{3n} \text{ oder } 2^{2n}$$

$$\log^2(n) \text{ oder } \sqrt[3]{n}$$

$$\lim_{n \rightarrow \infty} \frac{\log^2(n) \xrightarrow{n \rightarrow \infty} \infty}{\sqrt[3]{n} \xrightarrow{n \rightarrow \infty} \infty}$$

beides ableiten (L'Hôpital)

$$= \lim_{n \rightarrow \infty} \frac{2 \log(n) \cdot \frac{1}{n}}{\frac{1}{3} n^{-2/3}}$$

vereinfachen

$$= \lim_{n \rightarrow \infty} \frac{6 \log(n) \cdot n^{2/3}}{n} = \lim_{n \rightarrow \infty} \frac{6 \log(n)}{n^{1/3}}$$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

- kontante Faktoren weglassen nicht im Exponenten!

- Plots können helfen nicht immer!

- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10}n^2 \text{ oder } \frac{1}{10^{10}}n^3$$

$$2^{3n} \text{ oder } 2^{2n}$$

$$\log^2(n) \text{ oder } \sqrt[3]{n}$$

$$\lim_{n \rightarrow \infty} \frac{\log^2(n) \xrightarrow{n \rightarrow \infty} \infty}{\sqrt[3]{n} \xrightarrow{n \rightarrow \infty} \infty}$$

beides ableiten (L'Hôpital)

$$= \lim_{n \rightarrow \infty} \frac{2 \log(n) \cdot \frac{1}{n}}{\frac{1}{3} n^{-2/3}}$$

vereinfachen

$$= \lim_{n \rightarrow \infty} \frac{6 \log(n) \cdot n^{2/3}}{n} = \lim_{n \rightarrow \infty} \frac{6 \log(n) \xrightarrow{n \rightarrow \infty} \infty}{n^{1/3} \xrightarrow{n \rightarrow \infty} \infty}$$

beides ableiten (L'Hôpital)

$$= \lim_{n \rightarrow \infty} \frac{6 \cdot \frac{1}{n}}{\frac{1}{3} n^{-1/3}}$$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- kontante Faktoren weglassen **nicht im Exponenten!**

- Plots können helfen **nicht immer!**

- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10} n^2 \text{ oder } \frac{1}{10^{10}} n^3$$

$$2^{3n} \text{ oder } 2^{2n}$$

$$\log^2(n) \text{ oder } \sqrt[3]{n}$$

$$\begin{aligned} & \lim_{n \rightarrow \infty} \frac{\log^2(n) \xrightarrow{n \rightarrow \infty} \infty}{\sqrt[3]{n} \xrightarrow{n \rightarrow \infty} \infty} \\ &= \lim_{n \rightarrow \infty} \frac{2 \log(n) \cdot \frac{1}{n}}{\frac{1}{3} n^{-2/3}} \quad \begin{array}{l} \text{beides ableiten} \\ \text{(L'Hôpital)} \end{array} \\ &= \lim_{n \rightarrow \infty} \frac{6 \log(n) \cdot n^{2/3}}{n} = \lim_{n \rightarrow \infty} \frac{6 \log(n) \xrightarrow{n \rightarrow \infty} \infty}{n^{1/3} \xrightarrow{n \rightarrow \infty} \infty} \quad \begin{array}{l} \text{vereinfachen} \end{array} \\ &= \lim_{n \rightarrow \infty} \frac{6 \cdot \frac{1}{n}}{\frac{1}{3} n^{-1/3}} \quad \begin{array}{l} \text{beides ableiten} \\ \text{(L'Hôpital)} \end{array} \\ &= \lim_{n \rightarrow \infty} \frac{18 \cdot n^{1/3}}{n} = \lim_{n \rightarrow \infty} \frac{18}{n^{2/3}} \quad \begin{array}{l} \text{vereinfachen} \end{array} \end{aligned}$$

$$\blacksquare f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$$

■ kontante Faktoren weglassen **nicht im Exponenten!**

■ Plots können helfen **nicht immer!**

$$\blacksquare f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10} n^2 \text{ oder } \frac{1}{10^{10}} n^3$$

$$2^{3n} \text{ oder } 2^{2n}$$

$$\log^2(n) \text{ oder } \sqrt[3]{n}$$

$$\begin{aligned}
 & \lim_{n \rightarrow \infty} \frac{\log^2(n) \xrightarrow{n \rightarrow \infty} \infty}{\sqrt[3]{n} \xrightarrow{n \rightarrow \infty} \infty} \\
 = & \lim_{n \rightarrow \infty} \frac{2 \log(n) \cdot \frac{1}{n}}{\frac{1}{3} n^{-2/3}} \quad \begin{array}{l} \text{beides ableiten} \\ \text{(L'Hôpital)} \end{array} \\
 = & \lim_{n \rightarrow \infty} \frac{6 \log(n) \cdot n^{2/3}}{n} = \lim_{n \rightarrow \infty} \frac{6 \log(n) \xrightarrow{n \rightarrow \infty} \infty}{n^{1/3} \xrightarrow{n \rightarrow \infty} \infty} \quad \begin{array}{l} \text{vereinfachen} \end{array} \\
 = & \lim_{n \rightarrow \infty} \frac{6 \cdot \frac{1}{n}}{\frac{1}{3} n^{-1/3}} \quad \begin{array}{l} \text{beides ableiten} \\ \text{(L'Hôpital)} \end{array} \\
 = & \lim_{n \rightarrow \infty} \frac{18 \cdot n^{1/3}}{n} = \lim_{n \rightarrow \infty} \frac{18}{n^{2/3}} \quad \begin{array}{l} \text{vereinfachen} \end{array} \\
 = & 0
 \end{aligned}$$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$

- kontante Faktoren weglassen **nicht im Exponenten!**

- Plots können helfen **nicht immer!**

- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$



Asymptotik – Beispiele

Was wächst schneller?

$$5n^2 \text{ oder } 2n^3$$

$$10^{10}n^2 \text{ oder } \frac{1}{10^{10}}n^3$$

$$2^{3n} \text{ oder } 2^{2n}$$

$$\log^2(n) \text{ oder } \sqrt[3]{n}$$

$$\begin{aligned} & \lim_{n \rightarrow \infty} \frac{\log^2(n) \xrightarrow{n \rightarrow \infty} \infty}{\sqrt[3]{n} \xrightarrow{n \rightarrow \infty} \infty} \\ &= \lim_{n \rightarrow \infty} \frac{2 \log(n) \cdot \frac{1}{n}}{\frac{1}{3} n^{-2/3}} \quad \begin{array}{l} \text{beides ableiten} \\ \text{(L'Hôpital)} \end{array} \\ &= \lim_{n \rightarrow \infty} \frac{6 \log(n) \cdot n^{2/3}}{n} = \lim_{n \rightarrow \infty} \frac{6 \log(n) \xrightarrow{n \rightarrow \infty} \infty}{n^{1/3} \xrightarrow{n \rightarrow \infty} \infty} \quad \begin{array}{l} \text{vereinfachen} \end{array} \\ &= \lim_{n \rightarrow \infty} \frac{6 \cdot \frac{1}{n}}{\frac{1}{3} n^{-1/3}} \quad \begin{array}{l} \text{beides ableiten} \\ \text{(L'Hôpital)} \end{array} \\ &= \lim_{n \rightarrow \infty} \frac{18 \cdot n^{1/3}}{n} = \lim_{n \rightarrow \infty} \frac{18}{n^{2/3}} \quad \begin{array}{l} \text{vereinfachen} \end{array} \\ &= 0 \end{aligned}$$

$$\blacksquare f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0 : f(n) \leq c \cdot g(n)$$

\blacksquare konstante Faktoren weglassen nicht im Exponenten!

\blacksquare Plots können helfen nicht immer!

$$\blacksquare f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

\blacksquare Logarithmen wachsen langsamer als Polynome



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$

$\log(\sqrt{n})$ oder $\sqrt{\log(n)}$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$
- kontante Faktoren weglassen **nicht im Exponenten!**
- Plots können helfen **nicht immer!**
- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- Logarithmen wachsen langsamer als Polynome



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$

$\log(\sqrt{n})$ oder $\sqrt{\log(n)}$

Tipp: $\sqrt{n} = n^{0.5}$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen

nicht im Exponenten!

■ Plots können helfen

nicht immer!

■ $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

■ Logarithmen wachsen langsamer als Polynome

Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$

$\log(\sqrt{n})$ oder $\sqrt{\log(n)}$

Tipp: $\sqrt{n} = n^{0.5}$

$$\log(\sqrt{n}) = \log(n^{0.5})$$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$
- kontante Faktoren weglassen **nicht im Exponenten!**
- Plots können helfen **nicht immer!**
- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- Logarithmen wachsen langsamer als Polynome

Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$

$\log(\sqrt{n})$ oder $\sqrt{\log(n)}$

Tipp: $\sqrt{n} = n^{0.5}$

$$\log(a^x) = x \cdot \log(a)$$

$$\log(\sqrt{n}) = \log(n^{0.5})$$

$$= 0.5 \cdot \log(n)$$

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$
- kontante Faktoren weglassen **nicht im Exponenten!**
- Plots können helfen **nicht immer!**
- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- Logarithmen wachsen langsamer als Polynome



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$

$\log(\sqrt{n})$ oder $\sqrt{\log(n)}$

Tipp: $\sqrt{n} = n^{0.5}$

$$\log(a^x) = x \cdot \log(a)$$

$$\log(\sqrt{n}) = \log(n^{0.5})$$

$$= 0.5 \cdot \log(n)$$

$$\in \omega(\sqrt{\log(n)})$$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen **nicht im Exponenten!**

■ Plots können helfen **nicht immer!**

■ $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

■ Logarithmen wachsen langsamer als Polynome



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$

$\log(\sqrt{n})$ oder $\sqrt{\log(n)}$

Tipp: $\sqrt{n} = n^{0.5}$

$$\log(a^x) = x \cdot \log(a)$$

$$\log(\sqrt{n}) = \log(n^{0.5})$$

$$= 0.5 \cdot \log(n)$$

$$\in \omega(\sqrt{\log(n)})$$

■ $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$

■ kontante Faktoren weglassen **nicht im Exponenten!**

■ Plots können helfen **nicht immer!**

■ $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$

■ Logarithmen wachsen langsamer als Polynome

■ Logarithmengesetze sind hilfreich



Asymptotik – Beispiele

Was wächst schneller?

$5n^2$ oder $2n^3$

$10^{10}n^2$ oder $\frac{1}{10^{10}}n^3$

2^{3n} oder 2^{2n}

$\log^2(n)$ oder $\sqrt[3]{n}$

$\log(\sqrt{n})$ oder $\sqrt{\log(n)}$

Welche Laufzeiten sind gut in der Praxis?

- $f(n) \in O(g(n)) \iff \exists c \exists n_0 \forall n > n_0: f(n) \leq c \cdot g(n)$
- kontante Faktoren weglassen **nicht im Exponenten!**
- Plots können helfen **nicht immer!**
- $f(n) \in o(g(n)) \Rightarrow \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$
- Logarithmen wachsen langsamer als Polynome
- Logarithmengesetze sind hilfreich

Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code

Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code

wrongAlgo(*input*)

- richtige Abstraktionsebene

| **return** correct output



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.iti.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

