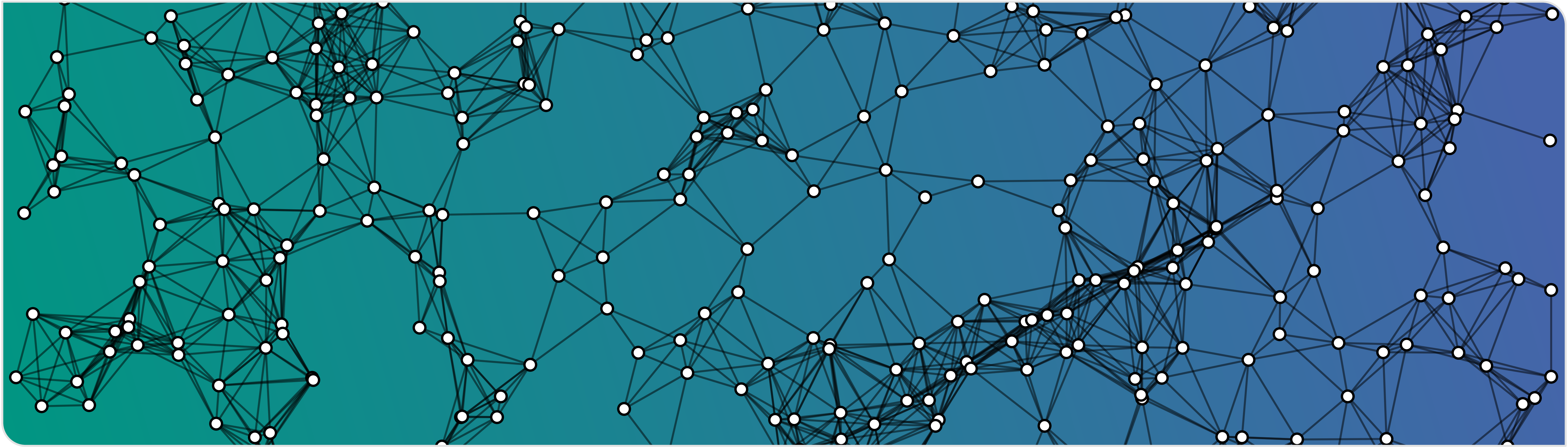
Karlsruher Institut für Technologie

Algorithmen 1

Übung 2
Aufgabe B01.A1, amortisierte Analyse



Organisatorisches

- Blatt 2, Aufgabe 1, erster Algo

Organisatorisches

- Blatt 2, Aufgabe 1, erster Algo
  - schlechte Aufgabenstellung, zu schwierig

Organisatorisches

- Blatt 2, Aufgabe 1, erster Algo
  - schlechte Aufgabenstellung, zu schwierig
  - $\Rightarrow$ volle Punktzahl + Bonuspunkte (siehe Vorlesung 4)

Organisatorisches

- Blatt 2, Aufgabe 1, erster Algo
  - schlechte Aufgabenstellung, zu schwierig
  - $\Rightarrow$ volle Punktzahl + Bonuspunkte (siehe Vorlesung 4)
  - Gleich: Vorstellung der Lösung

Organisatorisches

- Blatt 2, Aufgabe 1, erster Algo
  - schlechte Aufgabenstellung, zu schwierig
  - $\Rightarrow$ volle Punktzahl + Bonuspunkte (siehe Vorlesung 4)
  - Gleich: Vorstellung der Lösung
- Zusätzliche Tutorien zum Nachholen / Klausurvorbereitung im August

Organisatorisches

- Blatt 2, Aufgabe 1, erster Algo
  - schlechte Aufgabenstellung, zu schwierig
  - $\Rightarrow$ volle Punktzahl + Bonuspunkte (siehe Vorlesung 4)
  - Gleich: Vorstellung der Lösung
- Zusätzliche Tutorien zum Nachholen / Klausurvorbereitung im August
  - mehr Infos demnächst

Organisatorisches

Karlsruher Institut für Technologie

- Blatt 2, Aufgabe 1, erster Algo
  - schlechte Aufgabenstellung, zu schwierig
  - $\Rightarrow$ volle Punktzahl + Bonuspunkte (siehe Vorlesung 4)
  - Gleich: Vorstellung der Lösung
- Zusätzliche Tutorien zum Nachholen / Klausurvorbereitung im August
  - mehr Infos demnächst
- Update der Pseudocode Richtlinien

Übung 2: Aufgabe 1: ALGONE

Übung 2: Aufgabe 1: ALGONE

Gegeben:

Übung 2: Aufgabe 1: ALGONE

Karlsruher Institut für Technologie

Gegeben:

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Übung 2: Aufgabe 1: ALGONE

Karlsruher Institut für Technologie

Gegeben:

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Gesucht:

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \vee x = y$ **then**
    return 1
  return algOne(x-1, y-1) + algOne(x, y-1)

Gesucht:

- „Problemgröße“ n

Übung 2: Aufgabe 1: ALGONE

Karlsruher Institut für Technologie

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
    if $x = 0 \lor x = y$ **then**
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)

Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

    if $x = 0 \lor x = y$ **then**

        return 1

    return algOne(x-1, y-1) + **algOne**(x, y-1)

Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Ann

if $x = 0 \lor x = y$ then

return 1

return algOne(x-1, y-

Gesucht:

- „Problemgröße" n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

algTwo(x, y) // Annahme: initial $x \leq y$

if $x \geq y$ then

return x

$total$:= algTwo(x, $\lfloor \frac{y+3x}{4} \rfloor$)

$total$ += algTwo($\lfloor \frac{y+3x}{4} \rfloor$), $\lfloor \frac{y+x}{2} \rfloor$)

$total$ += algTwo($\lfloor \frac{y+x}{2} \rfloor$), $\lfloor \frac{3y+x}{4} \rfloor$)

return $total$

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne$(x\colon \mathbb{N}, y\colon \mathbb{N})$ // Ann

- **if** $x = 0 \vee x = y$ **then**
  - **return** 1
- **return algOne**(x-1, y-

algTwo(x, y) // Annahme: initial $x \leq y$

- **if** $x \geq y$ **then**
  - **return** x
- $total := $ **algTwo**$(x, \lfloor \frac{y+3x}{4} \rfloor)$
- $total \mathrel{+}= $ **algTwo**$(\lfloor \frac{y+3x}{4} \rfloor), \lfloor \frac{y+x}{2} \rfloor)$
- $total \mathrel{+}= $ **algTwo**$(\lfloor \frac{y+x}{2} \rfloor), \lfloor \frac{3y+x}{4} \rfloor)$
- **return** $total$

- „Problemgröße“ $n := \max\{0, y - x\}$

Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne$(x\colon \mathbb{N}, y\colon \mathbb{N})$ // Ann

if $x = 0 \lor x = y$ **then**

  return 1

return algOne(x-1, y-

Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

algTwo(x, y) // Annahme: initial $x \leq y$

if $x \geq y$ **then**

  return x

$total := \textbf{algTwo}(x, \lfloor \frac{y+3x}{4} \rfloor)$

$total \mathrel{+}= \textbf{algTwo}(\lfloor \frac{y+3x}{4} \rfloor), \lfloor \frac{y+x}{2} \rfloor)$

$total \mathrel{+}= \textbf{algTwo}(\lfloor \frac{y+x}{2} \rfloor), \lfloor \frac{3y+x}{4} \rfloor)$

return $total$

- „Problemgröße“ $n := \max\{0, y - x\}$
  - $n' = 1/4 \cdot n$

Übung 2: Aufgabe 1: ALGONE

Karlsruher Institut für Technologie

Gegeben:

$\textsf{algOne}(x\colon \mathbb{N}, y\colon \mathbb{N})$ // Ann

if $x = 0 \lor x = y$ then

return 1

return algOne(x-1, y-

Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

$\textsf{algTwo}(x, y)$ // Annahme: initial $x \leq y$

if $x \geq y$ then

return x

$total := \textsf{algTwo}(x, \lfloor \frac{y+3x}{4} \rfloor)$

$total \mathrel{+}= \textsf{algTwo}(\lfloor \frac{y+3x}{4} \rfloor), \lfloor \frac{y+x}{2} \rfloor)$

$total \mathrel{+}= \textsf{algTwo}(\lfloor \frac{y+x}{2} \rfloor), \lfloor \frac{3y+x}{4} \rfloor)$

return $total$

- „Problemgröße“ $n := \max\{0, y - x\}$
  - $n' = 1/4 \cdot n$
- $T(n) = 3T(n/4) + c_1$, $T(0) = c_2$

Übung 2: Aufgabe 1: ALGONE

Karlsruher Institut für Technologie

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Ann

if $x = 0 \lor x = y$ then

return 1

return algOne(x-1, y-

Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

algTwo(x, y) // Annahme: initial $x \leq y$

if $x \geq y$ then

return x

$total$:= algTwo(x, $\lfloor \frac{y+3x}{4} \rfloor$)

$total$ += algTwo($\lfloor \frac{y+3x}{4} \rfloor$), $\lfloor \frac{y+x}{2} \rfloor$)

$total$ += algTwo($\lfloor \frac{y+x}{2} \rfloor$), $\lfloor \frac{3y+x}{4} \rfloor$)

return $total$

- „Problemgröße“ $n := \max\{0, y - x\}$
  - $n' = 1/4 \cdot n$
- $T(n) = 3T(n/4) + c_1$, $T(0) = c_2$
- $T(n) \in \Theta(n^{\log_4(3)})$

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \vee x = y$ **then**
    return 1
  return algOne(x-1, y-1) + **algOne**(x, y-1)

Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Übung 2: Aufgabe 1: ALGONE

Gegeben:

