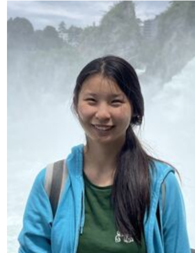
Karlsruher Institut für Technologie

Algorithmen 1

Übung 1
Asymptotik, Pseudocode & Amortisierte Analyse

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`

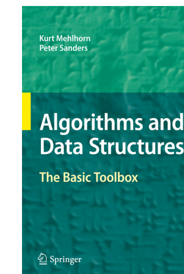
Übungsblätter

- 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

Karlsruher Institut für Technologie

- 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 $\boldsymbol{x}$ Ziffern $\rightarrow n = x$
  - Suche in einer Menge von $\boldsymbol{k}$ Zahlen die je maximal $\boldsymbol{x}$ Ziffern haben $\rightarrow n = k$

Abstraktionsniveau!

Asymptotik

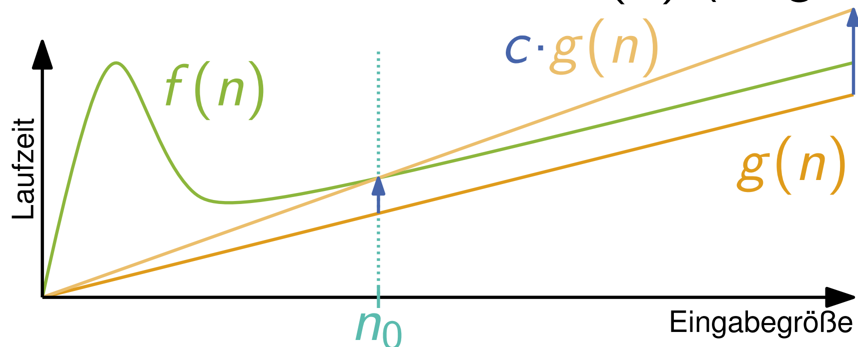
Karlsruher Institut für Technologie

- 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

Karlsruher Institut für Technologie

Was wächst schneller?

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

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

Nebenrechnung

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

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

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

$$1 \leq n \qquad n_0 = 1$$

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

Karlsruher Institut für Technologie

Was wächst schneller?

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

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

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

Nebenrechnung

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

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

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

$$1 \leq n \qquad n_0 = 1$$

Für $c = \frac{5}{2}$ (korrigiert: $\frac{10^{10}}{10^{-10}}$) gilt für alle $n > 1$: $5n^2 \leq c \cdot 2n^3$ (korrigiert: $10^{10}n^2 \leq c \cdot 10^{-10}n^3$).