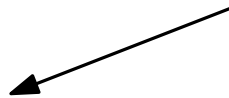
Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung

wollen wir **nie** sehen



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung

wollen wir **nie** sehen „Gib Pseudocode an“

Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung

wollen wir **nie** sehen

„Gib Pseudocode an“

- enthält Implementierungsdetails

Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung

wollen wir **nie** sehen

„Gib Pseudocode an“

- enthält Implementierungsdetails
- fehleranfällig



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

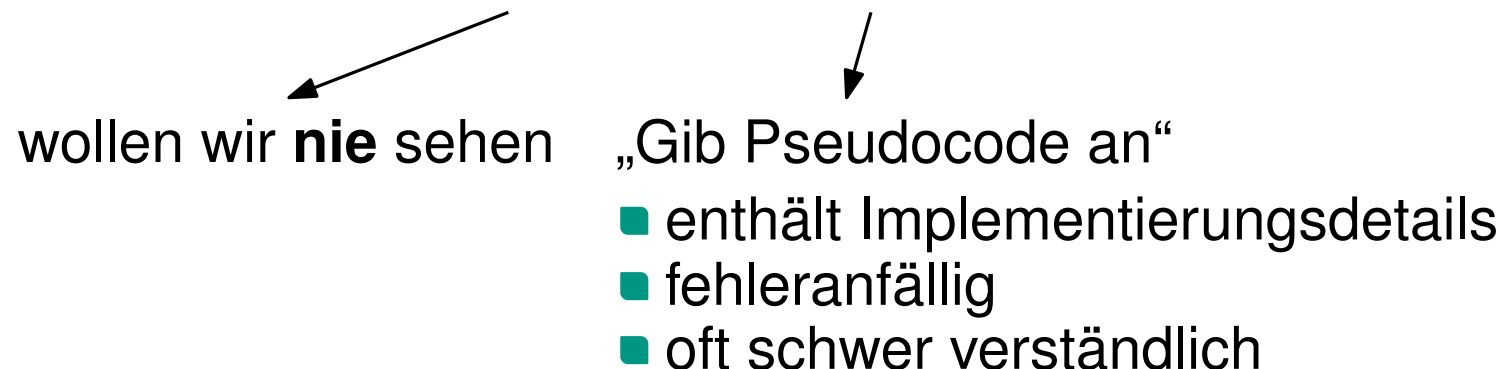
Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.iti.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung

wollen wir **nie** sehen

„Gib Pseudocode an“

- enthält Implementierungsdetails
- fehleranfällig
- oft schwer verständlich



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

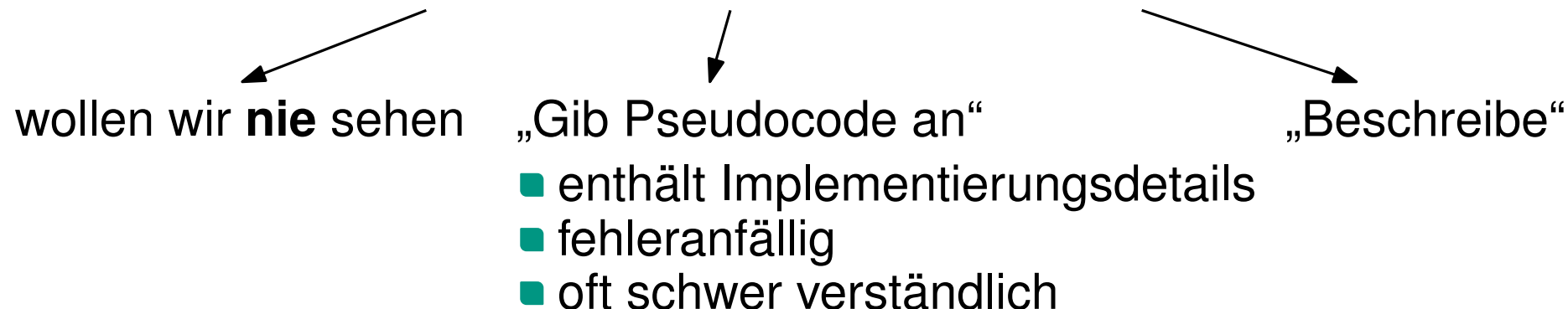
Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

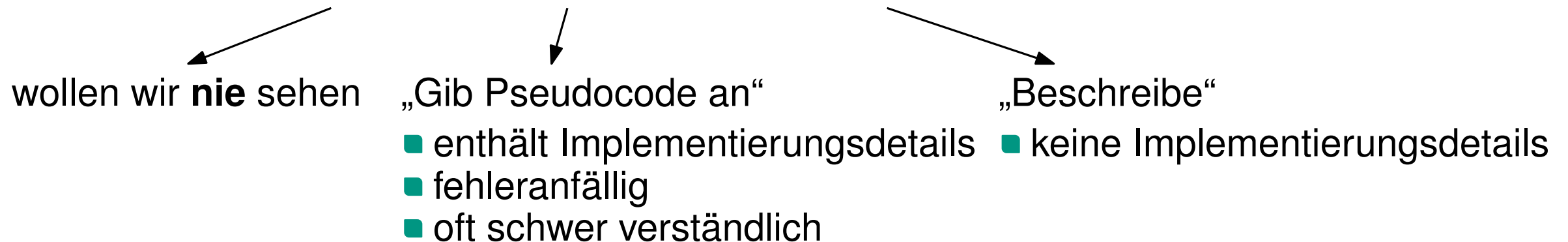
Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung



Pseudocode

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

Algorithmusidee Wir iterieren alle Ziffern und ...

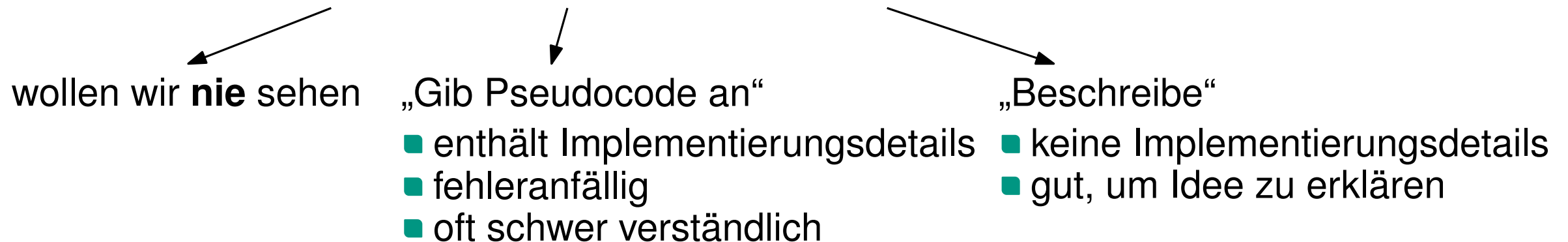
Pseudocode **for** i in $\{0, 1, \dots, n-1\}$ **do** ... <https://scale.itl.kit.edu/teaching/2024ss/algo1/start>

Code `for (int i = 0; i < n; ++i) { ... }`

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

~~wrongAlgo(input)~~
~~| return correct output~~

Code \neq Pseudocode \neq Beschreibung



Pseudocode & Rekursion

mysteryFunc(I, d, k)

if $k - d == 1$ **return** $I[d]$

$t := \lfloor (d + k) / 2 \rfloor$

$a :=$ **mysteryFunc**(I, d, t)

$b :=$ **mysteryFunc**(I, t, k)

if $a < b$ **then**

return a

else

return b

Pseudocode & Rekursion

mysteryFunc(I, d, k)

```
if  $k - d == 1$  return  $I[d]$   
 $t := \lfloor (d + k) / 2 \rfloor$   
 $a := \text{mysteryFunc}(I, d, t)$   
 $b := \text{mysteryFunc}(I, t, k)$   
if  $a < b$  then  
    return  $a$   
else  
    return  $b$ 
```

Was tut dieser Pseudocode?

Pseudocode & Rekursion

mysteryFunc(I, d, k)

```
if  $k - d == 1$  return  $I[d]$   
 $t := \lfloor (d + k) / 2 \rfloor$   
 $a := \text{mysteryFunc}(I, d, t)$   
 $b := \text{mysteryFunc}(I, t, k)$   
if  $a < b$  then  
    return  $a$   
else  
    return  $b$ 
```

Was tut dieser Pseudocode?

- Typen spezifizieren



Pseudocode & Rekursion

mysteryFunc(I : Array, d : \mathbb{N} , k : \mathbb{N})

```
if  $k - d == 1$  return  $I[d]$ 
```

```
 $t := \lfloor (d + k) / 2 \rfloor$ 
```

```
 $a := \text{mysteryFunc}(I, d, t)$ 
```

```
 $b := \text{mysteryFunc}(I, t, k)$ 
```

```
if  $a < b$  then
```

```
    return  $a$ 
```

```
else
```

```
    return  $b$ 
```

Was tut dieser Pseudocode?

- Typen spezifizieren

Pseudocode & Rekursion

mysteryFunc(I : Array, d : \mathbb{N} , k : \mathbb{N})

```
if  $k - d == 1$  return  $I[d]$ 
 $t := \lfloor (d + k) / 2 \rfloor$ 
 $a := \text{mysteryFunc}(I, d, t)$ 
 $b := \text{mysteryFunc}(I, t, k)$ 
if  $a < b$  then
    return  $a$ 
else
    return  $b$ 
```

Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen

Pseudocode & Rekursion

mysteryFunc(A : Array, $start$: \mathbb{N} , end : \mathbb{N})

```
if end – start == 1 return A[start]
mid :=  $\lfloor (start + end) / 2 \rfloor$ 
left := mysteryFunc( $A$ , start, mid)
right := mysteryFunc( $A$ , mid, end)
if left < right then
    return left
else
    return right
```

Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen



Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

```
if end – start == 1 return A[start]
mid :=  $\lfloor (start + end) / 2 \rfloor$ 
left := mysteryFunc(A, start, mid)
right := mysteryFunc(A, mid, end)
if left < right then
    return left
else
    return right
```

Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden



Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

```

if end – start == 1 return A[start]
mid :=  $\lfloor (\text{start} + \text{end}) / 2 \rfloor$ 
left := mysteryFunc(A, start, mid)
right := mysteryFunc(A, mid, end)
if left < right then
  return left
else
  return right

```

min(left, right)

Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden



Pseudocode & Rekursion

mysteryFunc(A : Array, start: \mathbb{N} , end: \mathbb{N})

```
if end – start == 1 return A[start]
mid :=  $\lfloor (start + end) / 2 \rfloor$ 
left := mysteryFunc( $A$ , start, mid)
right := mysteryFunc( $A$ , mid, end)
return min(left, right)
```

Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden



Pseudocode & Rekursion

mysteryFunc(A : Array, start: \mathbb{N} , end: \mathbb{N})

```
if end – start == 1 return A[start]
mid :=  $\lfloor (\text{start} + \text{end}) / 2 \rfloor$ 
left := mysteryFunc( $A$ , start, mid)
right := mysteryFunc( $A$ , mid, end)
return min(left, right)
```

mysteryFunc(A , 1, 4)

32	10	23	5	4	13
----	----	----	---	---	----

Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

if end – start == 1 **return** A[start]

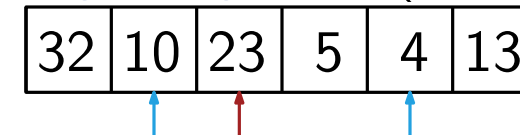
mid := $\lfloor (\text{start} + \text{end}) / 2 \rfloor$

left := **mysteryFunc**(A, start, mid)

right := **mysteryFunc**(A, mid, end)

return min(left, right)

mysteryFunc(A, 1, 4)



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

if end – start == 1 **return** A[start]

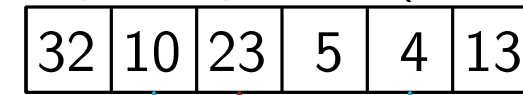
mid := $\lfloor (\text{start} + \text{end}) / 2 \rfloor$

left := **mysteryFunc**(A, start, mid)

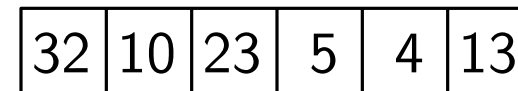
right := **mysteryFunc**(A, mid, end)

return min(left, right)

mysteryFunc(A, 1, 4)



mysteryFunc(A, 1, 2)



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

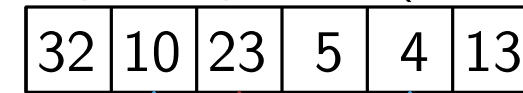
Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

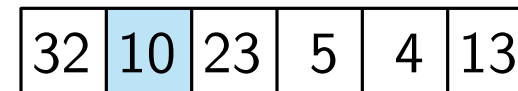
```

if end – start == 1 return A[start]
mid :=  $\lfloor (\text{start} + \text{end}) / 2 \rfloor$ 
left := mysteryFunc(A, start, mid)
right := mysteryFunc(A, mid, end)
return min(left, right)
  
```

mysteryFunc(A, 1, 4)



mysteryFunc(A, 1, 2)