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
  if x = 0 ∨ x = y then
    return 1
  return algOne(x-1, y-1) + algOne(x, y-1)
```

Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

$(4, 8)$

algOne$(x\colon \mathbb{N}, y\colon \mathbb{N})$ // Annahme: initial $x \leq y$

if $x = 0 \lor x = y$ **then**

  return 1

return **algOne**(x-1, y-1) + **algOne**(x, y-1)

Gesucht:

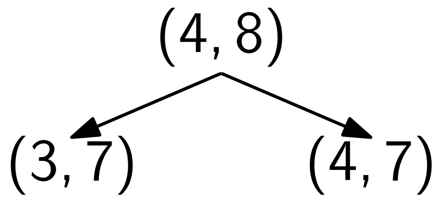
- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \lor x = y$ **then**
    return 1
  return algOne(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

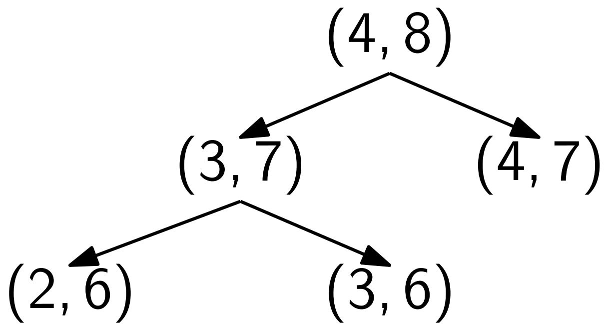
- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \lor x = y$ **then**
    return 1
  return algOne(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

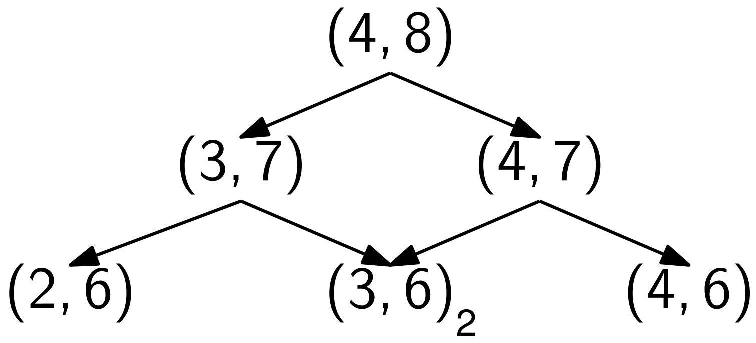
- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \vee x = y$ **then**
    return 1
  return algOne(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

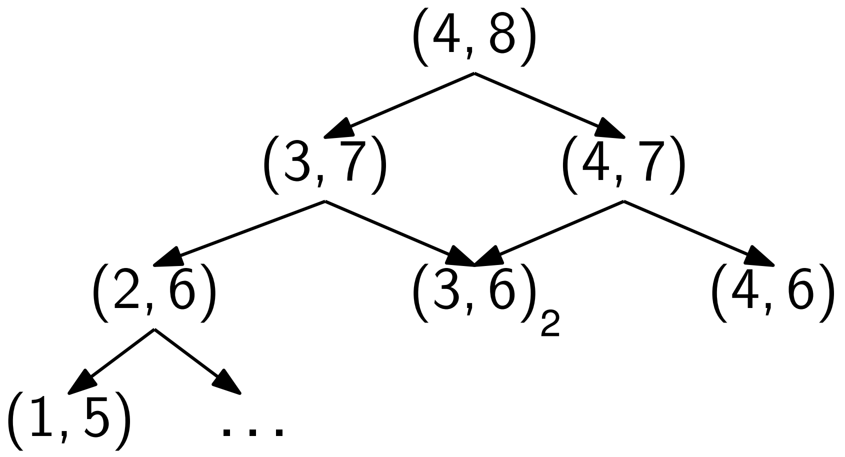
Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

  if $x = 0 \lor x = y$ **then**

    return 1

  return **algOne**(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

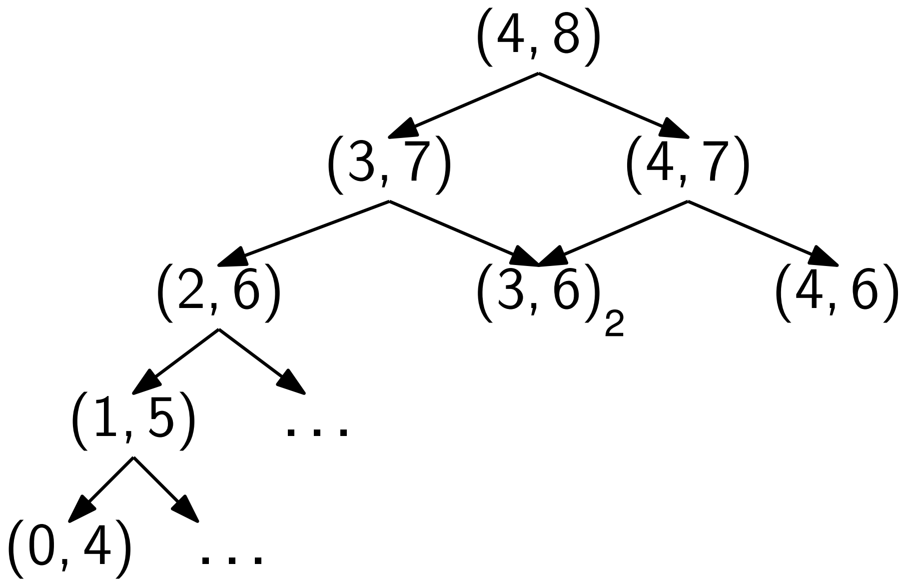
Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

    if $x = 0 \vee x = y$ **then**
        return 1
    return algOne(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

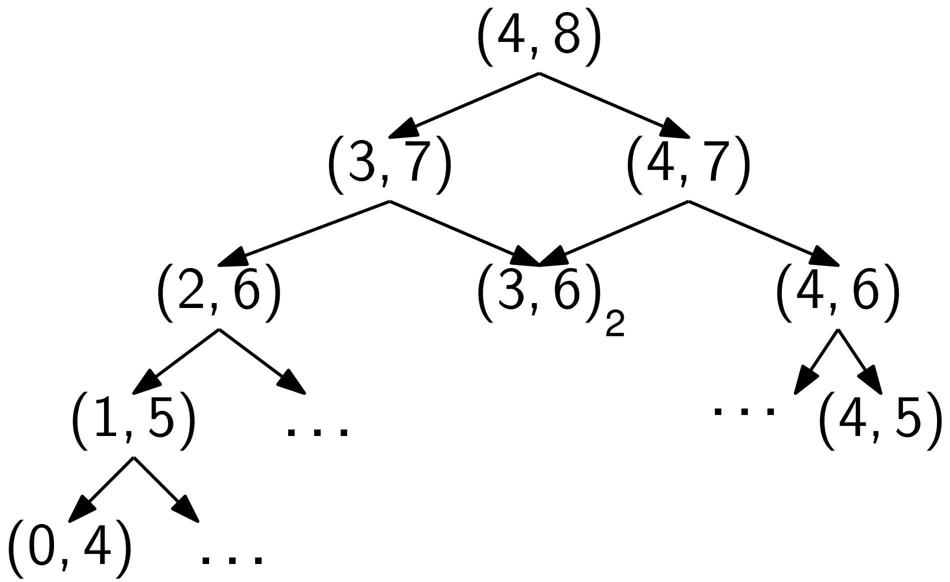
Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

  if $x = 0 \lor x = y$ **then**

    return 1

  return algOne(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

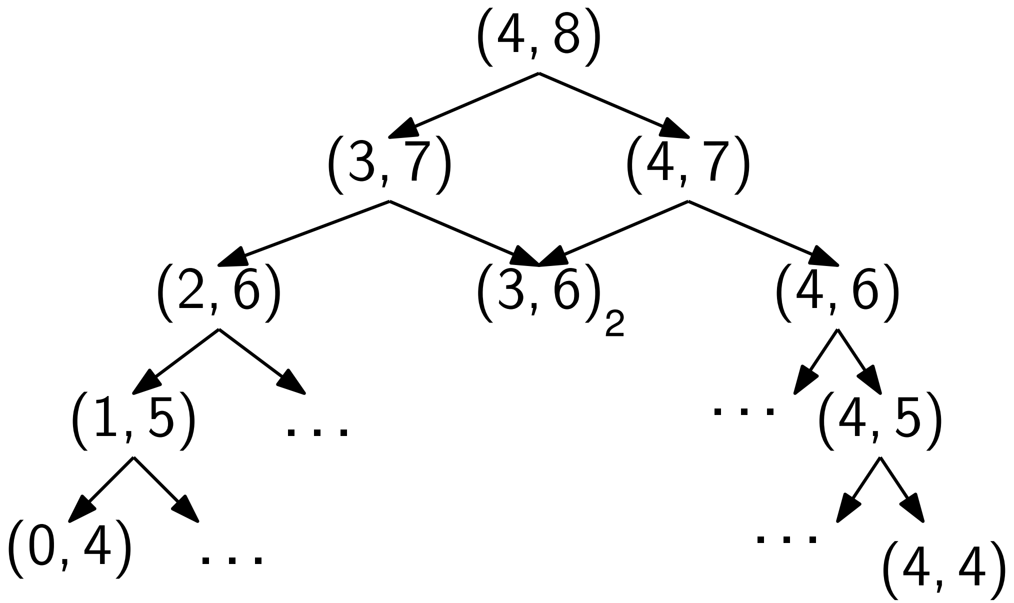
Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

    if $x = 0 \lor x = y$ **then**

        return 1

    return algOne(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

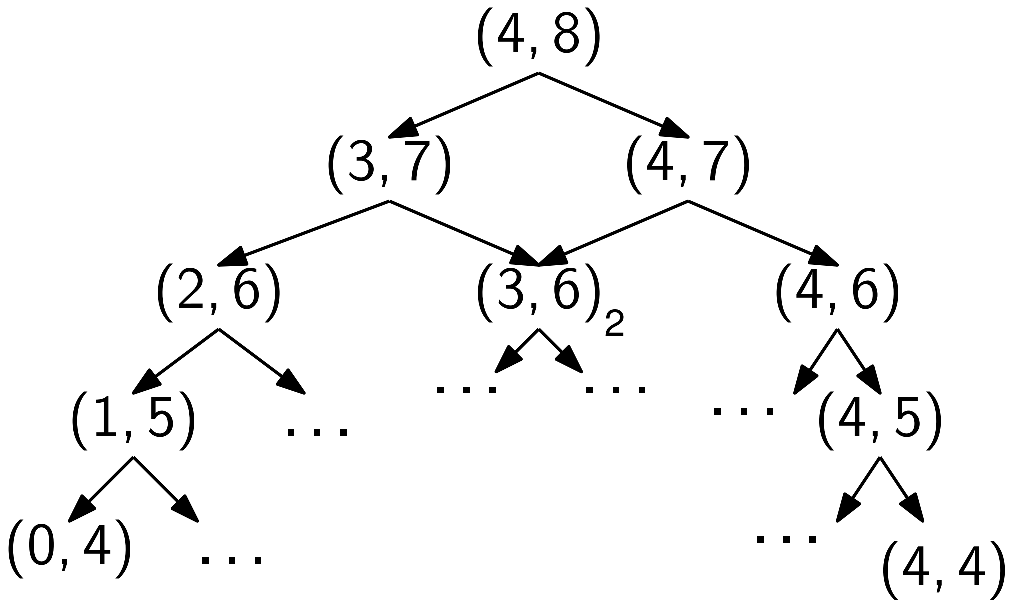
- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```



Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

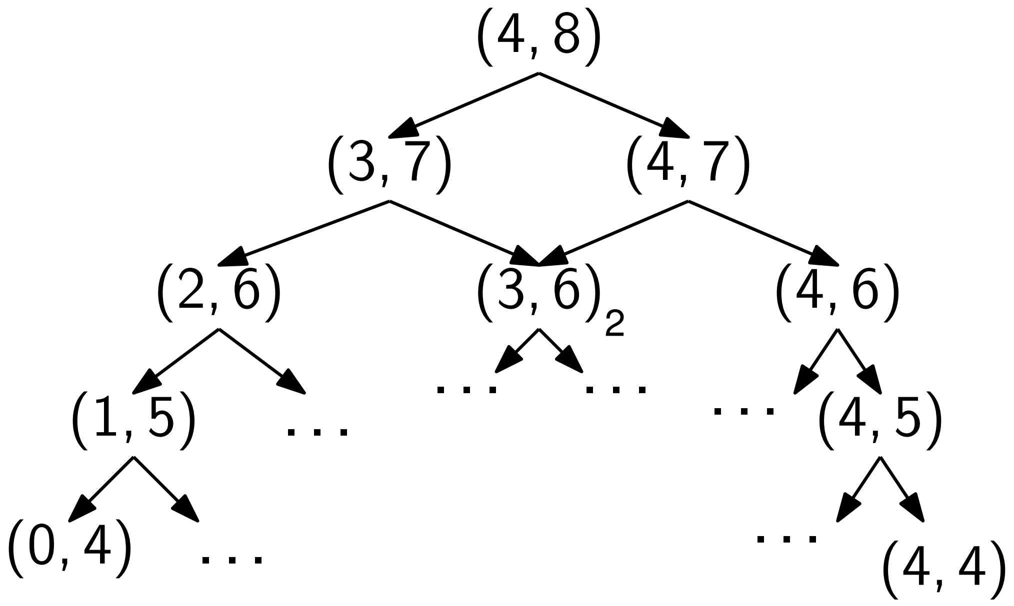
Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

    if $x = 0 \vee x = y$ **then**

        return 1

    return algOne(x-1, y-1) + **algOne**(x, y-1)



Das schaut kompliziert aus :(

Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Übung 2: Aufgabe 1: ALGONE

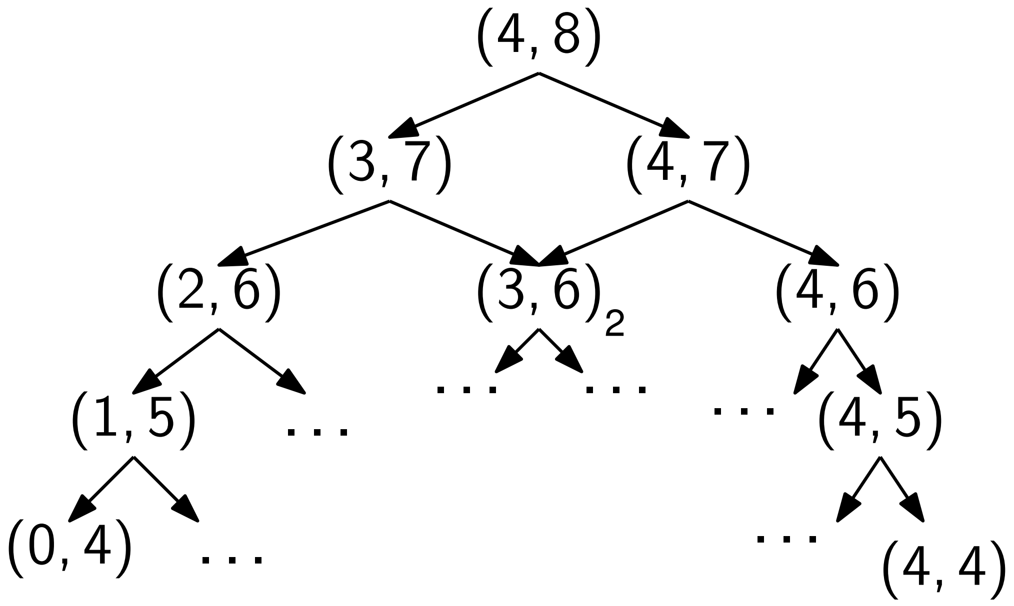
Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

  if $x = 0 \vee x = y$ **then**

    return 1

  return algOne(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Das schaut kompliziert aus :(

Plan

Übung 2: Aufgabe 1: ALGONE

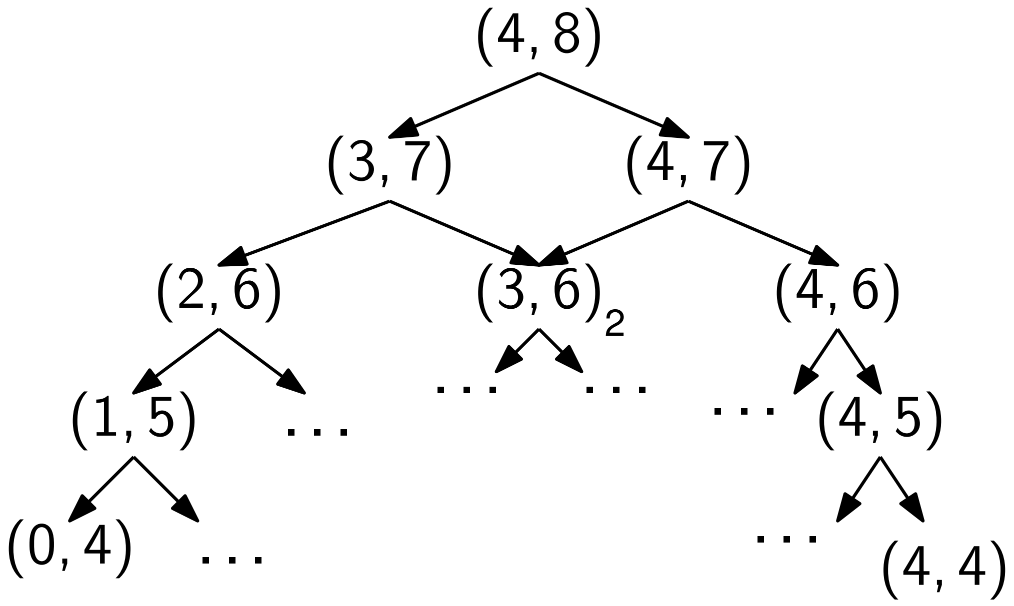
Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

    if $x = 0 \lor x = y$ **then**

        return 1

    return algOne(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Das schaut kompliziert aus :(

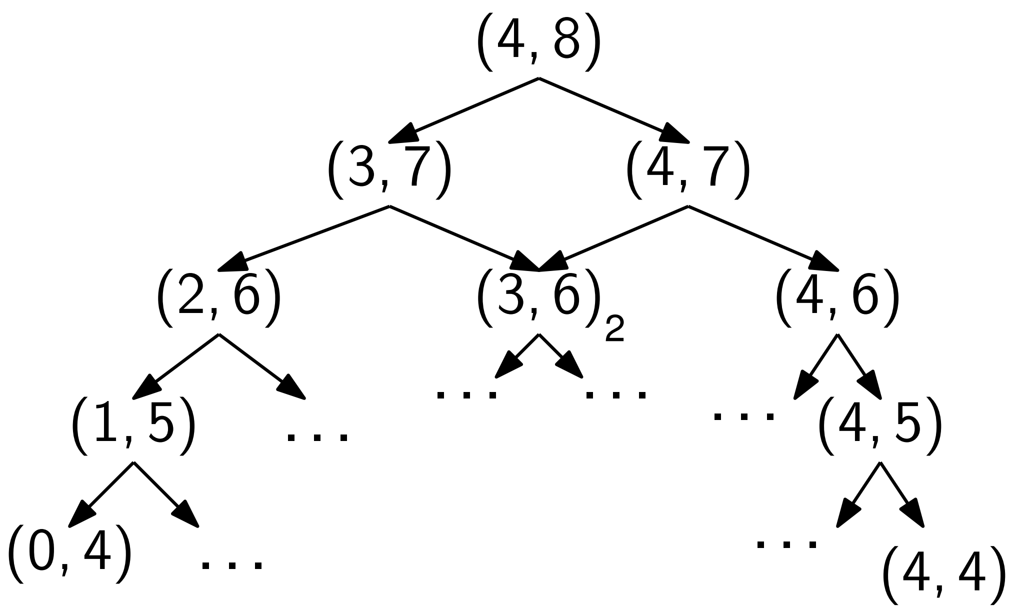
Plan

- (grobe) obere Schranke in y

Übung 2: Aufgabe 1: ALGONE

Gegeben:

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
if $x = 0 \lor x = y$ **then**
  return 1
return **algOne**(x-1, y-1) + **algOne**(x, y-1)



Gesucht:

- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei algOne?

Das schaut kompliziert aus :(

Plan

- (grobe) obere Schranke in y
- im Anschluss: genaue Analyse

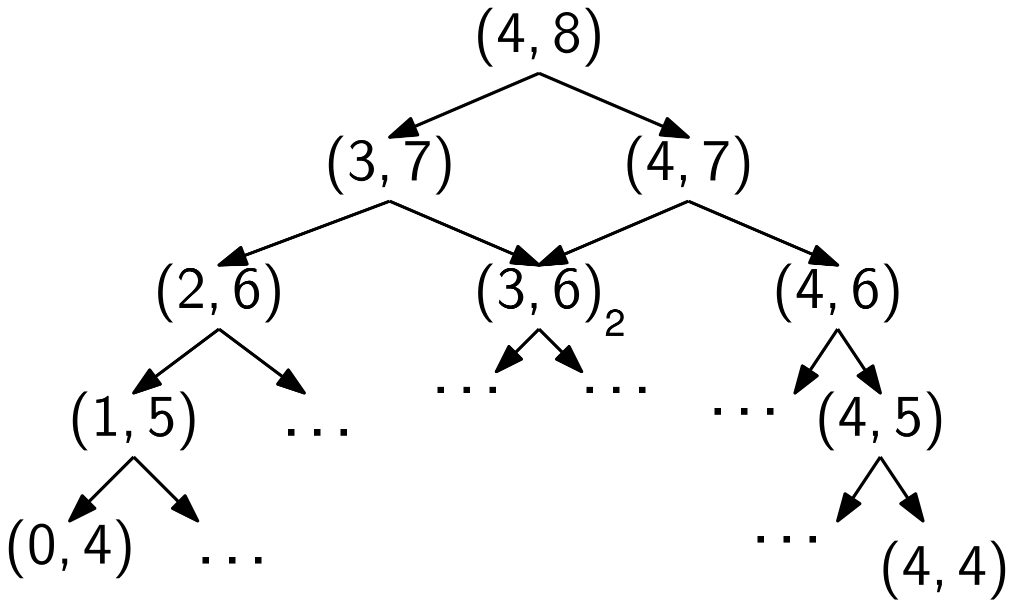
Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

if $x = 0 \vee x = y$ **then**

  return 1

return algOne(x-1, y-1) + **algOne**(x, y-1)



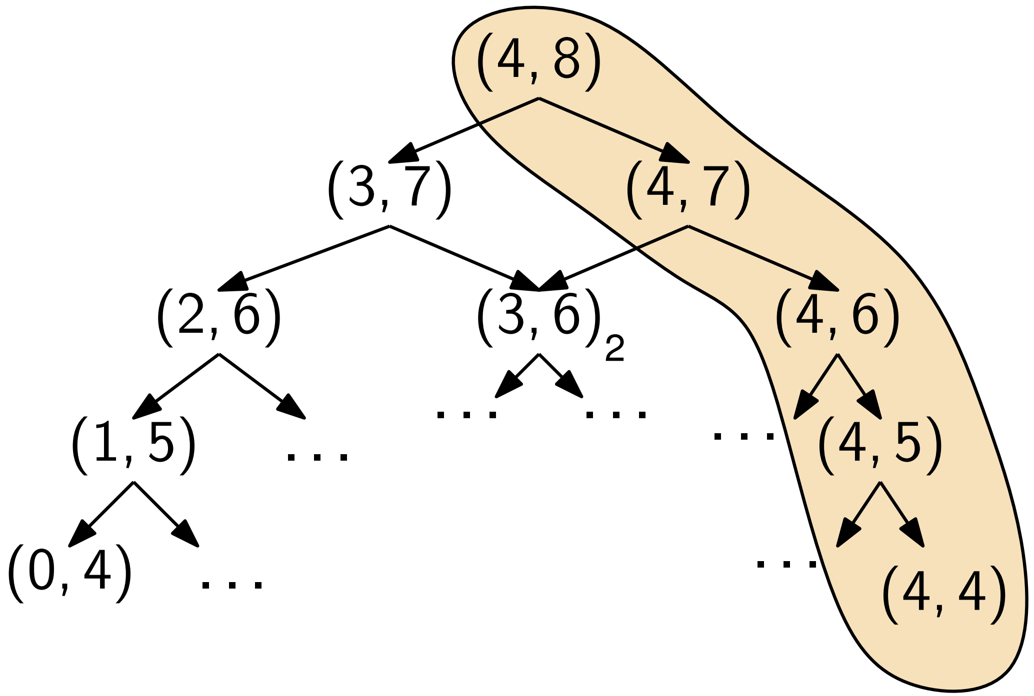
Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

    if $x = 0 \lor x = y$ **then**

        return 1

    return algOne(x-1, y-1) + **algOne**(x, y-1)



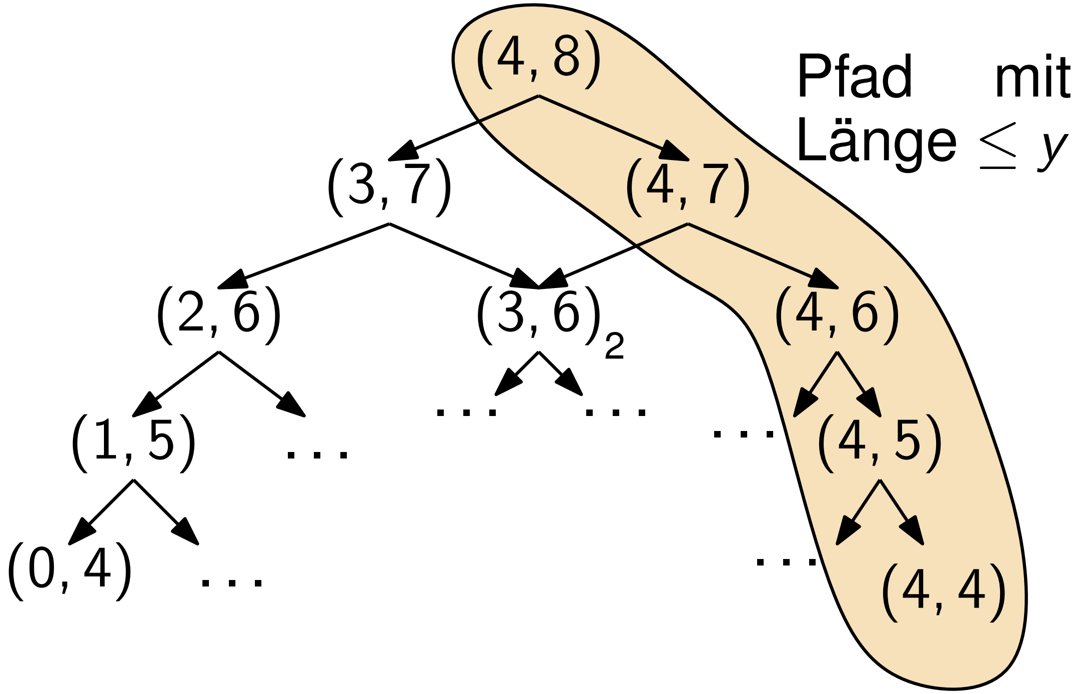
Ungenaue obere Schranke ALGONE

algOne($x\colon \mathbb{N}, y\colon \mathbb{N}$) // Annahme: initial $x \leq y$

if $x = 0 \lor x = y$ **then**

  return 1

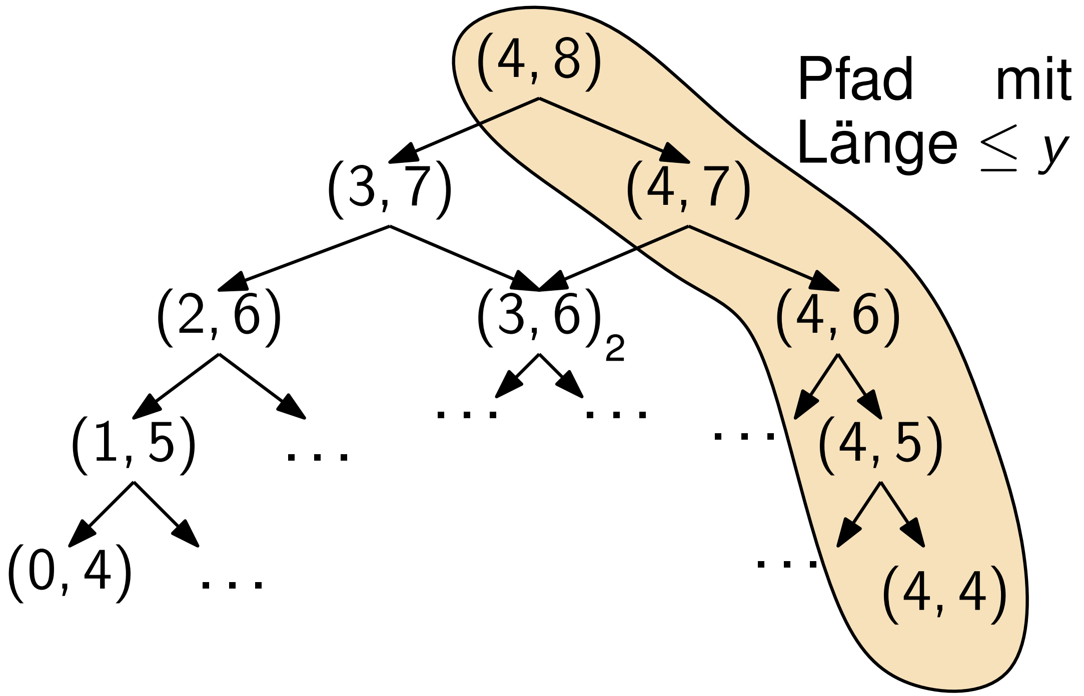
return algOne(x-1, y-1) + algOne(x, y-1)



Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \lor x = y$ **then**
    return 1
  return **algOne**(x-1, y-1) + **algOne**(x, y-1)

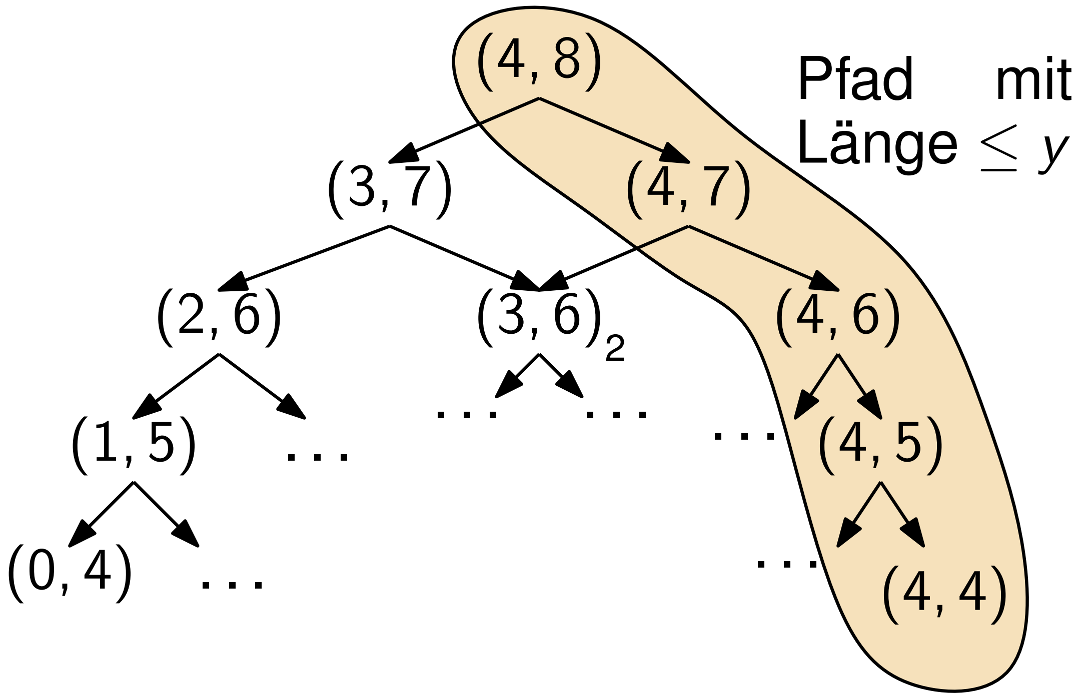
Beobachtung



Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

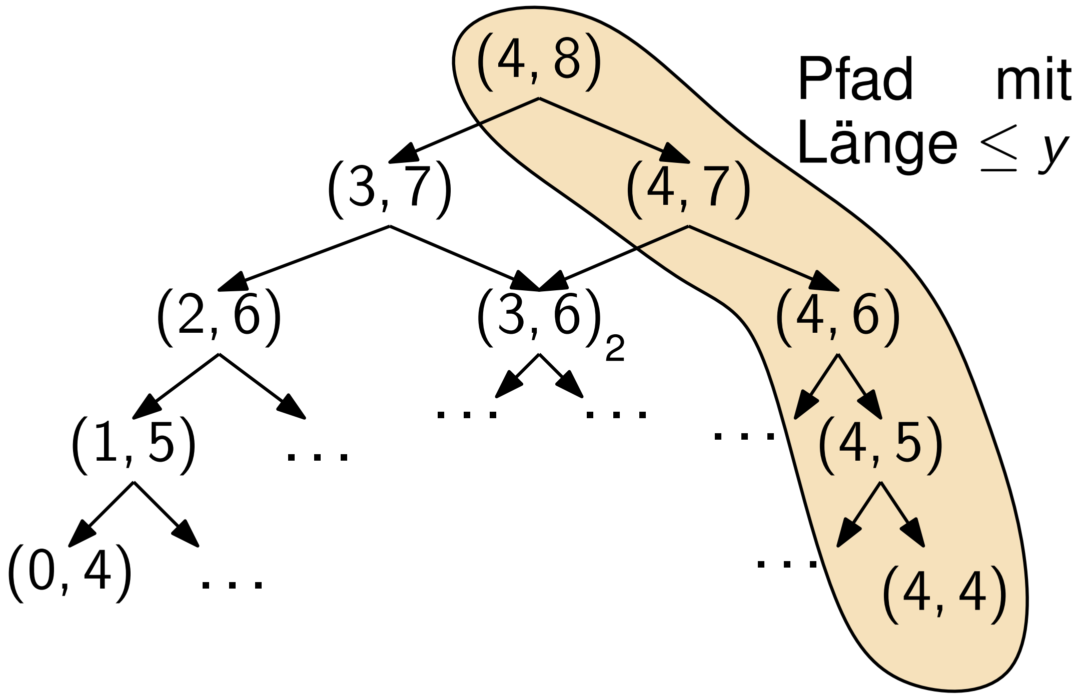


Beobachtung

- y sinkt in jedem Aufruf um 1

Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \lor x = y$ **then**
    return 1
  return algOne(x-1, y-1) + algOne(x, y-1)



Beobachtung

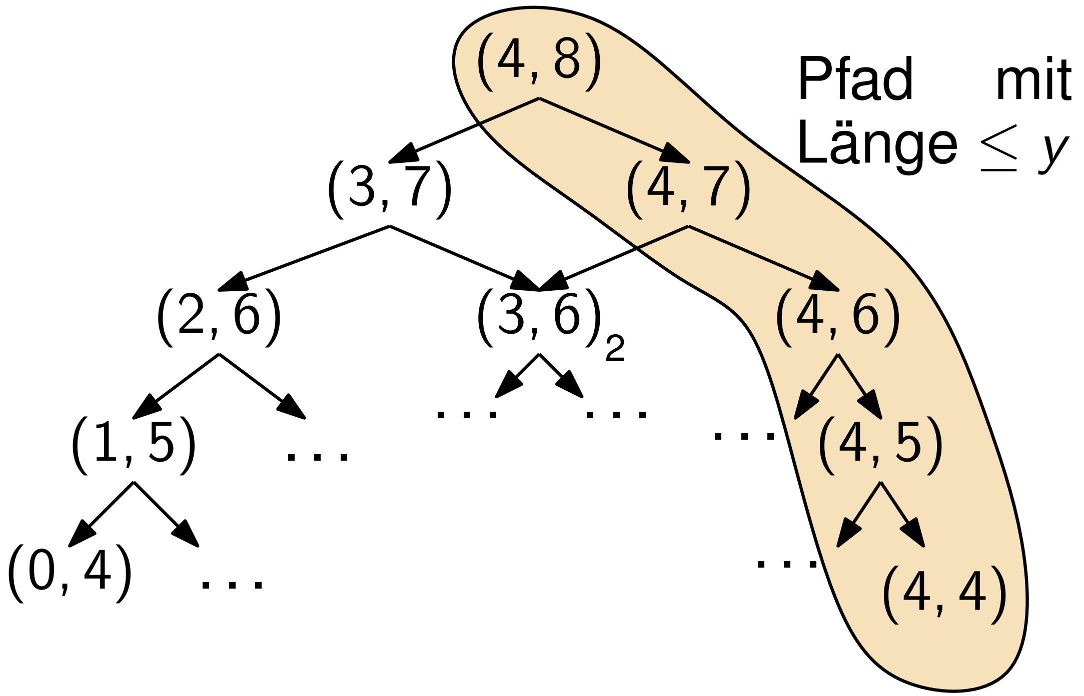
- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y

Ungenaue obere Schranke ALGONE