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

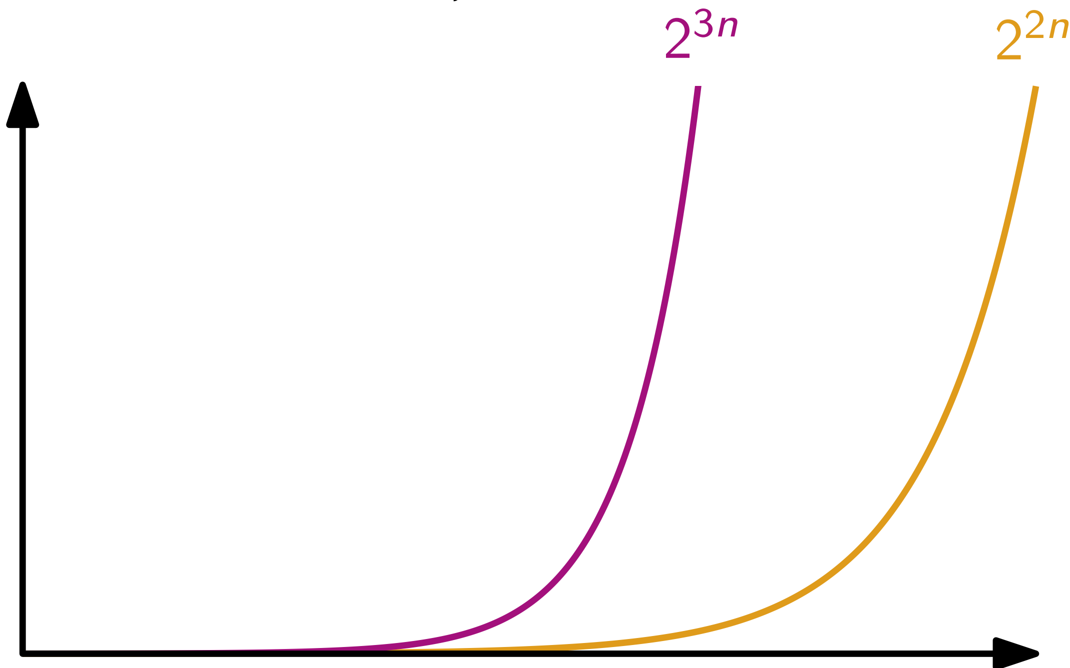
Asymptotik – Beispiele

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

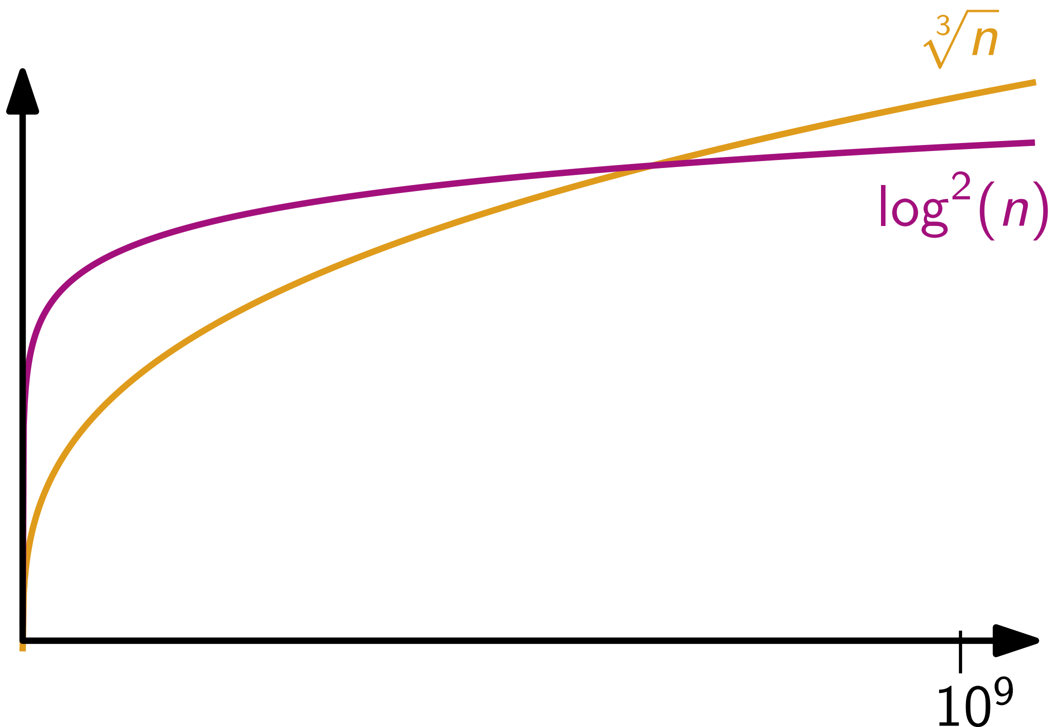
Was wächst schneller?

$5n^2$ 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

Karlsruher Institut für Technologie

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 $\sqrt[3]{n}$

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

beides ableiten (L'Hôpital)

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

vereinfachen

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

beides ableiten (L'Hôpital)

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

vereinfachen

$$= \lim_{n\to\infty} \frac{18\cdot n^{1/3}}{n} = \lim_{n\to\infty} \frac{18}{n^{2/3}}$$

$$= 0$$

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

Asymptotik – Beispiele

Karlsruher Institut für Technologie

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!
- $f(n) \in o(g(n)) \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
- Logarithmen wachsen langsamer als Polynome
- Logarithmengesetze sind hilfreich