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

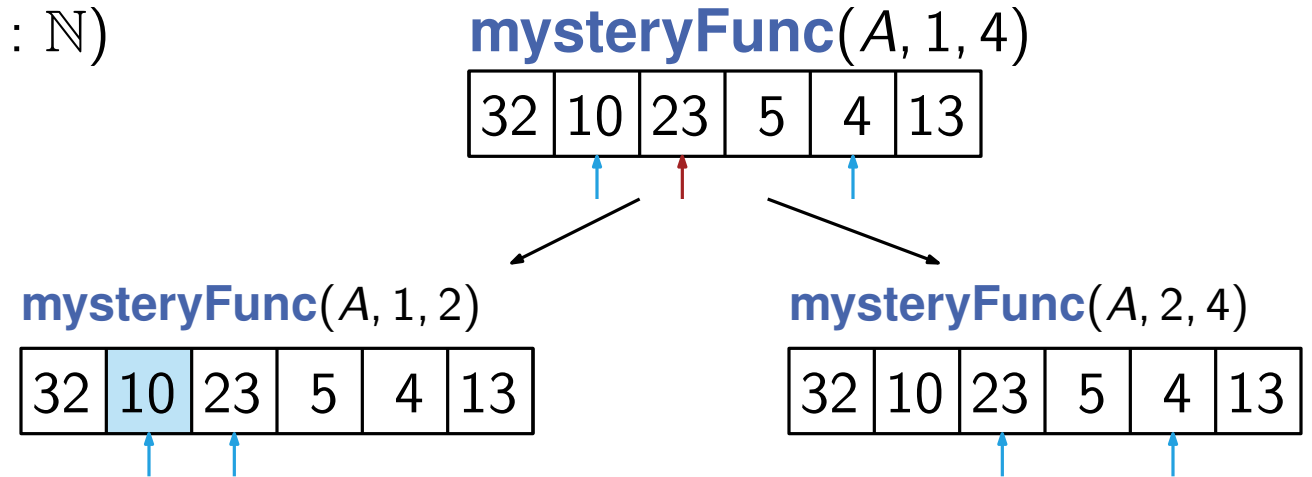
if end – start == 1 **return** A[start]

mid := $\lfloor (\text{start} + \text{end}) / 2 \rfloor$

left := **mysteryFunc**(A, start, mid)

right := **mysteryFunc**(A, mid, end)

return min(left, right)



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

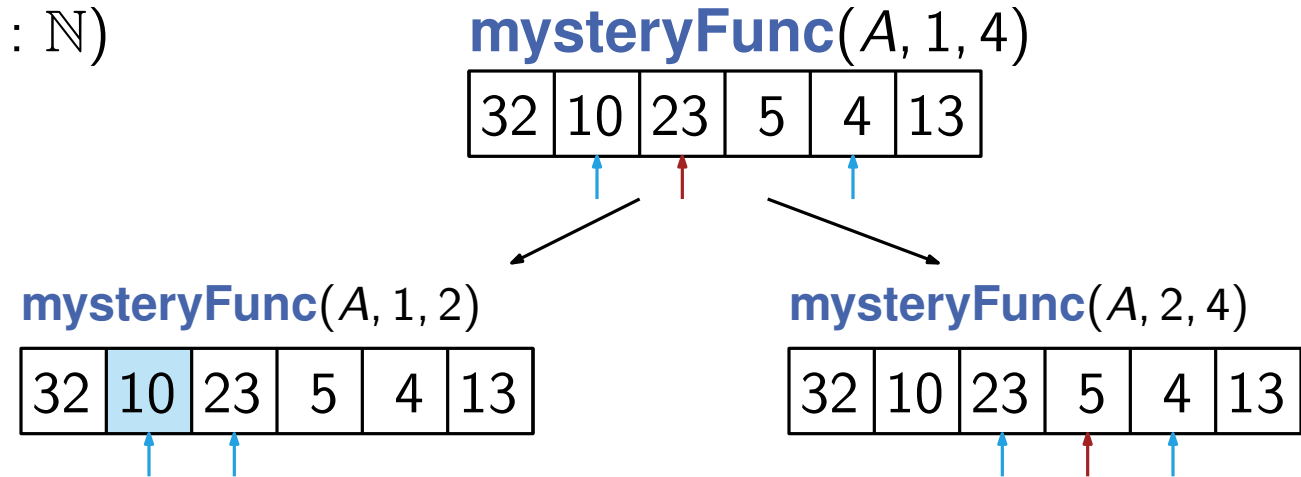
if end – start == 1 **return** A[start]

mid := $\lfloor (\text{start} + \text{end}) / 2 \rfloor$

left := **mysteryFunc**(A, start, mid)

right := **mysteryFunc**(A, mid, end)

return min(left, right)



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

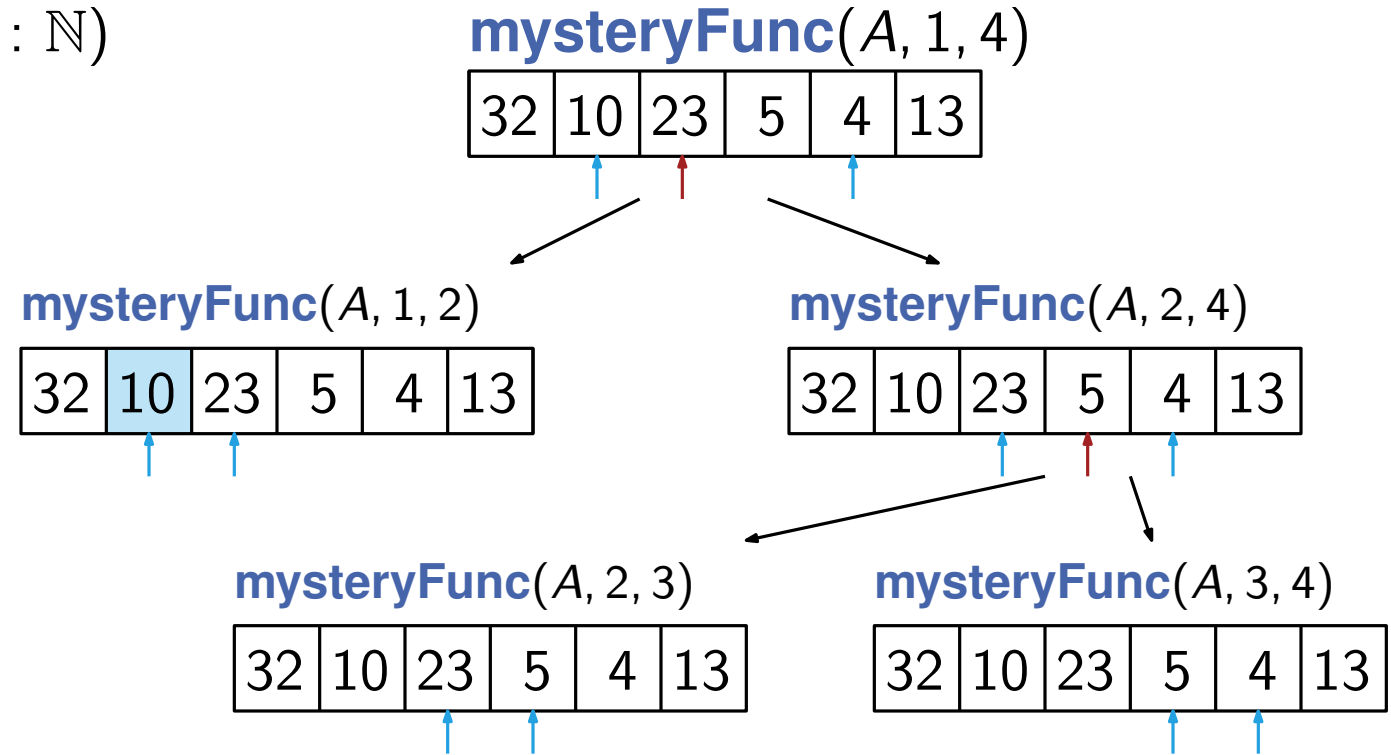
if end – start == 1 **return** A[start]

mid := $\lfloor (\text{start} + \text{end}) / 2 \rfloor$

left := **mysteryFunc**(A, start, mid)

right := **mysteryFunc**(A, mid, end)

return min(left, right)



Was tut dieser Pseudocode?

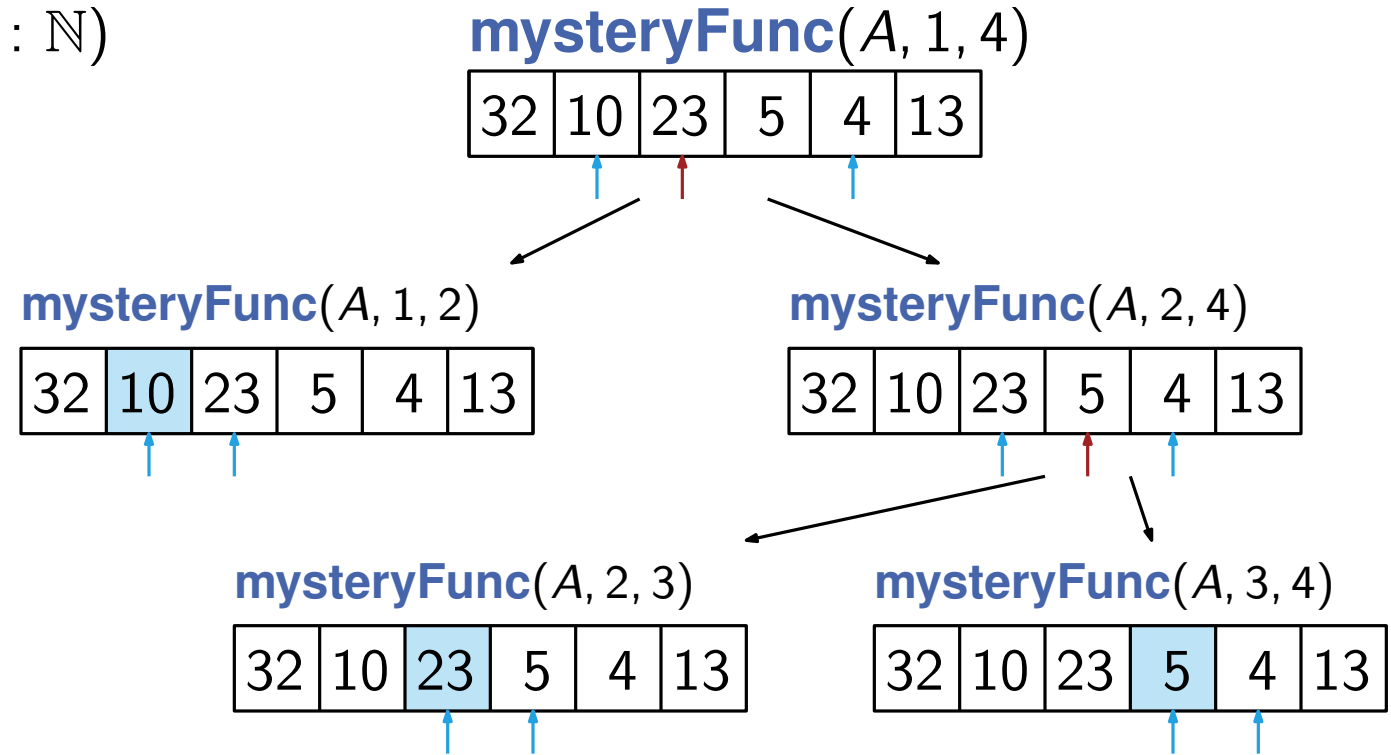
- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

```

if end – start == 1 return A[start]
mid :=  $\lfloor (\text{start} + \text{end}) / 2 \rfloor$ 
left := mysteryFunc(A, start, mid)
right := mysteryFunc(A, mid, end)
return min(left, right)
  