```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

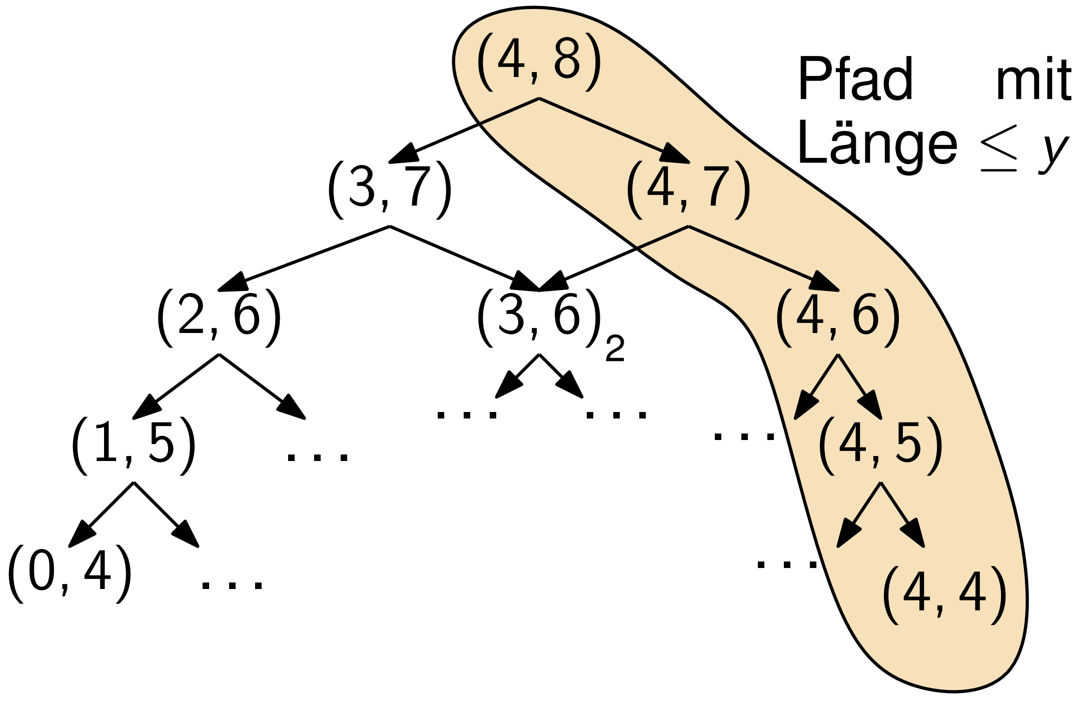
Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe



Ungenaue obere Schranke ALGONE

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
  if x = 0 ∨ x = y then
    return 1
  return algOne(x-1, y-1) + algOne(x, y-1)
```



Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\leq 2 \cdot 2^y$

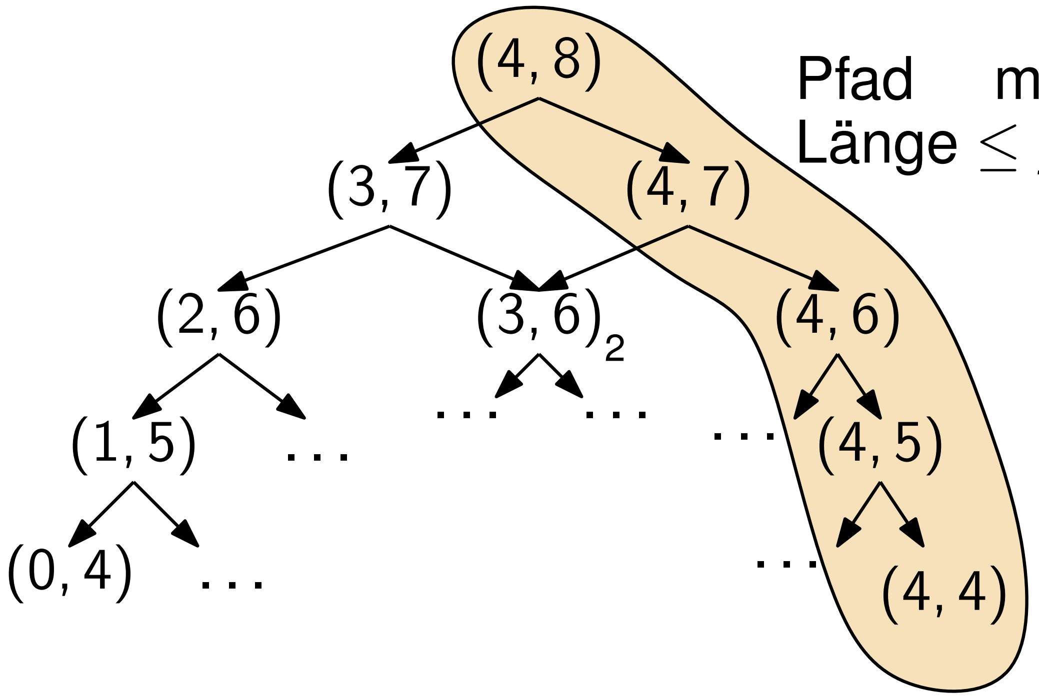
Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

  if $x = 0 \vee x = y$ **then**

    return 1

  return algOne(x-1, y-1) + **algOne**(x, y-1)



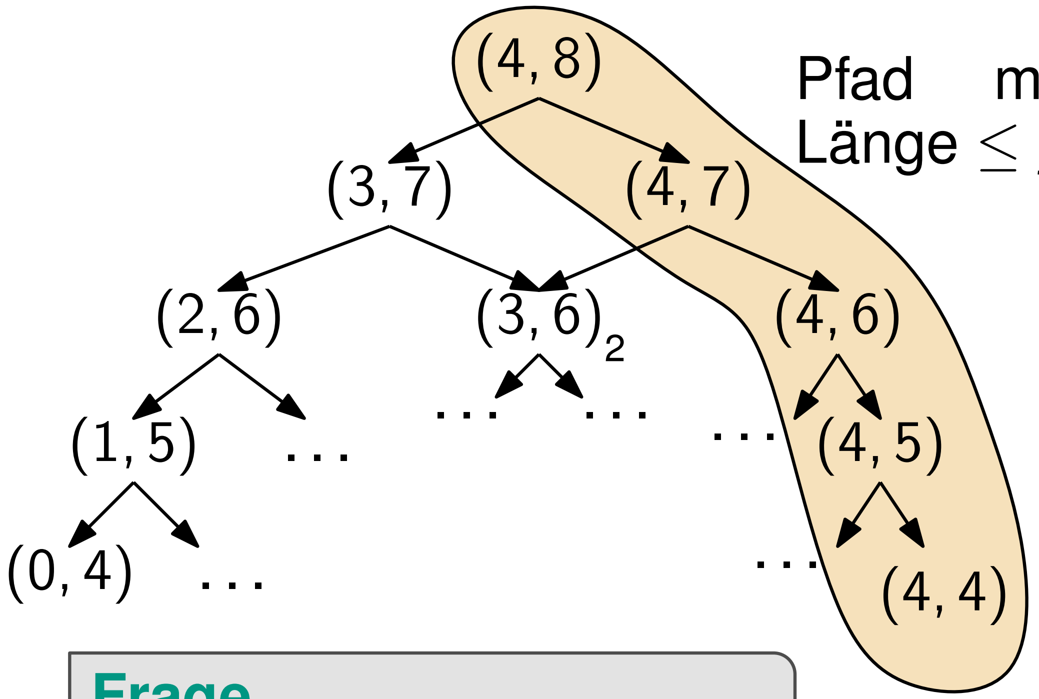
Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\leq 2 \cdot 2^y$
- Für „Problemgröße“ $n := y$: Laufzeit in $O(2^n)$

Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```



Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\leq 2 \cdot 2^y$
- Für „Problemgröße" $n := y$: Laufzeit in $O(2^n)$

Frage
Ist diese Analyse gut?

Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

$(1, y)$

Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\leq 2 \cdot 2^y$
- Für „Problemgröße" $n := y$: Laufzeit in $O(2^n)$

Frage
Ist diese Analyse gut?

Ungenaue obere Schranke ALGONE

algOne$(x\colon \mathbb{N}, y\colon \mathbb{N})$ // Annahme: initial $x \leq y$

  if $x = 0 \vee x = y$ **then**

    return 1

  return algOne(x-1, y-1) + **algOne**(x, y-1)

$$\begin{tikzcd}[column sep=small] & (1, y) \arrow[dl] \arrow[dr] & \\ (0, y\text{-}1) & & (1, y\text{-}1) \end{tikzcd}$$

Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\leq 2 \cdot 2^y$
- Für „Problemgröße“ $n := y$: Laufzeit in $O(2^n)$

Frage
Ist diese Analyse gut?

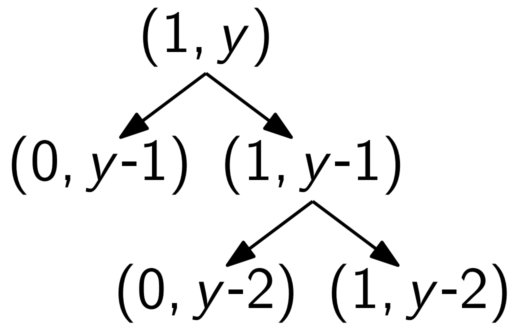
Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \le y$

 if $x = 0 \lor x = y$ **then**

  return 1

 return algOne(x-1, y-1) + **algOne**(x, y-1)



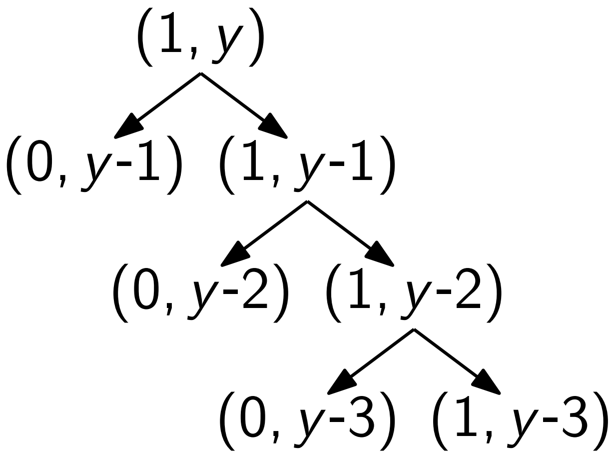
Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\le 2 \cdot 2^y$
- Für „Problemgröße“ $n := y$: Laufzeit in $O(2^n)$

Frage
Ist diese Analyse gut?

Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \vee x = y$ **then**
    return 1
  return **algOne**(x-1, y-1) + **algOne**(x, y-1)



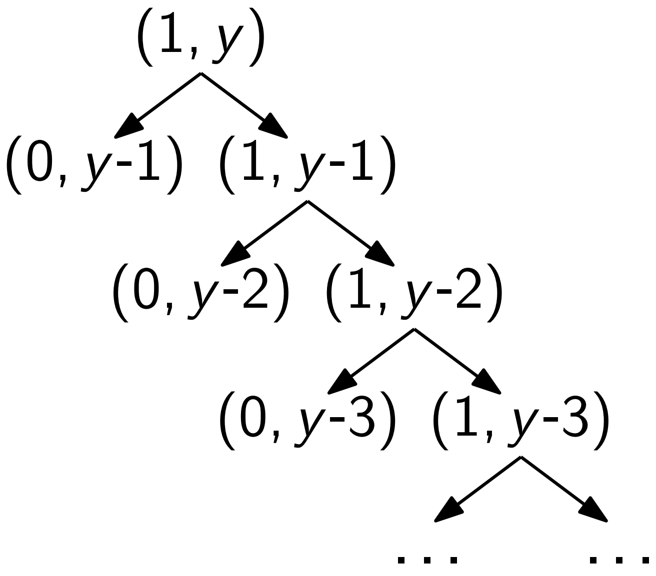
Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\leq 2 \cdot 2^y$
- Für „Problemgröße“ $n := y$: Laufzeit in $O(2^n)$

Frage
Ist diese Analyse gut?

Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \lor x = y$ **then**
    return 1
  return algOne(x-1, y-1) + **algOne**(x, y-1)



Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\leq 2 \cdot 2^y$
- Für „Problemgröße“ $n := y$: Laufzeit in $O(2^n)$

Frage
Ist diese Analyse gut?

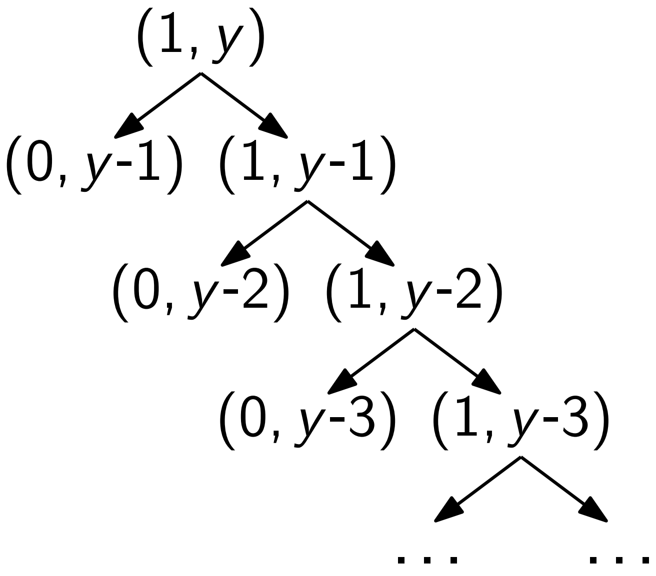
Ungenaue obere Schranke ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

  if $x = 0 \vee x = y$ **then**

    return 1

  return algOne(x-1, y-1) + **algOne**(x, y-1)



Beobachtung

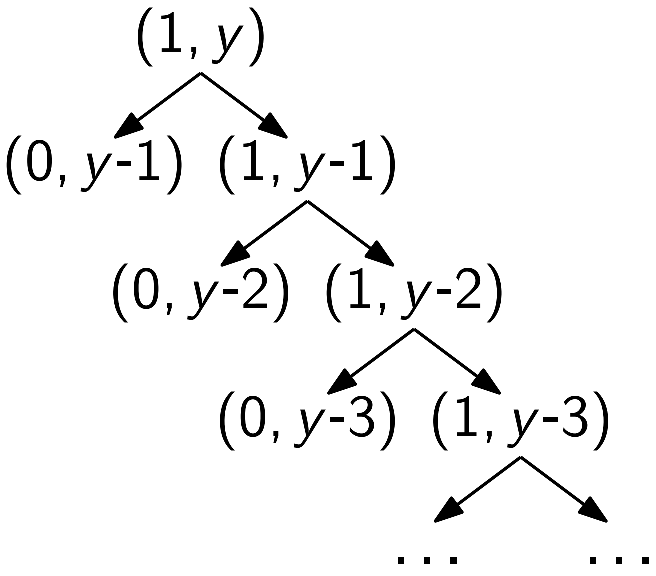
- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\leq 2 \cdot 2^y$
- Für „Problemgröße“ $n := y$: Laufzeit in $O(2^n)$

Frage
Ist diese Analyse gut?

Nicht für kleine x

Ungenaue obere Schranke ALGONE

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
  if x = 0 ∨ x = y then
    return 1
  return algOne(x-1, y-1) + algOne(x, y-1)
```



Beobachtung

- y sinkt in jedem Aufruf um 1
- Länge der Pfade im Rekursionsbaum maximal y
- Pro Aufruf: zwei Rekursive Aufrufe
- #Knoten in Rekursionsbaum $\leq 2 \cdot 2^y$
- Für „Problemgröße“ $n := y$: Laufzeit in $O(2^n)$

Frage
Ist diese Analyse gut?

Nicht für kleine x, große x

Genaue Analyse ALGONE

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

if $x = 0 \vee x = y$ **then**

  return 1

return algOne(x-1, y-1) + **algOne**(x, y-1)

Schritt 1

- Kosten ausrechnen. . .