Asymptotik – Beispiele

Karlsruher Institut für Technologie

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!
- $f(n) \in o(g(n)) \Rightarrow \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$
- Logarithmen wachsen langsamer als Polynome
- Logarithmengesetze sind hilfreich

Pseudocode

Karlsruher Institut für Technologie

- Ein Zwischenschritt auf dem Weg von der Algorithmusidee zur Implementierung

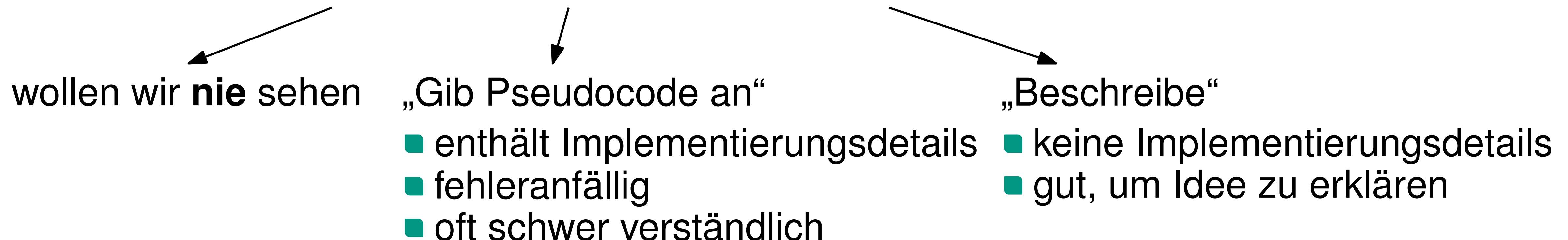
Algorithmusidee	Wir iterieren alle Ziffern und ...
Pseudocode	**for** i in $\{0, 1, \ldots, n-1\}$ **do** ...
Code	`for (int i = 0; i < n; ++i) { ... }`

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

- für Menschen besser lesbar als Code
- richtige Abstraktionsebene

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

Code ≠ Pseudocode ≠ Beschreibung



wollen wir **nie** sehen

„Gib Pseudocode an“
- enthält Implementierungsdetails
- fehleranfällig
- oft schwer verständlich

„Beschreibe“
- keine Implementierungsdetails
- gut, um Idee zu erklären

Pseudocode & Rekursion

Karlsruher Institut für Technologie

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

```
if k − d == 1 return I[d]
t := ⌊(d + k)/2⌋
a := mysteryFunc(I, d, t)
b := mysteryFunc(I, t, k)
if a < b then
    return a
else
    return b
```

Was tut dieser Pseudocode?