```



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

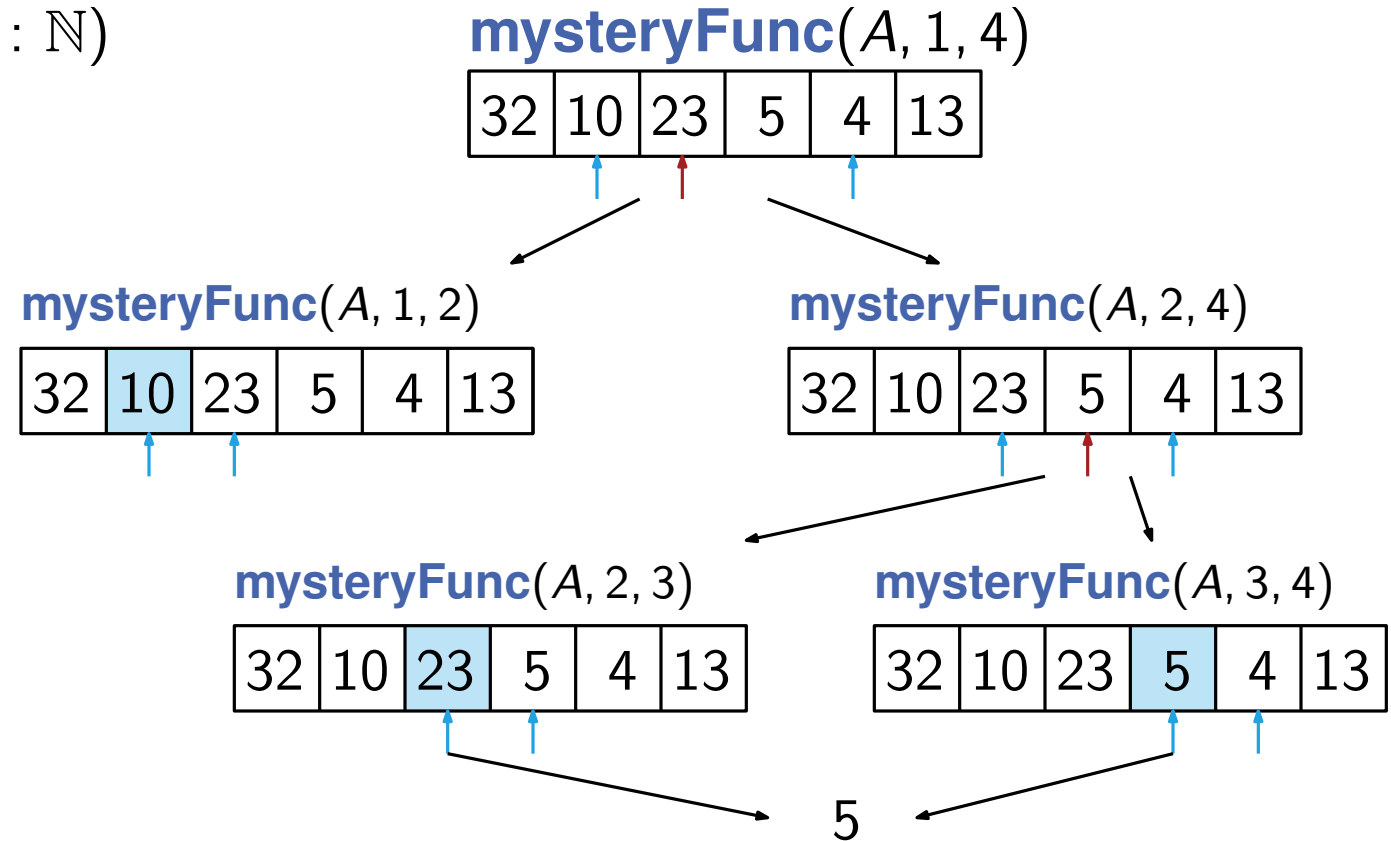
if end – start == 1 **return** A[start]

mid := $\lfloor (\text{start} + \text{end}) / 2 \rfloor$

left := **mysteryFunc**(A, start, mid)

right := **mysteryFunc**(A, mid, end)

return min(left, right)



Was tut dieser Pseudocode?

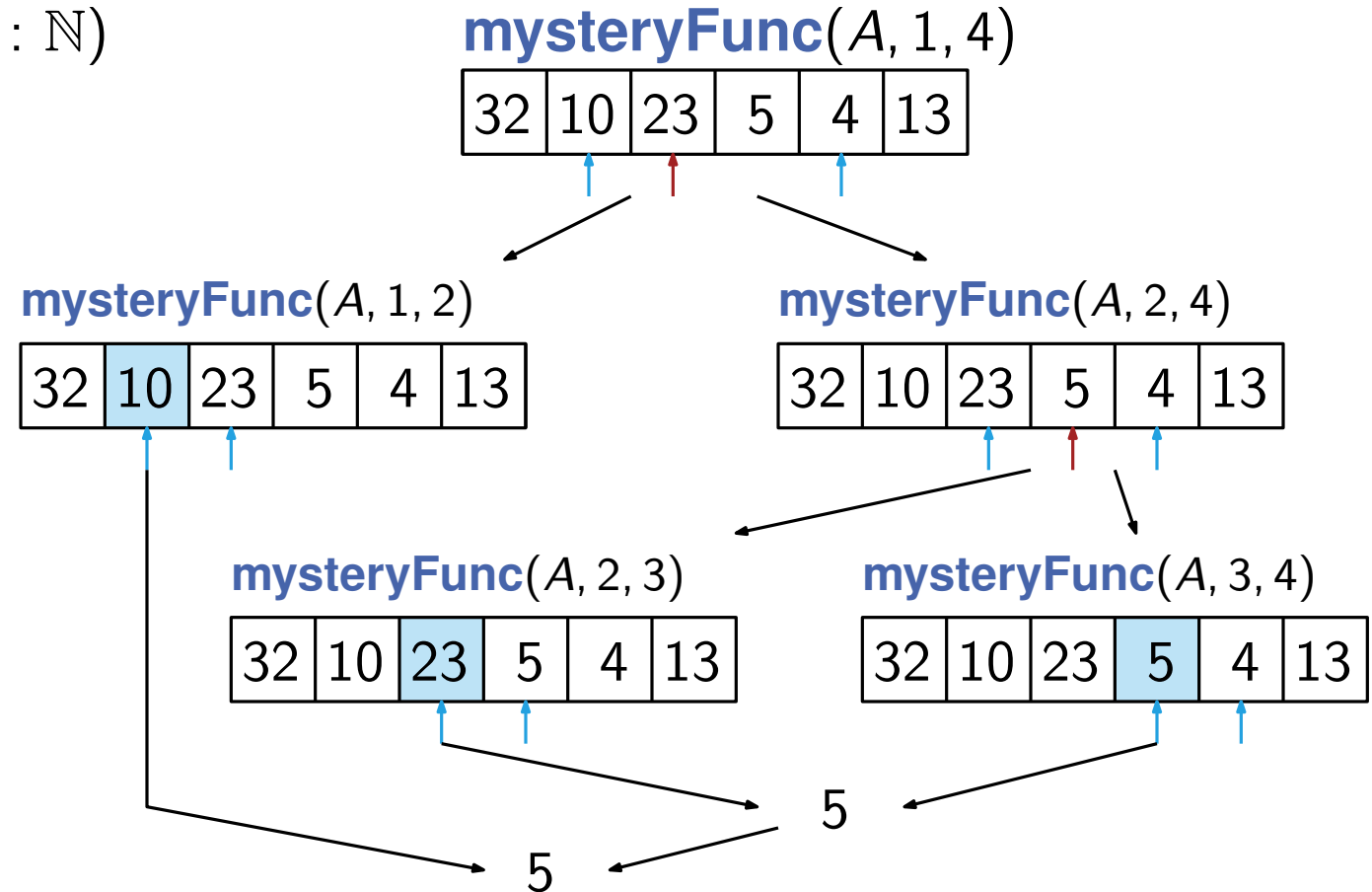
- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

mysteryFunc(A: Array, start: \mathbb{N} , end: \mathbb{N})

```

if end – start == 1 return A[start]
mid :=  $\lfloor (start + end) / 2 \rfloor$ 
left := mysteryFunc(A, start, mid)
right := mysteryFunc(A, mid, end)
return min(left, right)
  
```



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

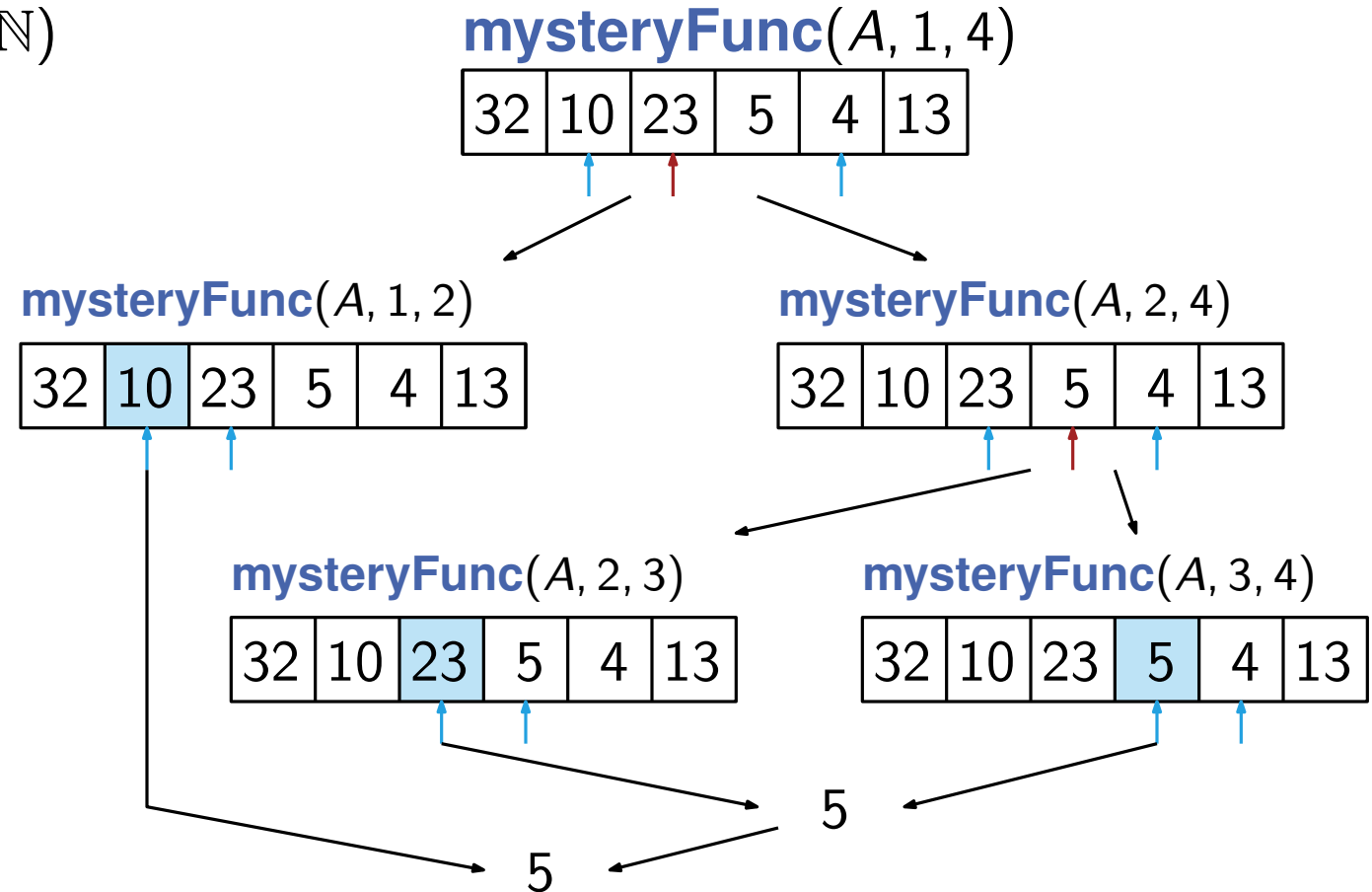
if end – start == 1 **return** A [start]

mid := $\lfloor (start + end) / 2 \rfloor$

left := **minInterval**(A , start, mid)

right := **minInterval**(A , mid, end)

return min(left, right)



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

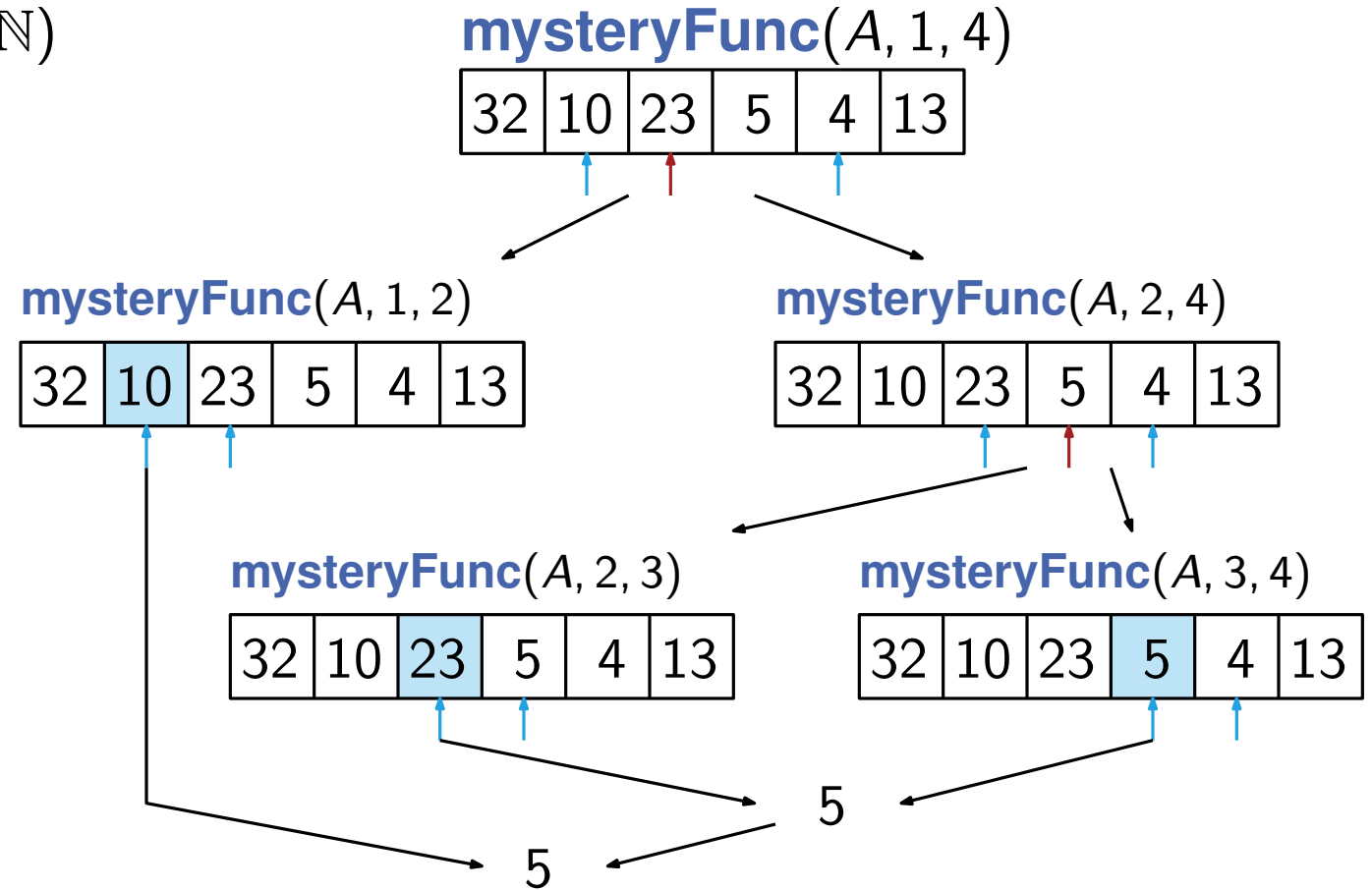
if end – start == 1 **return** A [start]