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

x,y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2		1	3	6	10	15	21	28	36	45	55	66	78	91	105
3			1	4	10	20	35	56	84	120	165	220	286	364	455
4				1	5	15	35	70	126	210	330	495	715	1001	1365
5					1	6	21	56	126	252	462	792	1287	2002	3003
6						1	7	28	84	210	462	924	1716	3003	5005
7							1	8	36	120	330	792	1716	3432	6435
8								1	9	45	165	495	1287	3003	6435
9									1	10	55	220	715	2002	5005
10										1	11	66	286	1001	3003
11											1	12	78	364	1365
12												1	13	91	455
13													1	14	105
14														1	15
15															1

So

- K

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \lor x = y$ **then**
    return 1
  return **algOne**(x-1, y-1) + **algOne**(x, y-1)

Schritt 1

- Kosten ausrechnen. . .

Genaue Analyse ALGONE

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
  if x = 0 ∨ x = y then
    return 1
  return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

x,y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2		1	3	6	10	15	21	28	36	45	55	66	78	91	105
3			1	4	10	20	35	56	84	120	165	220	286	364	455
4				1	5	15	35	70	126	210	330	495	715	1001	1365
5					1	6	21	56	126	252	462	792	1287	2002	3003
6						1	7	28	84	210	462	924	1716	3003	5005
7							1	8	36	120	330	792	1716	3432	6435
8								1	9	45	165	495	1287	3003	6435
9									1	10	55	220	715	2002	5005
10										1	11	66	286	1001	3003
11											1	12	78	364	1365
12												1	13	91	455
13													1	14	105
14														1	15
15															1

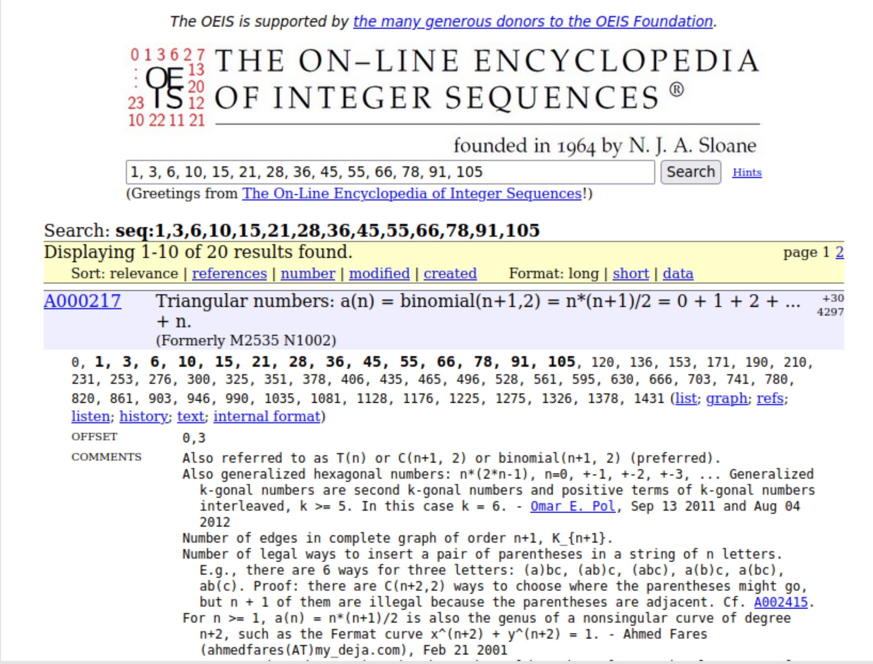
So

- K
- Z

Genaue Analyse ALGONE

algOne$(x\colon \mathbb{N}, y\colon \mathbb{N})$ // Annahme: initial $x \le y$

x,y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2		1	3	6	10	15	21	28	36	45	55	66	78	91	105
3			1	4	10	20	35	56	84	120	165	220	286	364	455
4									26	210	330	495	715	1001	1365
5									26	252	462	792	1287	2002	3003
6									84	210	462	924	1716	3003	5005
7									36	120	330	792	1716	3432	6435
8									9	45	165	495	1287	3003	6435
9									1	10	55	220	715	2002	5005
10										1	11	66	286	1001	3003
11											1	12	78	364	1365
12												1	13	91	455
13													1	14	105
14														1	15
15															1


The OEIS is supported by the many generous donors to the OEIS Foundation.

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES ®

founded in 1964 by N. J. A. Sloane

1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105 Search Hints

(Greetings from The On-Line Encyclopedia of Integer Sequences!)

Search: **seq:1,3,6,10,15,21,28,36,45,55,66,78,91,105**

Displaying 1-10 of 20 results found. page 1 2

Sort: relevance | references | number | modified | created Format: long | short | data

A000217 Triangular numbers: a(n) = binomial(n+1,2) = n*(n+1)/2 = 0 + 1 + 2 + ... + n. +30 4297

(Formerly M2535 N1002)

0, **1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105**, 120, 136, 153, 171, 190, 210, 231, 253, 276, 300, 325, 351, 378, 406, 435, 465, 496, 528, 561, 595, 630, 666, 703, 741, 780, 820, 861, 903, 946, 990, 1035, 1081, 1128, 1176, 1225, 1275, 1326, 1378, 1431 (list; graph; refs; listen; history; text; internal format)

OFFSET 0,3

COMMENTS Also referred to as T(n) or C(n+1, 2) or binomial(n+1, 2) (preferred).

Also generalized hexagonal numbers: n*(2*n-1), n=0, +-1, +-2, +-3, ... Generalized k-gonal numbers are second k-gonal numbers and positive terms of k-gonal numbers interleaved, k >= 5. In this case k = 6. - Omar E. Pol, Sep 13 2011 and Aug 04 2012

Number of edges in complete graph of order n+1, K_{n+1}.

Number of legal ways to insert a pair of parentheses in a string of n letters. E.g., there are 6 ways for three letters: (a)bc, (ab)c, (abc), a(b)c, a(bc), ab(c). Proof: there are C(n+2,2) ways to choose where the parentheses might go, but n + 1 of them are illegal because the parentheses are adjacent. Cf. A002415.

For n >= 1, a(n) = n*(n+1)/2 is also the genus of a nonsingular curve of degree n+2, such as the Fermat curve x^(n+2) + y^(n+2) = 1. - Ahmed Fares (ahmedfares(AT)my_deja.com), Feb 21 2001

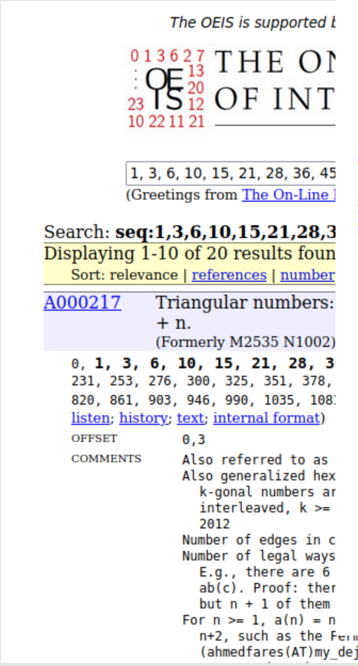
Genaue Analyse ALGONE

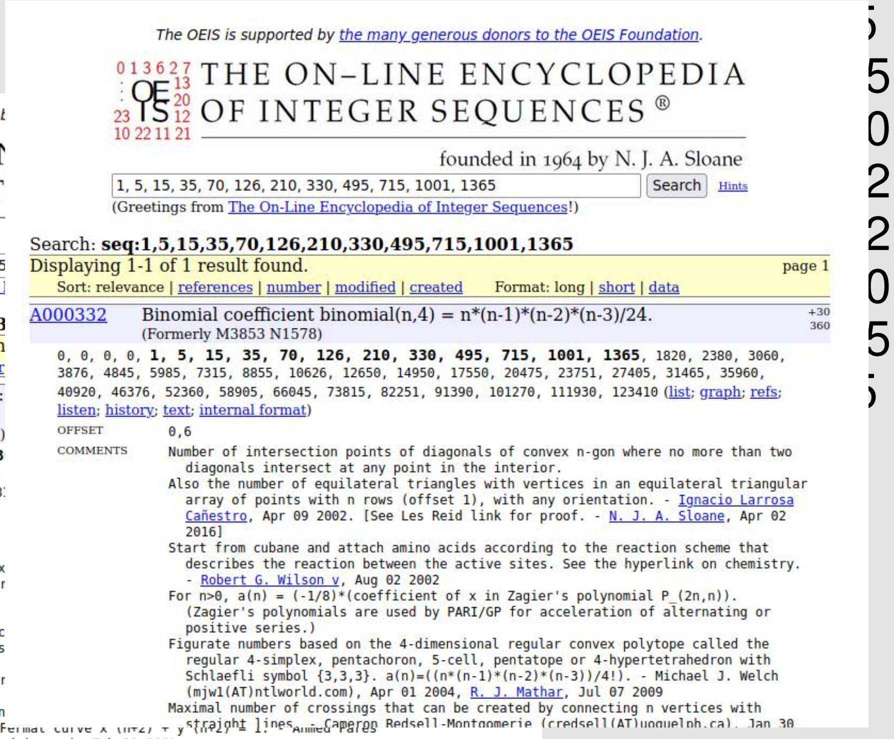
algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

x,y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2		1	3	6								66	78	91	105
3			1	4								220	286	364	455
4												495	715	1001	1365
5												792	1287	2002	3003
6												924	1716	3003	5005
7												792	1716	3432	6435
8												495	1287	3003	6435
9												220	715	2002	5005
10												66	286	1001	3003
11												12	78	364	1365
12												1	13	91	455
13													1	14	105
14														1	15
15															1

So

- K
- Z





Genaue Analyse ALGONE

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Genaue Analyse ALGONE

Karlsruher Institut für Technologie

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

Genaue Analyse ALGONE

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen. . .
- Zahlen anstarren

Schritt 2

- Vermutung bilden:

Genaue Analyse AlgOne

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen. . .
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$

Genaue Analyse ALGONE

Karlsruher Institut für Technologie

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$
  if $x = 0 \vee x = y$ **then**
    return 1
  return **algOne**(x-1, y-1) + **algOne**(x, y-1)

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Beweis per Induktion

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Beweis per Induktion

- für $i \geq 0$: $T(i,i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x,y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen…
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$T(x, y) = T(x-1, y-1) + T(x, y-1) + 1$$

Genaue Analyse ALGONE

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$
\begin{aligned}
T(x, y) &= T(x-1, y-1) + T(x, y-1) + 1 \\
&= 2\binom{y-1}{x-1} - 1 + 2\binom{y-1}{x} - 1 + 1
\end{aligned}
$$

Genaue Analyse AlgOne

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen…
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$\begin{aligned} T(x, y) &= T(x-1, y-1) + T(x, y-1) + 1 \\ &= 2\binom{y-1}{x-1} - 1 + 2\binom{y-1}{x} - 1 + 1 \\ &= 2\left(\binom{y-1}{x-1} + \binom{y-1}{x}\right) - 1 \end{aligned}$$

Genaue Analyse ALGONE

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$
\begin{aligned}
T(x, y) &= T(x-1, y-1) + T(x, y-1) + 1 \\
&= 2\binom{y-1}{x-1} - 1 + 2\binom{y-1}{x} - 1 + 1 \\
&= 2\left(\binom{y-1}{x-1} + \binom{y-1}{x}\right) - 1 \\
&= 2\binom{y}{x} - 1
\end{aligned}
$$

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

if $x = 0 \lor x = y$ then

Eigenschaften [Bearbeiten | Quelltext bearbeiten]

Wird außer k auch n auf nichtnegative ganze Zahlen eingeschränkt, so gilt:

- $\binom{n}{k}$ ist stets eine nichtnegative ganze Zahl. Ist $k \leq n$, so ist $\binom{n}{k} \geq 1$, anderenfalls ist $\binom{n}{k} = 0$.
- $\binom{n}{0} = 1 = \binom{n}{n}$
- $\binom{n}{1} = n = \binom{n}{n-1}$
- $\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$
- $\binom{n}{k} = \frac{n}{k} \cdot \binom{n-1}{k-1} \Leftrightarrow k \cdot \binom{n}{k} = n \cdot \binom{n-1}{k-1}$
- $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$. Für $n = k$ ist der rechte Summand $\binom{n}{k+1} = 0$.