- Typen spezifizieren

Pseudocode & Rekursion

Karlsruher Institut für Technologie

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

```
if end − start == 1 return A[start]
mid := ⌊(start + end)/2⌋
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

Karlsruher Institut für Technologie

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

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?

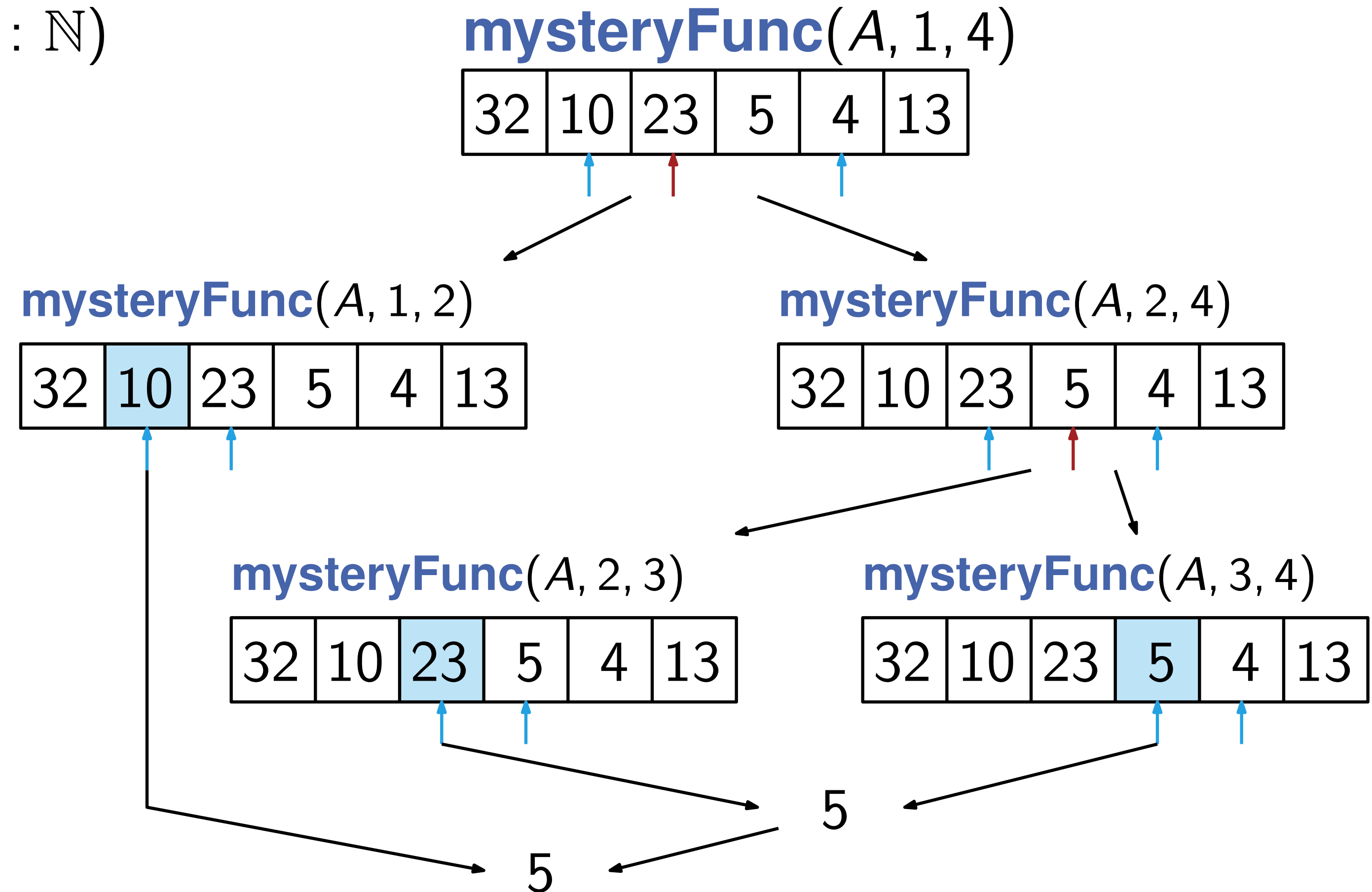
- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

Karlsruher Institut für Technologie

$\textbf{mysteryFunc}(A\colon \text{Array}, \text{start}\colon \mathbb{N}, \text{end}\colon \mathbb{N})$

- **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

$\text{minInterval}(A\colon \text{Array}, \text{start}\colon \mathbb{N}, \text{end}\colon \mathbb{N})$

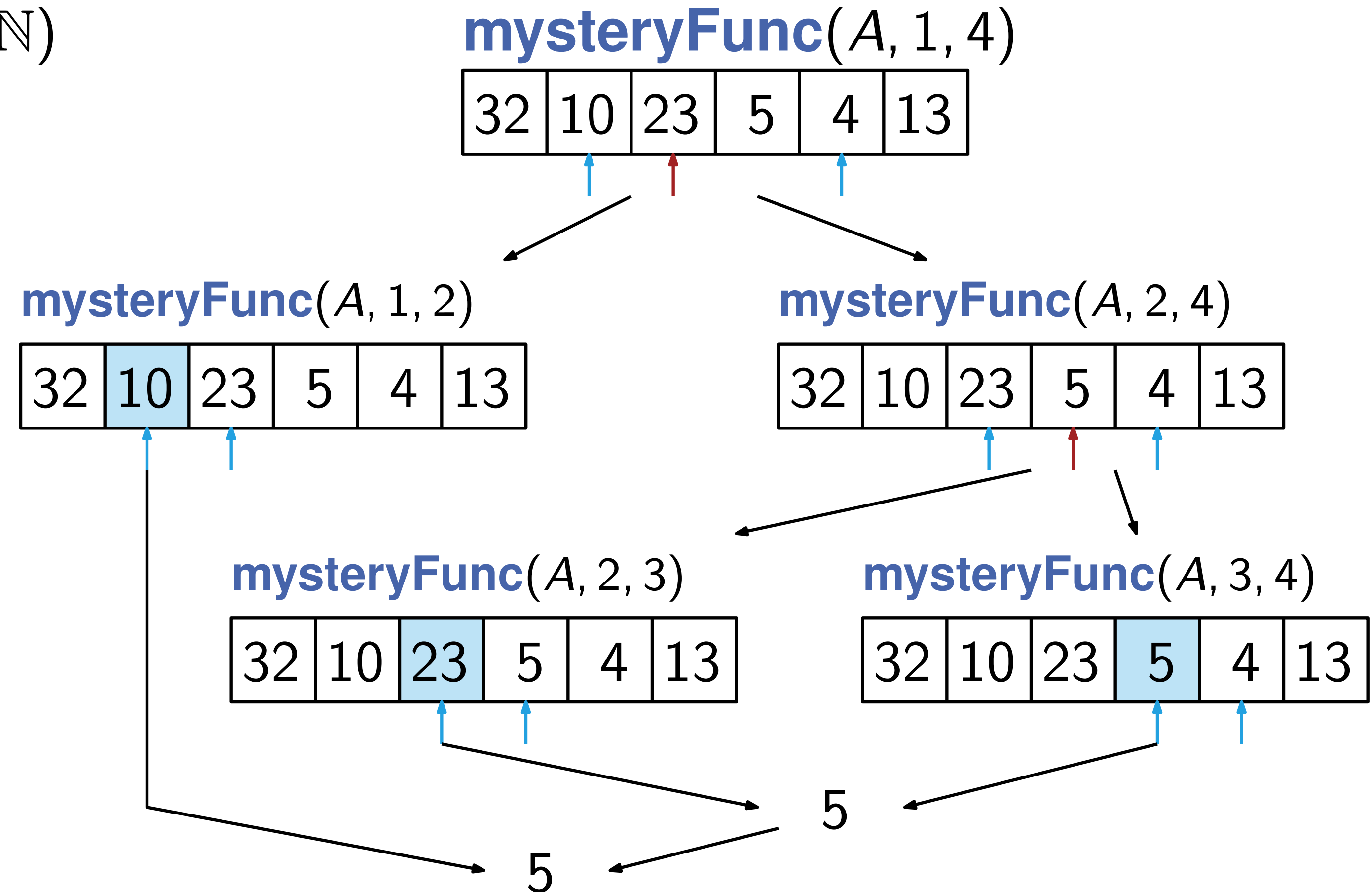
$\quad$ **if** $\text{end} - \text{start} == 1$ **return** $A[\text{start}]$