mid := $\lfloor (start + end) / 2 \rfloor$

left := **minInterval**(A , start, mid)

right := **minInterval**(A , mid, end)

return min(left, right)



\Rightarrow bestimmt Minimum im Intervall $[start, end)$ in A

Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

```
if end – start == 1 return A[start]
mid :=  $\lfloor (\text{start} + \text{end}) / 2 \rfloor$ 
left := minInterval( $A$ , start, mid)
right := minInterval( $A$ , mid, end)
return min(left, right)
```

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

```
if end – start == 1 return A[start]
mid := ⌊(start + end)/2⌋
left := minInterval(A, start, mid)
right := minInterval(A, mid, end)
return min(left, right)
```

Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

if end – start == 1 **return** $A[\text{start}]$

mid := $\lfloor (\text{start} + \text{end}) / 2 \rfloor$

left := **minInterval**(A , start, mid)

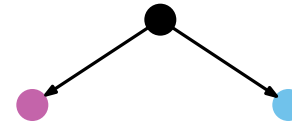
right := **minInterval**(A , mid, end)

return min(left, right)

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

if end – start == 1 **return** $A[\text{start}]$

mid := $\lfloor (\text{start} + \text{end}) / 2 \rfloor$

left := **minInterval**(A , start, mid)

right := **minInterval**(A , mid, end)

return min(left, right)

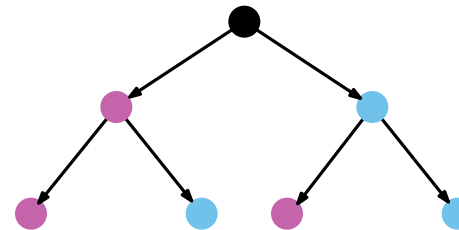
Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

if end – start == 1 **return** $A[\text{start}]$

mid := $\lfloor (\text{start} + \text{end}) / 2 \rfloor$

left := **minInterval**(A , start, mid)

right := **minInterval**(A , mid, end)

return min(left, right)

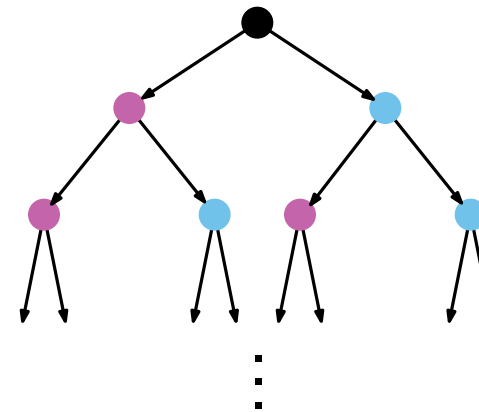
Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

```

if end – start == 1 return A[start]
mid := ⌊(start + end)/2⌋
left := minInterval(A, start, mid)
right := minInterval(A, mid, end)
return min(left, right)
  
```

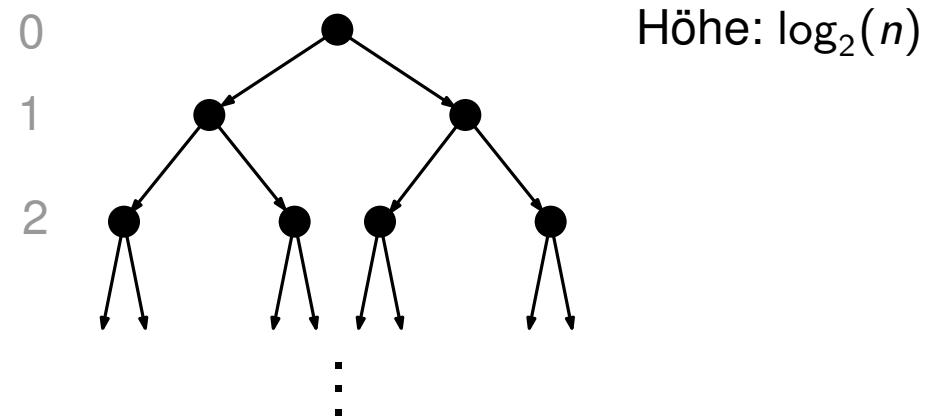
Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

```

if end – start == 1 return A[start]
mid := ⌊(start + end)/2⌋
left := minInterval(A, start, mid)
right := minInterval(A, mid, end)
return min(left, right)
  
```

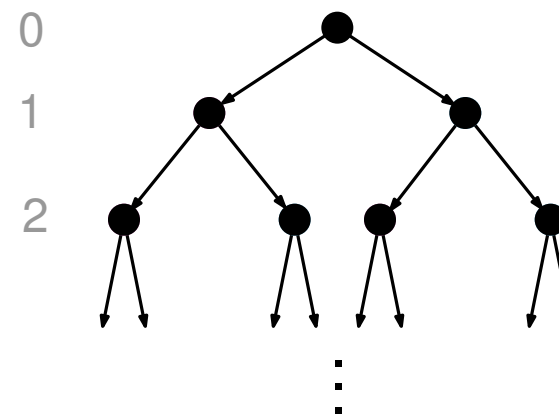
Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Höhe: $\log_2(n)$

Knoten in Lage i : 2^i

Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

```

if end – start == 1 return A[start]
mid :=  $\lfloor (start + end) / 2 \rfloor$ 
left := minInterval(A, start, mid)
right := minInterval(A, mid, end)
return min(left, right)
  
```

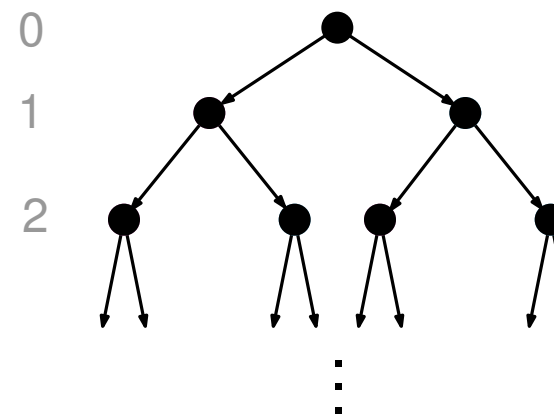
Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Höhe: $\log_2(n)$

Knoten in Lage i : 2^i

Kosten pro Knoten
in Lage i : $\Theta(1)$

Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

```

if end – start == 1 return A[start]
mid := ⌊(start + end)/2⌋
left := minInterval(A, start, mid)
right := minInterval(A, mid, end)
return min(left, right)
  
```

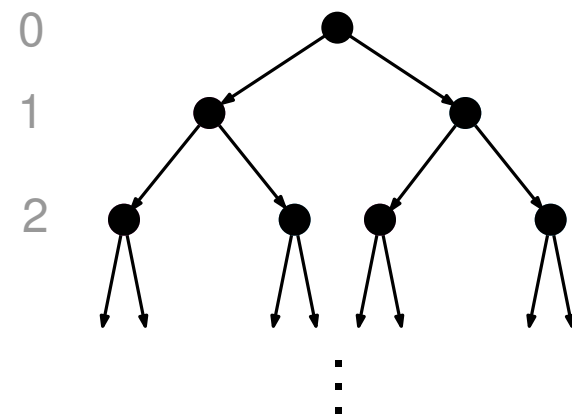
Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Höhe: $\log_2(n)$

Knoten in Lage i : 2^i

Kosten pro Knoten
in Lage i : $\Theta(1)$

Laufzeit: $\sum_{i=0}^{\log(n)} 2^i \cdot 1$



Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

```

if end – start == 1 return A[start]
mid := ⌊(start + end)/2⌋
left := minInterval(A, start, mid)
right := minInterval(A, mid, end)
return min(left, right)
  
```

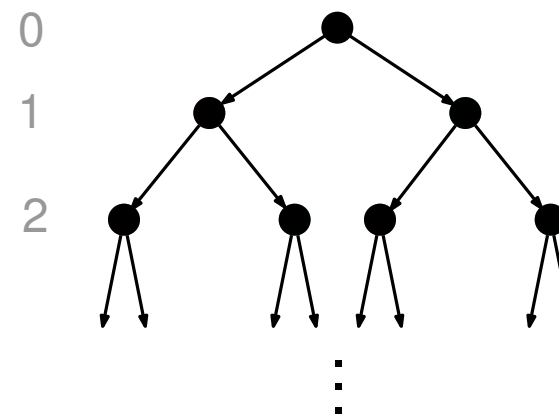
Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Höhe: $\log_2(n)$

Knoten in Lage i : 2^i

Kosten pro Knoten
in Lage i : $\Theta(1)$

$$\text{Laufzeit: } \sum_{i=0}^{\log(n)} 2^i \cdot 1 = 2^{\log(n)+1} - 1$$



Pseudocode & Rekursion

minInterval(A : Array, start: \mathbb{N} , end: \mathbb{N})

```

if end – start == 1 return A[start]
mid := ⌊(start + end)/2⌋
left := minInterval(A, start, mid)
right := minInterval(A, mid, end)
return min(left, right)
  
```

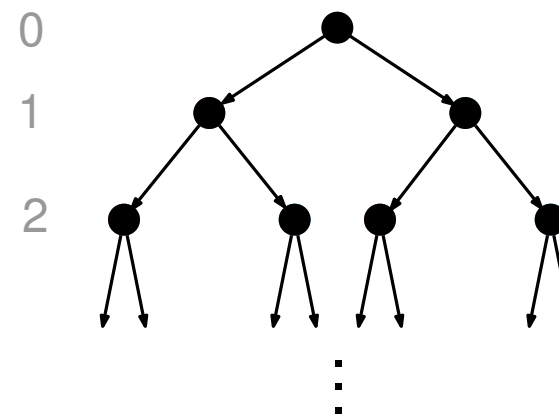
Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

- $O(n)$
- $O(n^2)$
- $O(n \log(n))$
- $O(2^n)$

Rekursionsbaum



Höhe: $\log_2(n)$

Knoten in Lage i : 2^i

Kosten pro Knoten
in Lage i : $\Theta(1)$

$$\text{Laufzeit: } \sum_{i=0}^{\log(n)} 2^i \cdot 1 = 2^{\log(n)+1} - 1 \in \Theta(n)$$



2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$$x = 8$$

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$$x = 8$$

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$$x = 8$$

Beschreibung Algorithmus

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$$x = 8$$

Beschreibung Algorithmus

- beginne Suche unten links



2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$$x = 8$$

Beschreibung Algorithmus

- beginne Suche unten links

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$$x = 8$$

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$:

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$:

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 8$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 14$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 14$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 14$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 14$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 14$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 14$

Wohin jetzt?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 14$

Korrektheit?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

$x = 14$

Korrektheit?