S

S

- $T(x, y) \leq 2\binom{y}{x} - 1$

- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$

g.: Vermutung gilt für alle kleineren
erte von x und y

$$y) = T(x-1, y-1) + T(x, y-1) + 1$$
$$= 2\binom{y-1}{x-1} - 1 + 2\binom{y-1}{x} - 1 + 1$$
$$= 2\left(\binom{y-1}{x-1} + \binom{y-1}{x}\right) - 1$$
$$= 2\binom{y}{x} - 1$$

Genaue Analyse ALGONE

Karlsruher Institut für Technologie

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$
\begin{aligned}
T(x, y) &= T(x-1, y-1) + T(x, y-1) + 1 \\
&= 2\binom{y-1}{x-1} - 1 + 2\binom{y-1}{x} - 1 + 1 \\
&= 2\left(\binom{y-1}{x-1} + \binom{y-1}{x}\right) - 1 \\
&= 2\binom{y}{x} - 1
\end{aligned}
$$

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$\begin{aligned} T(x, y) &= T(x-1, y-1) + T(x, y-1) + 1 \\ &= 2\binom{y}{x} - 1 \end{aligned}$$

Genaue Analyse ALGONE

Karlsruher Institut für Technologie

```
algOne(x: ℕ, y: ℕ) // Annahme: initial x ≤ y
  if x = 0 ∨ x = y then
    return 1
  return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$\begin{aligned} T(x, y) &= T(x-1, y-1) + T(x, y-1) + 1 \\ &= 2\binom{y}{x} - 1 \end{aligned}$$

- Moment mal...

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
    if x = 0 ∨ x = y then
        return 1
    return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$T(x, y) = T(x-1, y-1) + T(x, y-1) + 1$$
$$= 2\binom{y}{x} - 1$$

- Moment mal...
  - $\binom{y}{x} = \binom{y-1}{x-1} + \binom{y-1}{x} \ldots ?$

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$T(x, y) = T(x-1, y-1) + T(x, y-1) + 1$$
$$= 2\binom{y}{x} - 1$$

- Moment mal...
  - $\binom{y}{x} = \binom{y-1}{x-1} + \binom{y-1}{x}$...?
  - Das ist doch genau **algOne**!

Genaue Analyse ALGONE

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$\begin{aligned} T(x, y) &= T(x-1, y-1) + T(x, y-1) + 1 \\ &= 2\binom{y}{x} - 1 \end{aligned}$$

- Moment mal...
  - $\binom{y}{x} = \binom{y-1}{x-1} + \binom{y-1}{x} \ldots ?$
  - Das ist doch genau **algOne**!
  - $T(x, y) \in \Theta(\binom{y}{x})$

Genaue Analyse AlgOne

algOne(x: $\mathbb{N}$, y: $\mathbb{N}$) // Annahme: initial $x \leq y$