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

$\quad \text{left} := \text{minInterval}(A, \text{start}, \text{mid})$

$\quad \text{right} := \text{minInterval}(A, \text{mid}, \text{end})$

$\quad$ **return min**$(\text{left}, \text{right})$



Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Pseudocode & Rekursion

Karlsruher Institut für Technologie

$\textbf{minInterval}(A\colon \text{Array}, \text{start}\colon \mathbb{N}, \text{end}\colon \mathbb{N})$

- **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)

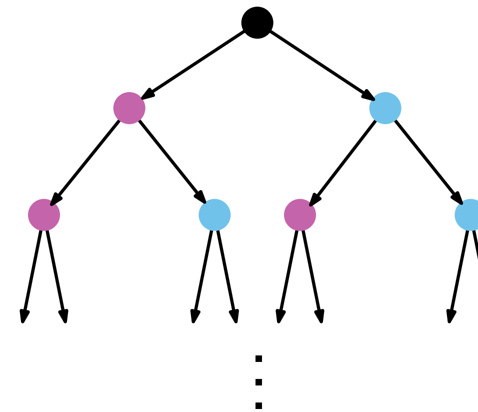
Was tut dieser Pseudocode?

- Typen spezifizieren
- sinnvolle Variablennamen wählen
- bekannte Subroutinen verwenden

Laufzeit?

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

Rekursionsbaum



Pseudocode & Rekursion

Karlsruher Institut für Technologie

minInterval$(A\colon \text{Array}, \text{start}\colon \mathbb{N}, \text{end}\colon \mathbb{N})$