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall
- Induktion

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall
- Induktion
 - falls $> x$:

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall
- Induktion
 - falls $> x$: auch rechts alles $> x$



2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall
- Induktion
 - falls $> x$: auch rechts alles $> x$

2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall
- Induktion
 - falls $> x$: auch rechts alles $> x$
 - falls $< x$:



2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall
- Induktion
 - falls $> x$: auch rechts alles $> x$
 - falls $< x$: auch oben alles $< x$



2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall
- Induktion
 - falls $> x$: auch rechts alles $> x$
 - falls $< x$: auch oben alles $< x$



2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall
- Induktion
 - falls $> x$: auch rechts alles $> x$
 - falls $< x$: auch oben alles $< x$
 - falls $= x$: fertig



2D-Search

Problem: 2D-SEARCH

Eingabe: 2D-Array A der Größe $n \times n$, jede Spalte und jede Zeile in sich sortiert
Zahl x

Ausgabe: Indexpaar (i, j) mit $A[i][j] = x$, falls x in A vorkommt

3	5	6	7	10
4	6	8	9	11
7	10	11	12	14
9	12	13	14	18
12	15	16	17	19

Laufzeit?

- $O(n^2)$
- $O(n)$
- $O(\sqrt{n})$

Beschreibung Algorithmus

- beginne Suche unten links
- immer nur nach oben oder nach rechts
- falls $> x$: nach oben
- falls $< x$: nach rechts
- falls $= x$: Indizes ausgeben
- falls außerhalb des Arrays: abbrechen

Korrektheit

- Invariante: das gesuchte Element befindet sich rechts oben vom aktuellen
- Basisfall
- Induktion
 - falls $> x$: auch rechts alles $> x$
 - falls $< x$: auch oben alles $< x$
 - falls $= x$: fertig



Amortisierte Analyse

Beispiel: Array duplizieren



Amortisierte Analyse

Beispiel: Array duplizieren

duplicateArray(Array A : $[\mathbb{N}, n]$): Array

for i from 0 to $n - 1$ **do**

A .**pushBack**($A[i]$)

return A

Amortisierte Analyse

Beispiel: Array duplizieren

duplicateArray(Array A : $[\mathbb{N}, n]$): Array

```
for  $i$  from 0 to  $n - 1$  do
```

```
    A.pushBack(A[i])
```

```
return A
```

Laufzeit

Amortisierte Analyse

Beispiel: Array duplizieren

duplicateArray(Array A : $[\mathbb{N}, n]$): Array

```
for  $i$  from 0 to  $n - 1$  do  
   $A$ .pushBack( $A[i]$ )           //  $\Theta(n)$  (?)  
return  $A$ 
```

Laufzeit

- **pushBack** im Worst Case $\Theta(n)$

Amortisierte Analyse

Beispiel: Array duplizieren

duplicateArray(Array A : $[\mathbb{N}, n]$): Array

```
for  $i$  from 0 to  $n - 1$  do  
    A.pushBack(A[i])           //  $\Theta(n)$  (?)  
return A
```

Laufzeit

- **pushBack** im Worst Case $\Theta(n)$
- Laufzeit von **duplicateArray** in $\Theta(n^2)$???

Amortisierte Analyse

Beispiel: Array duplizieren

duplicateArray(Array A : $[\mathbb{N}, n]$): Array

```
for  $i$  from 0 to  $n - 1$  do  
    A.pushBack(A[i])           //  $\Theta(n)$  (?)  
return A
```

Laufzeit

- **pushBack** im Worst Case $\Theta(n)$
 - Laufzeit von **duplicateArray** in $\Theta(n^2)$???
- Nein!**

Amortisierte Analyse

Beispiel: Array duplizieren

duplicateArray(Array A : $[\mathbb{N}, n]$): Array

```
for  $i$  from 0 to  $n - 1$  do  
   $A$ .pushBack( $A[i]$ )           //  $\Theta(n)$  (?)  
return  $A$ 
```

Laufzeit

- **pushBack** im Worst Case $\Theta(n)$
- Laufzeit von **duplicateArray** in $\Theta(n^2)$??? **Nein!**
- amortisierte Kosten von **pushBack** in $\Theta(1)$

Amortisierte Analyse

Beispiel: Array duplizieren

duplicateArray(Array A : $[\mathbb{N}, n]$): Array

```
for  $i$  from 0 to  $n - 1$  do
  A.pushBack(A[i])           //  $\Theta(n)$  (?)
return A
```

Laufzeit

- **pushBack** im Worst Case $\Theta(n)$
- Laufzeit von **duplicateArray** in $\Theta(n^2)$??? **Nein!**
- amortisierte Kosten von **pushBack** in $\Theta(1)$
 - ↳ durchschnittliche Kosten einer Op in einer Folge von Ops

Amortisierte Analyse

Beispiel: Array duplizieren

duplicateArray(Array A : $[\mathbb{N}, n]$): Array

```
for  $i$  from 0 to  $n - 1$  do
  A.pushBack(A[i])           //  $\Theta(n)$  (?)
return A
```

Laufzeit

- **pushBack** im Worst Case $\Theta(n)$
 - Laufzeit von **duplicateArray** in $\Theta(n^2)$??? **Nein!**
 - amortisierte Kosten von **pushBack** in $\Theta(1)$
 - ↳ durchschnittliche Kosten einer Op in einer Folge von Ops
- ⇒ **duplicateArray** im Worst Case $\Theta(n)$

Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

Aggregatmethode

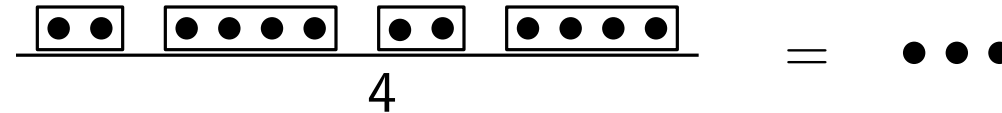
- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

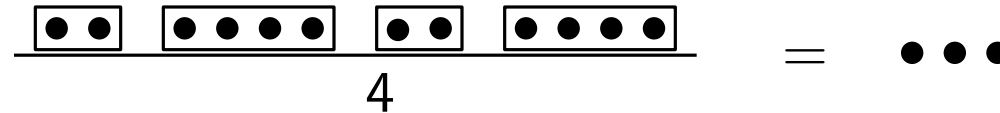


Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen



Charging

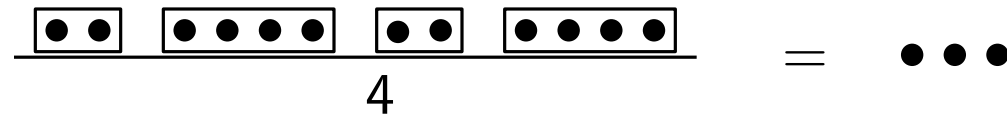
- verteile Kosten-Tokens von teuren zu günstigen Ops

Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

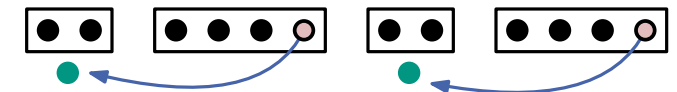
Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen



Charging

- verteile Kosten-Tokens von teuren zu günstigen Ops

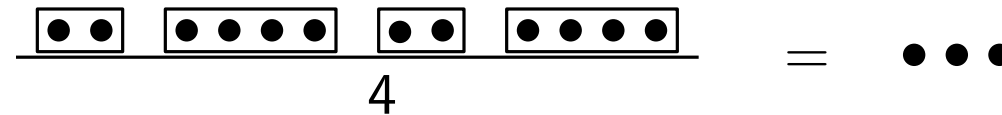


Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

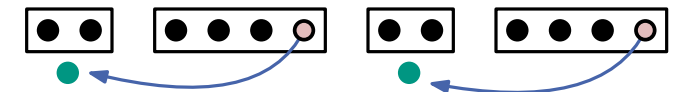
Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen



Charging

- verteile Kosten-Tokens von teuren zu günstigen Ops



Kontomethode

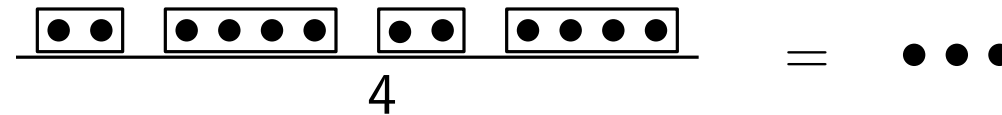
- günstige Op zahlt mehr ins Konto ein, teure Op hebt ab
- Achtung: Kontostand ≥ 0

Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

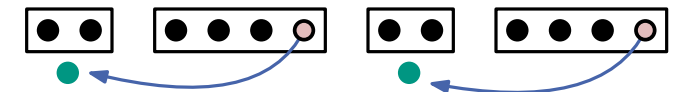
Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen



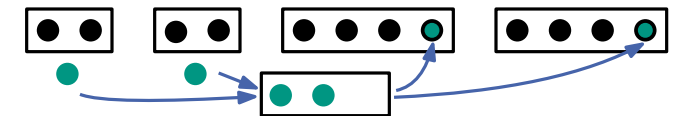
Charging

- verteile Kosten-Tokens von teuren zu günstigen Ops



Kontomethode

- günstige Op zahlt mehr ins Konto ein, teure Op hebt ab
- Achtung: Kontostand ≥ 0

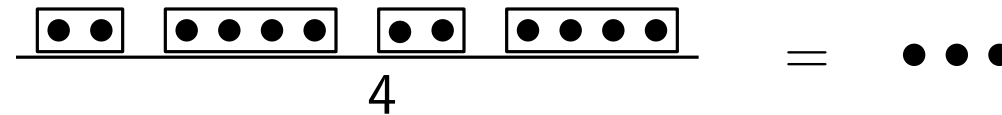


Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

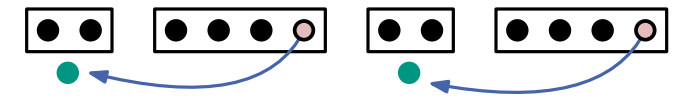
Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen



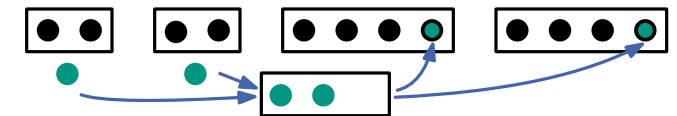
Charging

- verteile Kosten-Tokens von teuren zu günstigen Ops



Kontomethode

- günstige Op zahlt mehr ins Konto ein, teure Op hebt ab
- Achtung: Kontostand ≥ 0



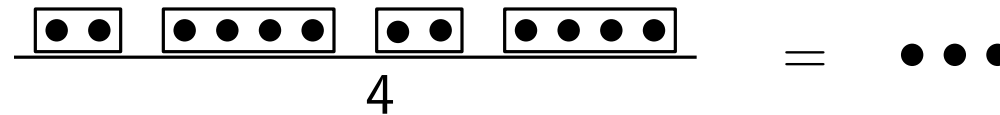
Potentialmethode

Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

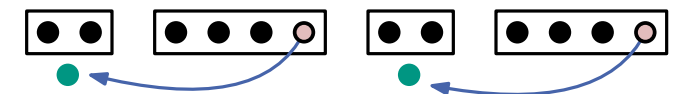
Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen



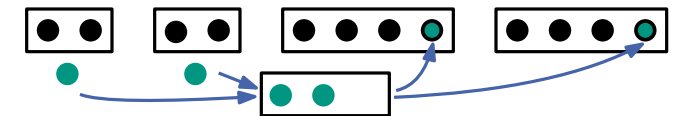
Charging

- verteile Kosten-Tokens von teuren zu günstigen Ops



Kontomethode

- günstige Op zahlt mehr ins Konto ein, teure Op hebt ab
- Achtung: Kontostand ≥ 0



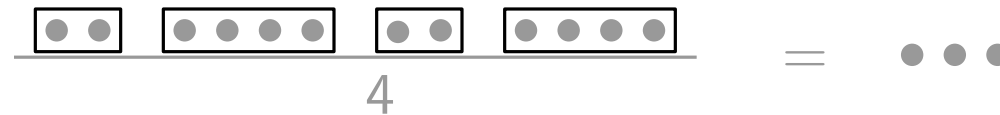
Potentialmethode ← nicht heute

Amortisierte Analyse

- viele günstige Operationen, wenige teure Operationen
- beachte das bei Laufzeitanalyse

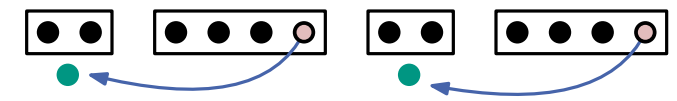
Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen



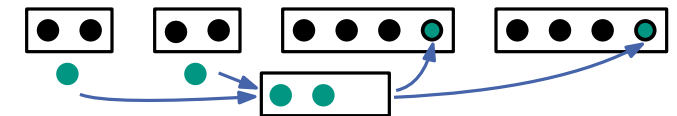
Charging

- verteile Kosten-Tokens von teuren zu günstigen Ops



Kontomethode

- günstige Op zahlt mehr ins Konto ein, teure Op hebt ab
- Achtung: Kontostand ≥ 0

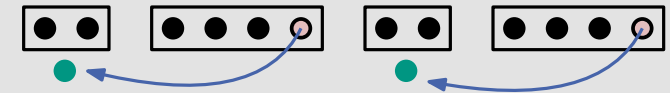


Potentialmethode ← nicht heute

Amortisierte Analyse

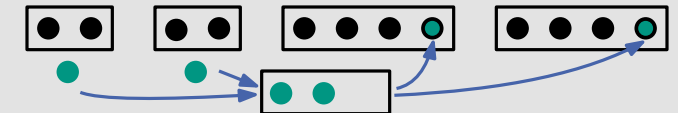
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

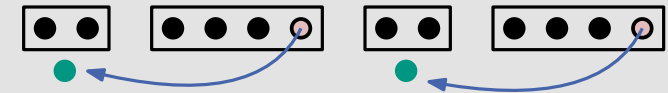


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe

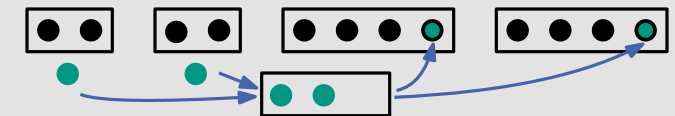
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

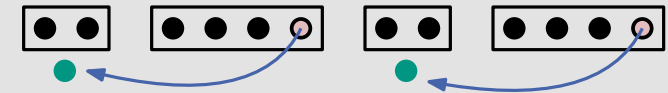


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$

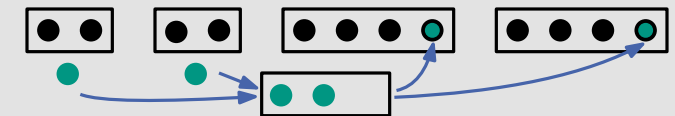
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

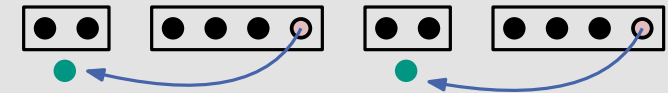


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$

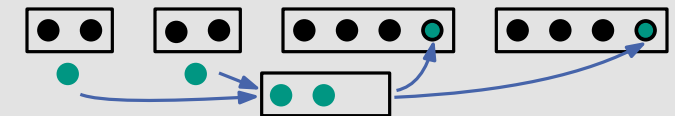
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

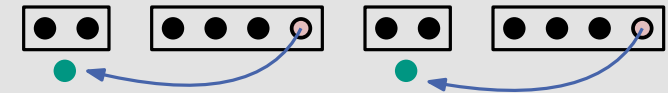


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$

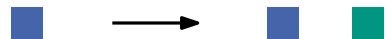
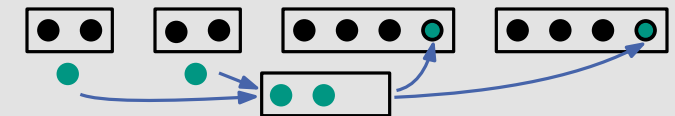
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

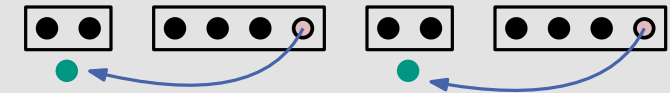


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$

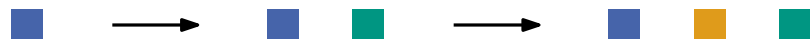
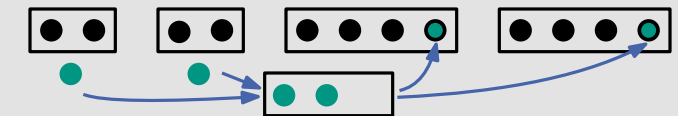
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

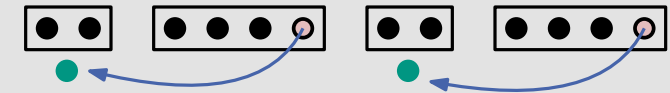


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$

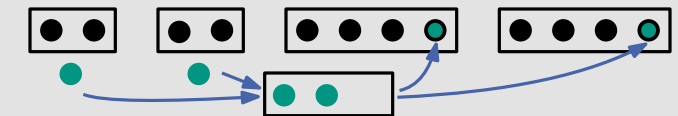
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

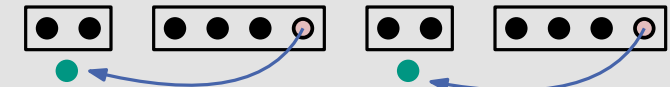


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$

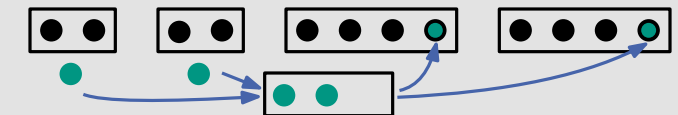
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

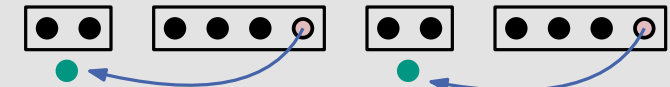


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$

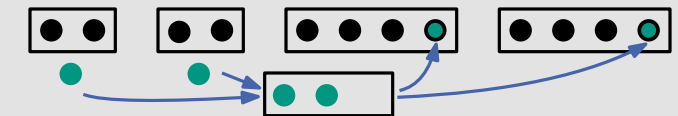
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

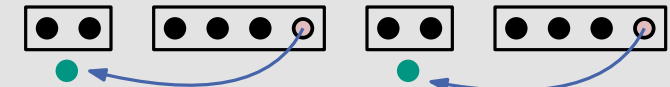


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$

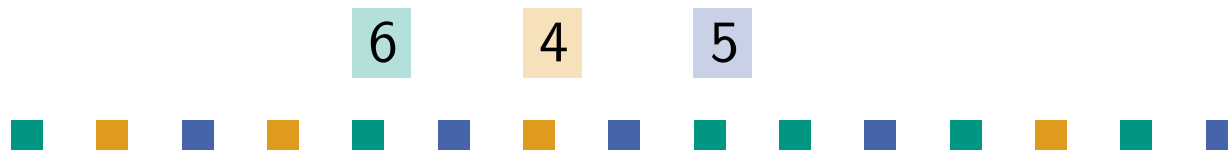
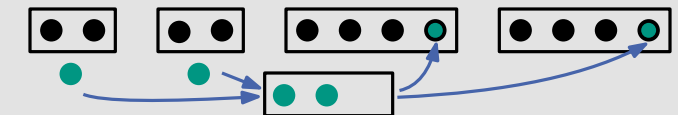
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

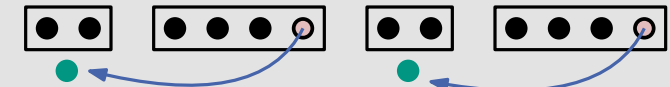


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$

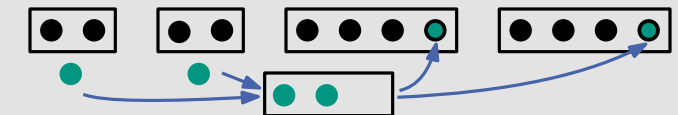
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

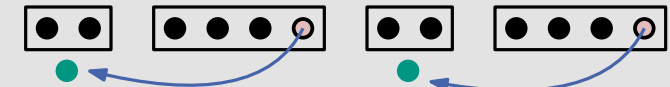


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$

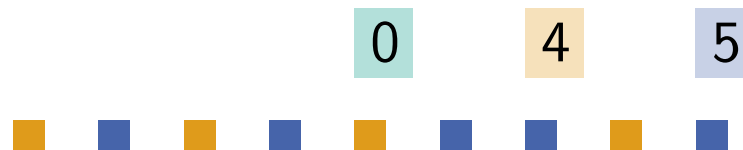
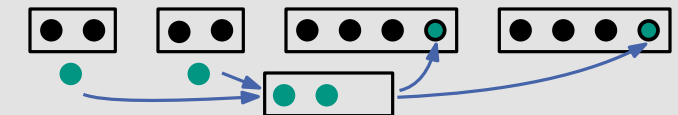
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0

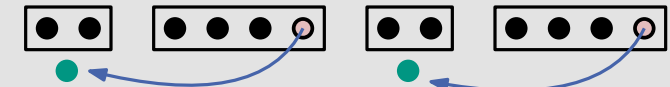


Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

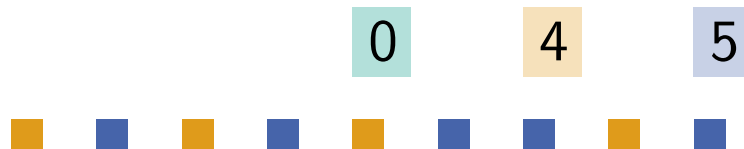
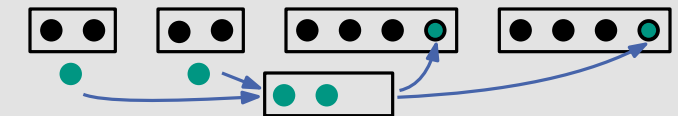
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



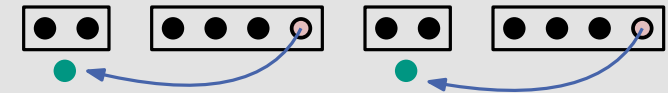
Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

Charging

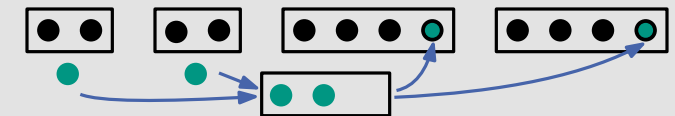
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



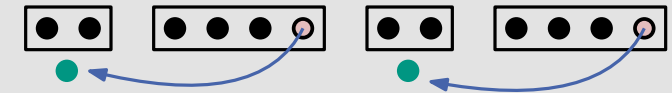
Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

Charging

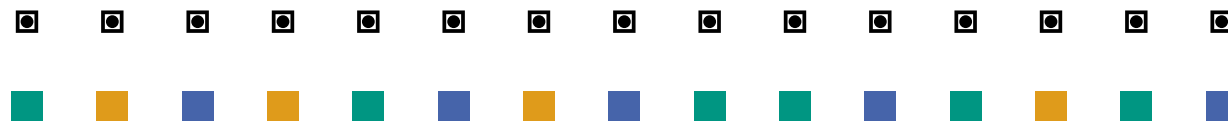
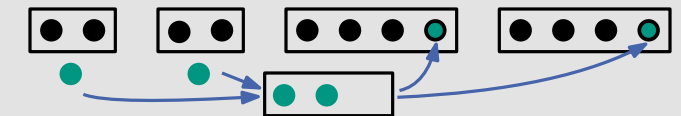
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



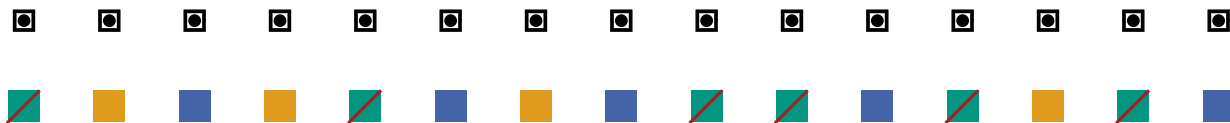
1 Token pro **insert**

Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

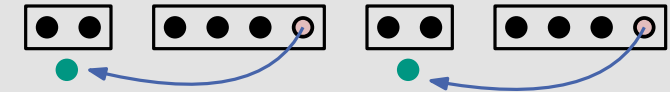
Charging

n



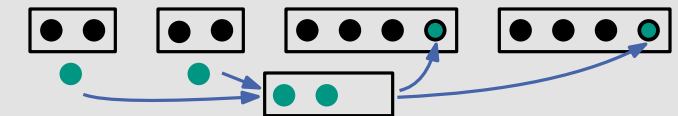
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