```
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)
```

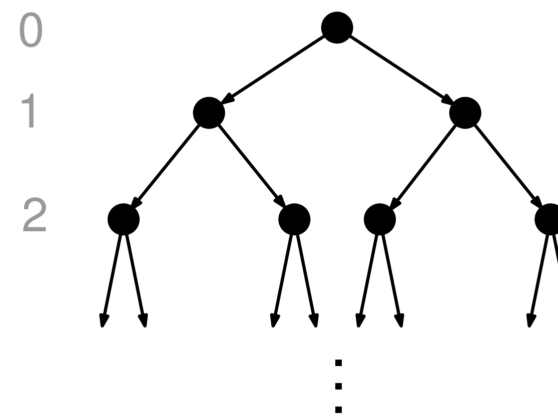
Was tut dieser Pseudocode?

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

2D-Search

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

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

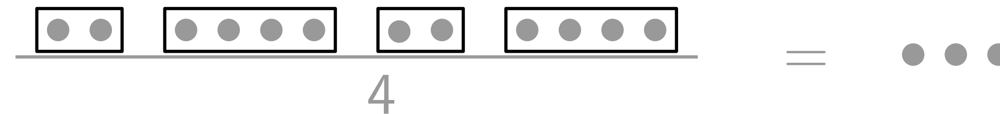
$\Longrightarrow$ **duplicateArray** im Worst Case $\Theta(n)$

Amortisierte Analyse

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

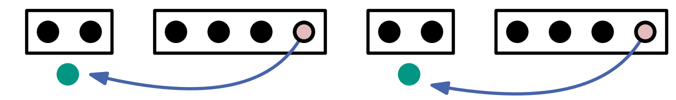
Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen



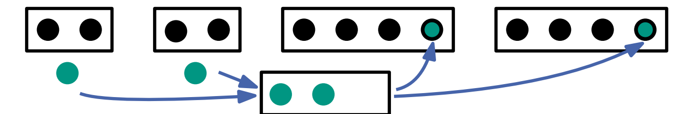
Charging

- 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

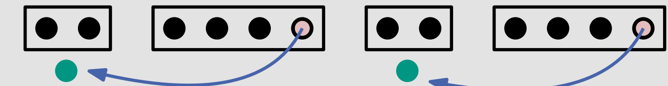
Amortisierte Analyse

Karlsruher Institut für Technologie

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

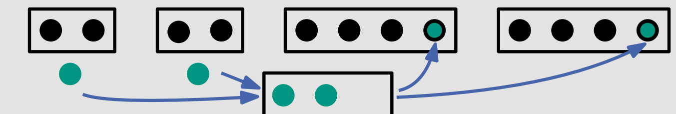
Charging

- Kostentoken: teuere Op $\rightarrow$ günstige Op



Kontomethode

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



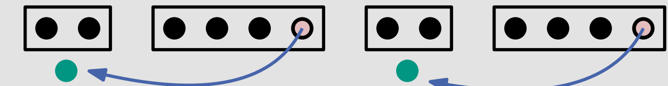
Amortisierte Analyse

Karlsruher Institut für Technologie