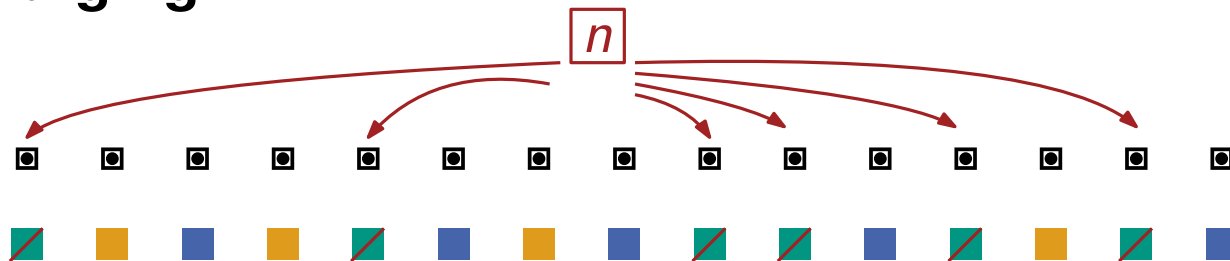
n Token pro **removeMost**

1 Token pro **insert**

Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

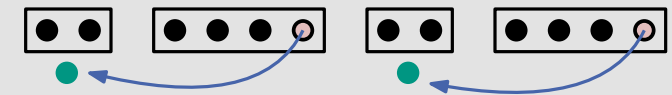
Charging



- verteile auf **inserts** der gelöschten Elemente

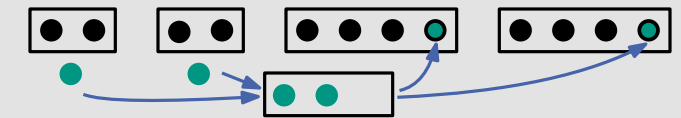
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



n Token pro **removeMost**

1 Token pro **insert**

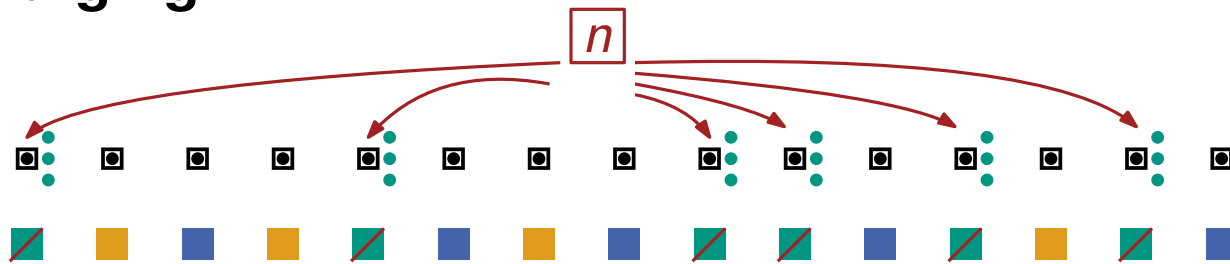
Wie viele Token bekommen die **inserts** zusätzlich?



Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

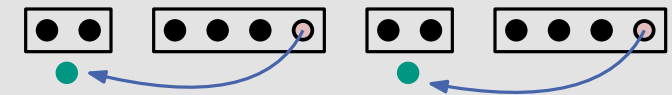
Charging



- verteile auf **inserts** der gelöschten Elemente
- jedes **insert** bekommt \leq drei zusätzliche Token

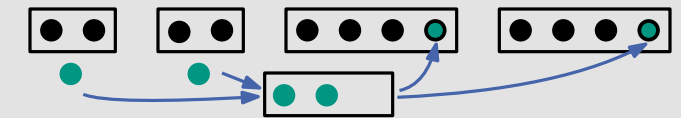
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



n Token pro **removeMost**

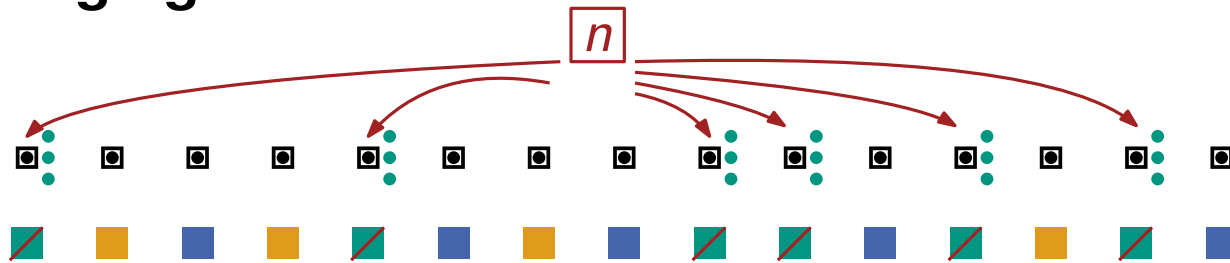
1 Token pro **insert**

Wie viele Token bekommen die **inserts** zusätzlich?

Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

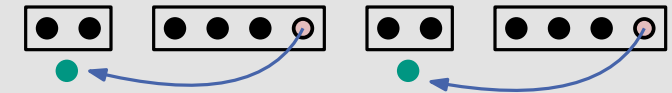
Charging



- verteile auf **inserts** der gelöschten Elemente
- jedes **insert** bekommt \leq drei zusätzliche Token
- amortisiert: 0 Token pro **removeMost**, 4 Token pro **insert**

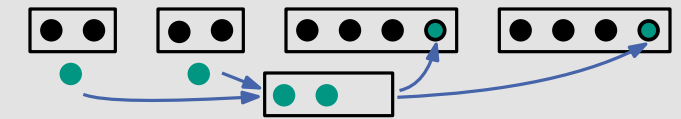
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



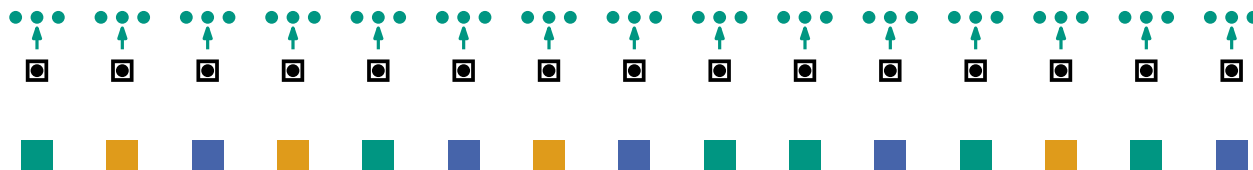
n Token pro **removeMost**

1 Token pro **insert**

Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

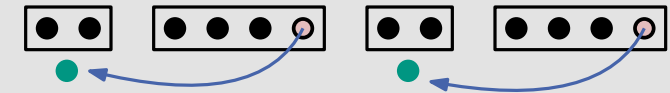
Konto



- zahle drei Token pro **insert** ins Konto

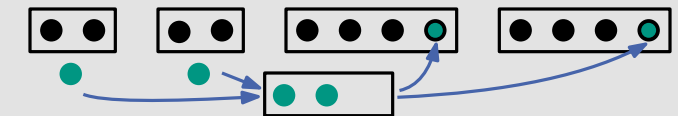
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

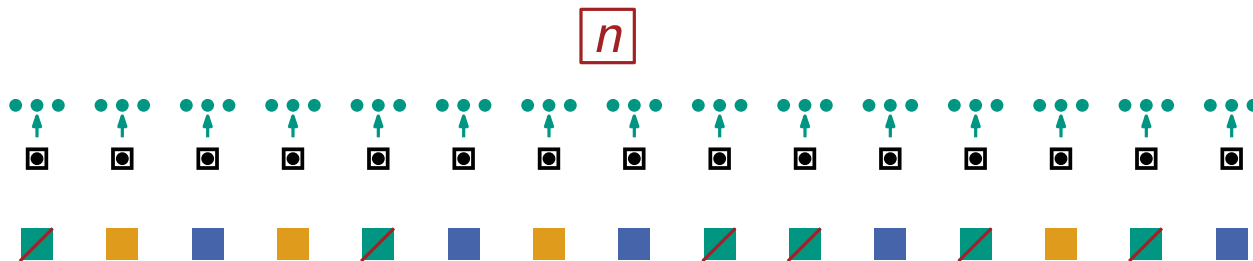
- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

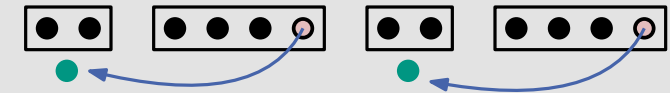
Konto



- zahle drei Token pro **insert** ins Konto
- hebe n Token pro **removeMost** vom Konto ab

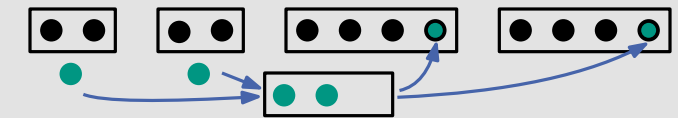
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



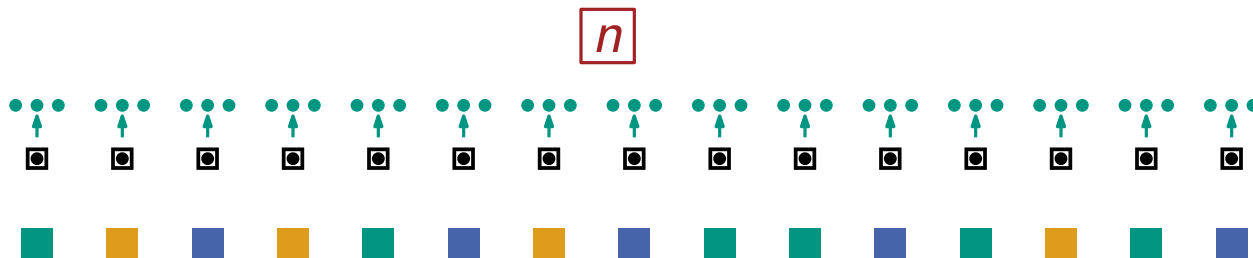
n Token pro **removeMost**

1 Token pro **insert**

Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

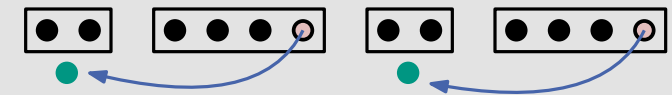
Konto



- zahle drei Token pro **insert** ins Konto
- hebe n Token pro **removeMost** vom Konto ab
- Kontostand = $3 \cdot \#Kugeln$

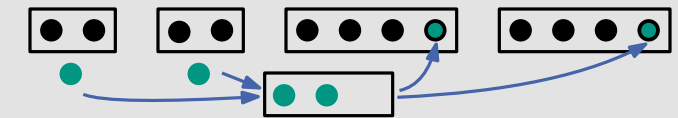
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



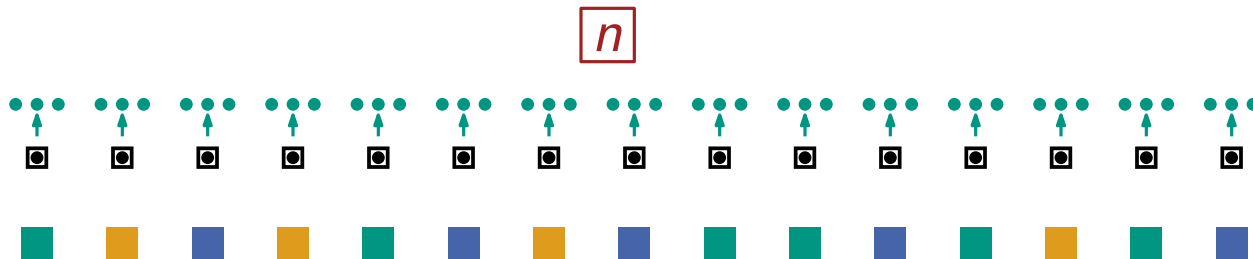
n Token pro **removeMost**

1 Token pro **insert**

Amortisierte Analyse

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - **insert**(pos, color) in $\Theta(1)$
 - **removeMost**(): Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ Kugeln gelöscht

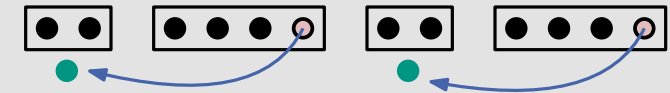
Konto



- zahle drei Token pro **insert** ins Konto
- hebe n Token pro **removeMost** vom Konto ab
- Kontostand = $3 \cdot \#Kugeln \geq 0$

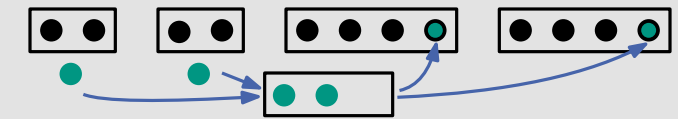
Charging

- Kostentoken: teure Op \rightarrow günstige Op



Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op
- Konto ≥ 0



n Token pro **removeMost**

1 Token pro **insert**