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

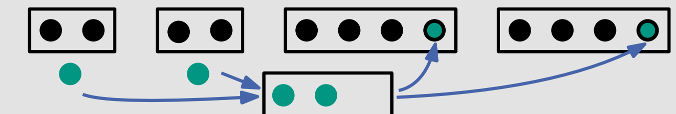
Charging

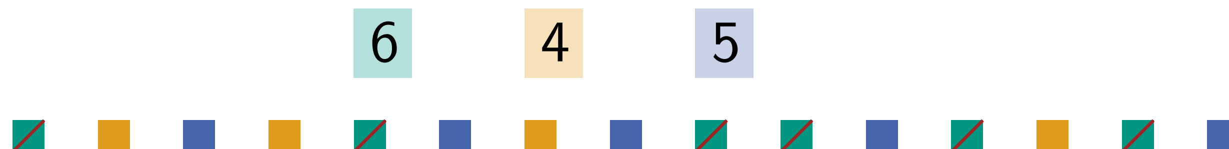
- Kostentoken: teuere Op $\rightarrow$ günstige Op



Kontomethode

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

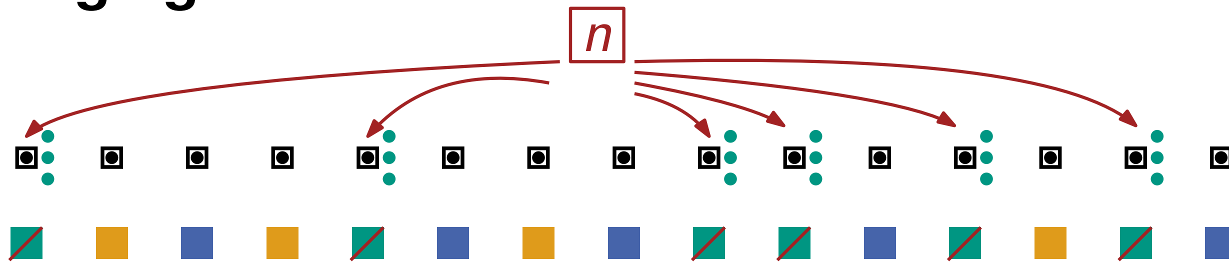




Amortisierte Analyse

- 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



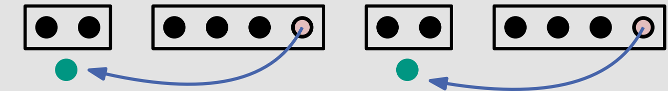
n Token pro **removeMost**

1 Token pro **insert**

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

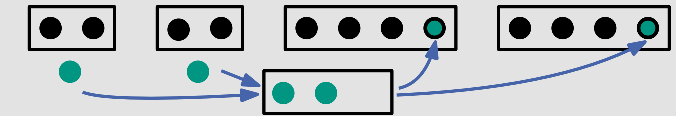
Charging

- Kostentoken: teuere Op $\rightarrow$ günstige Op



Kontomethode

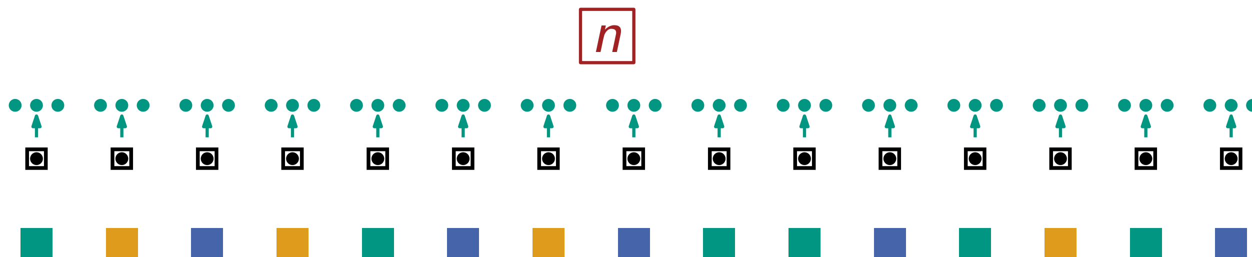
- günstige Op $\rightarrow$ Konto, Konto $\rightarrow$ teure Op
- Konto ≥ 0



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

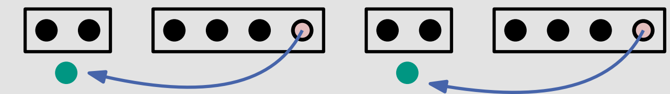
Konto



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

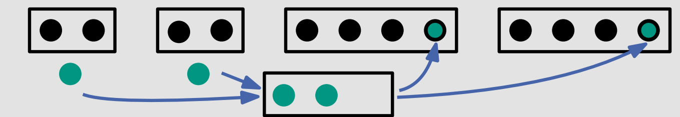
Charging

- Kostentoken: teuere Op $\rightarrow$ günstige Op



Kontomethode

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



n Token pro **removeMost**

1 Token pro **insert**