```
if x = 0 ∨ x = y then
    return 1
return algOne(x-1, y-1) + algOne(x, y-1)
```

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:
  - $T(x, y) \leq 2\binom{y}{x} - 1$
- Vermutung beweisen

Beweis per Induktion

- für $i \geq 0$: $T(i, i) = 2\binom{i}{i} - 1 = 2 - 1 = 1$
- Ang.: Vermutung gilt für alle kleineren Werte von x und y

$$\begin{aligned} T(x, y) &= T(x-1, y-1) + T(x, y-1) + 1 \\ &= 2\binom{y}{x} - 1 \end{aligned}$$

- Moment mal...
  - $\binom{y}{x} = \binom{y-1}{x-1} + \binom{y-1}{x} \ldots ?$
  - Das ist doch genau **algOne**!
  - $T(x, y) \in \Theta(\binom{y}{x})$

:-)

Motivation: Datenstrukturen

Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher

Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher

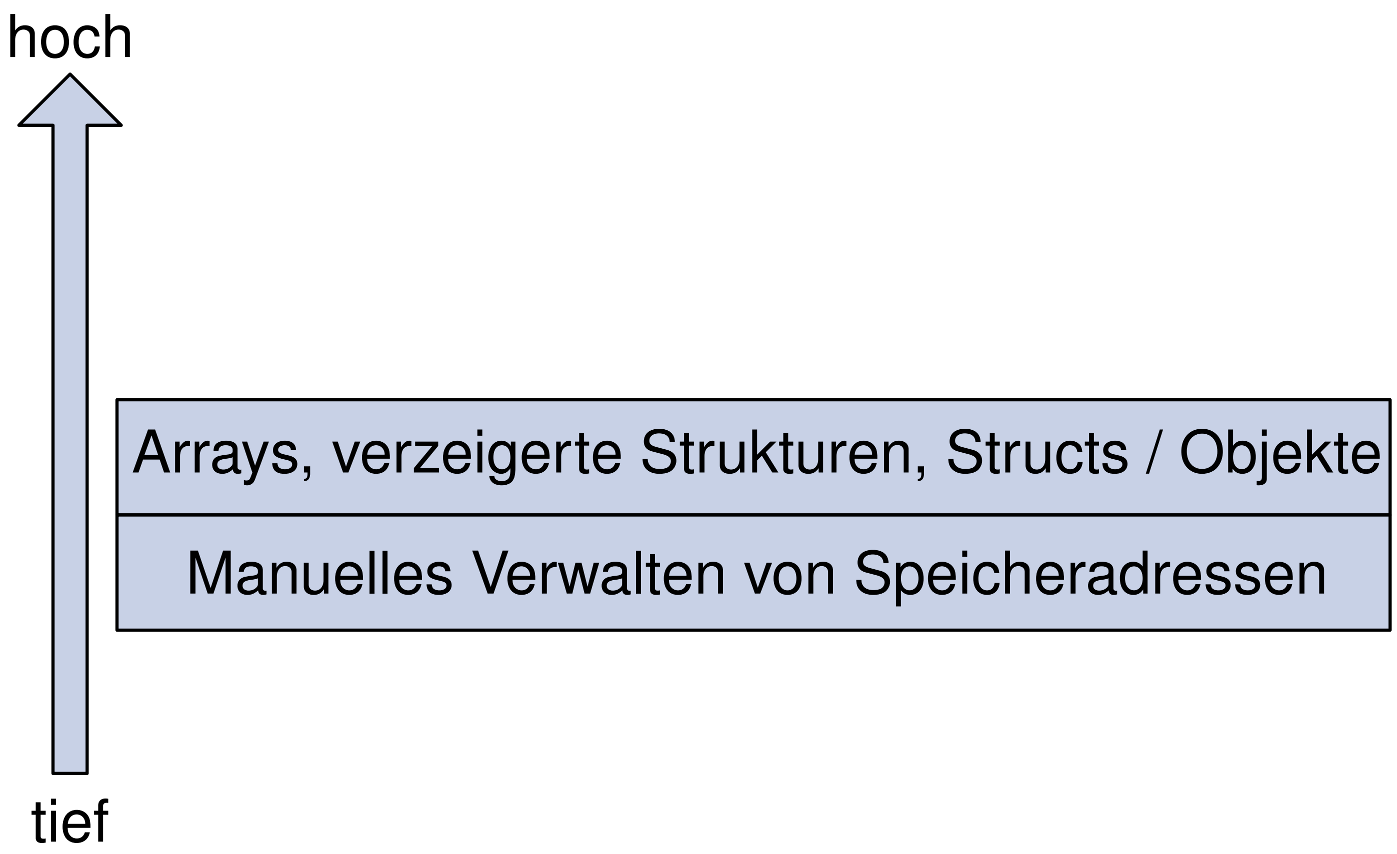
hoch

Manuelles Verwalten von Speicheradressen

tief

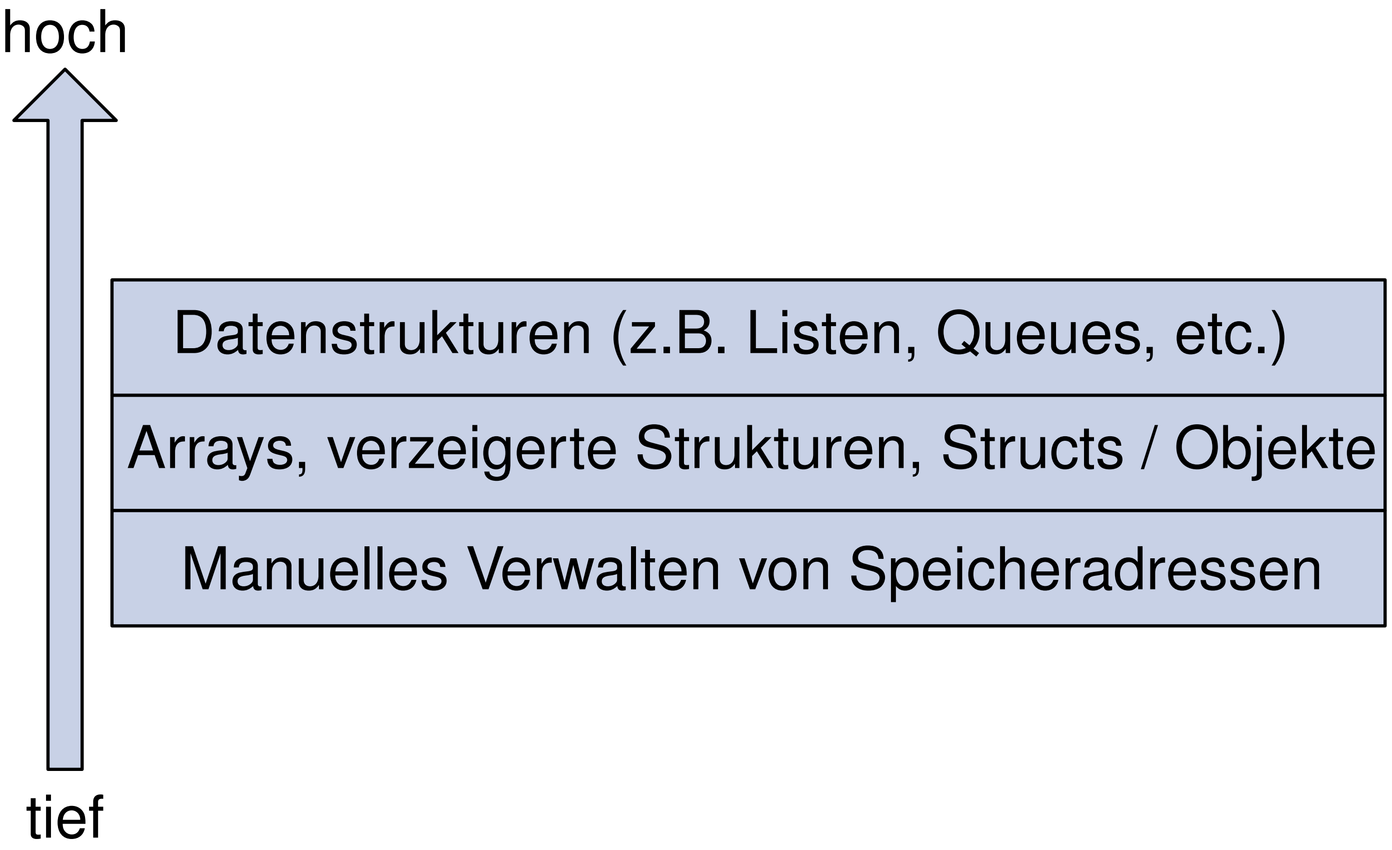
Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher



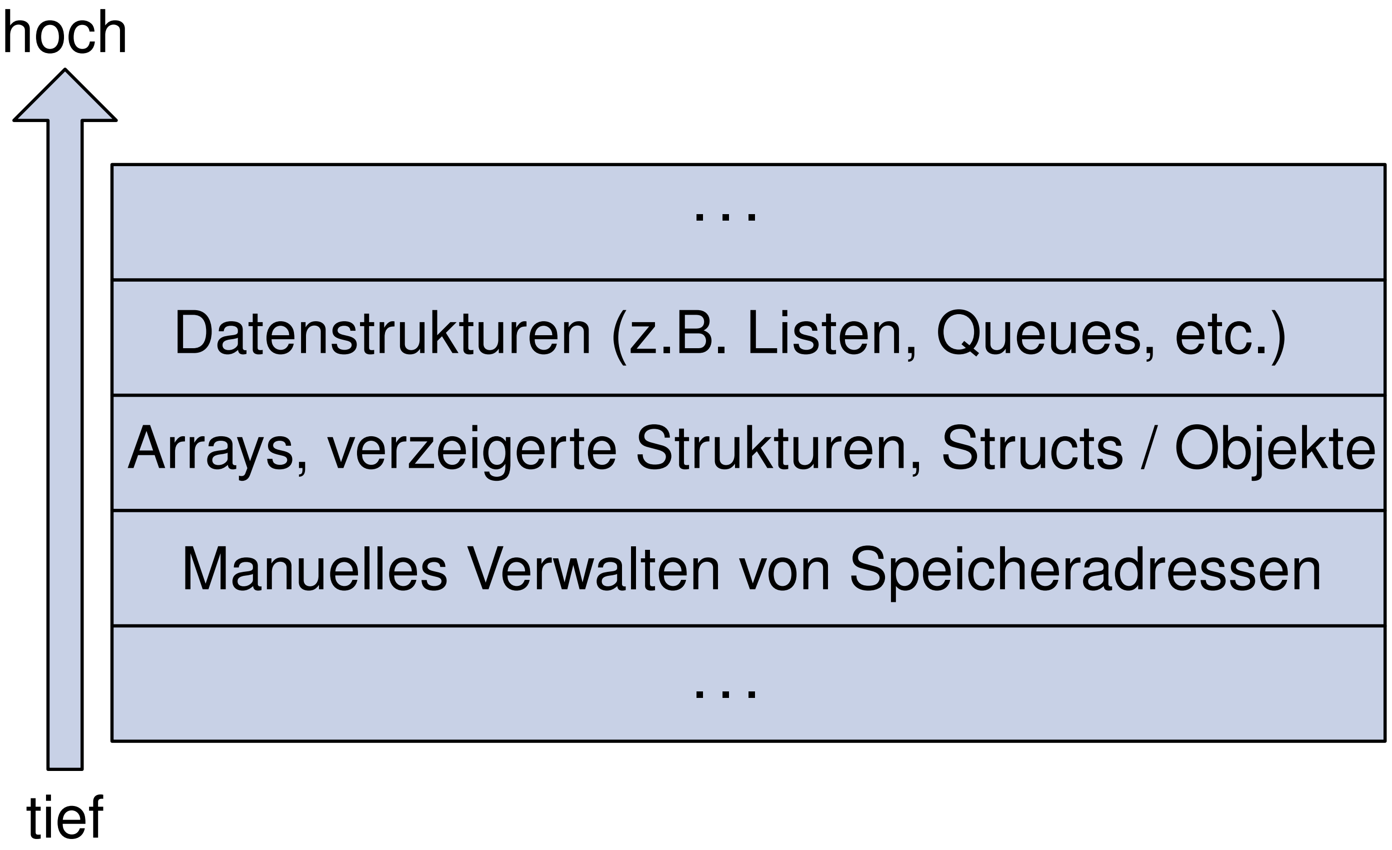
Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher



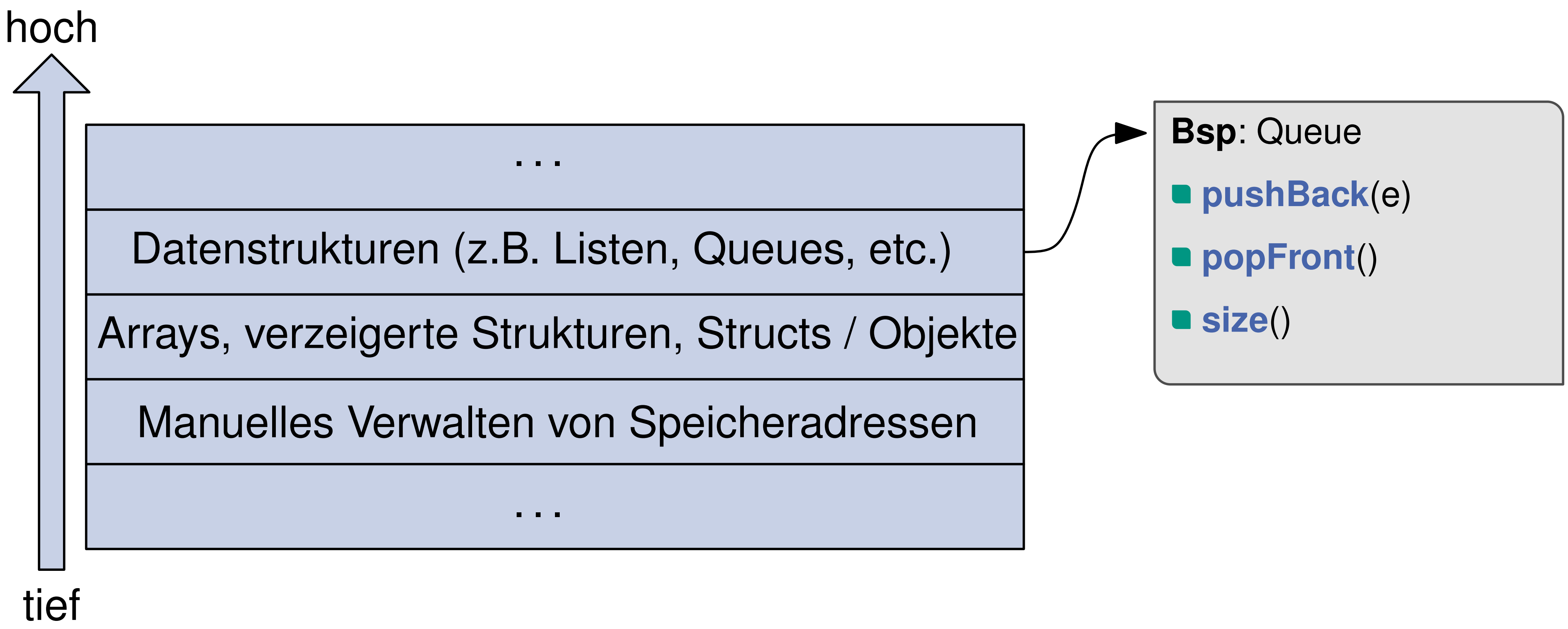
Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher



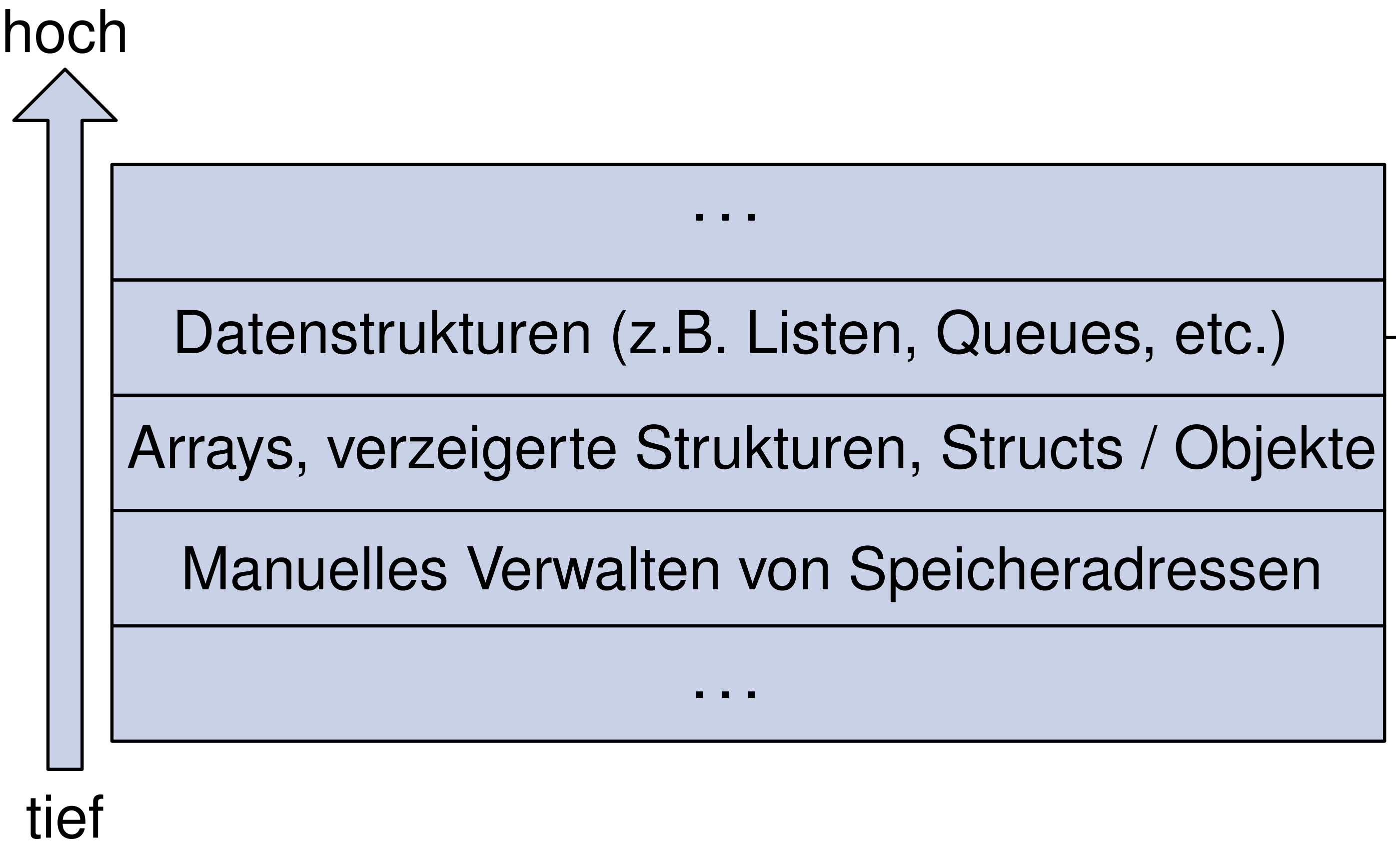
Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher



Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher



Bsp: Queue

- **pushBack**(e) $O(1)$
- **popFront**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste

Motivation: Amortisierte Analyse

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

Beispiel: unbeschränktes Array

4	48		

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

Beispiel: unbeschränktes Array

4	48	89	

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

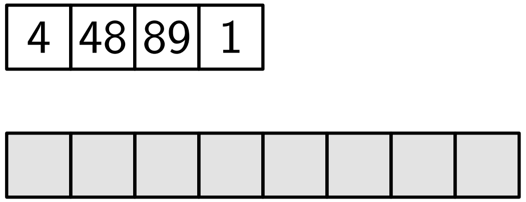
Beispiel: unbeschränktes Array

4	48	89	1

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

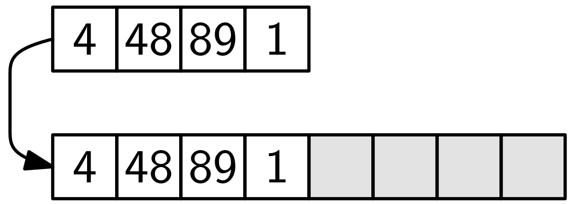
Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

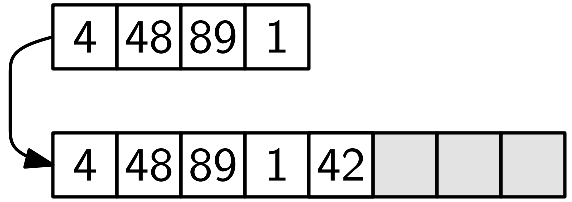
Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

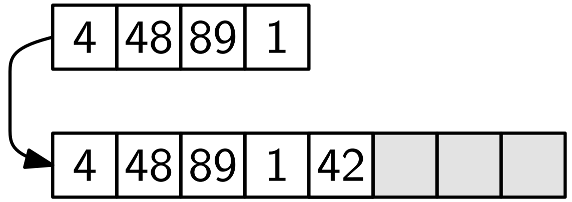
Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
  - z.B.: Operation meist günstig, seltener teurer

Beispiel: unbeschränktes Array



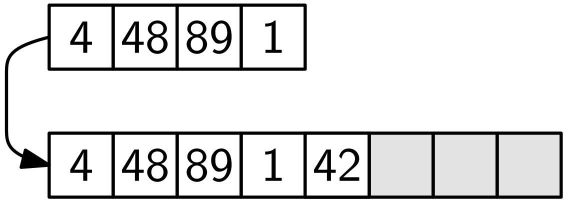
Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
  - z.B.: Operation meist günstig, seltener teurer

Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

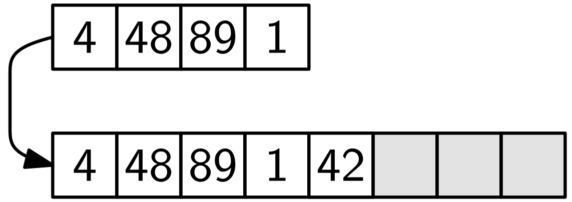
- Laufzeit von Operationen auf DS nicht immer gleich
  - z.B.: Operation meist günstig, seltener teurer

Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

Verschiedene Techniken zur Analyse

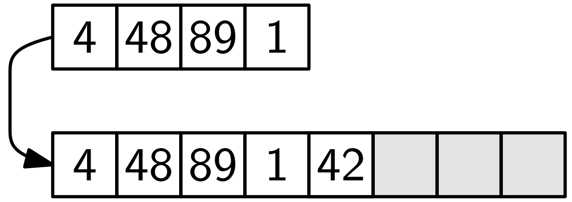
Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
  - z.B.: Operation meist günstig, seltener teurer

Beispiel: unbeschränktes Array



Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

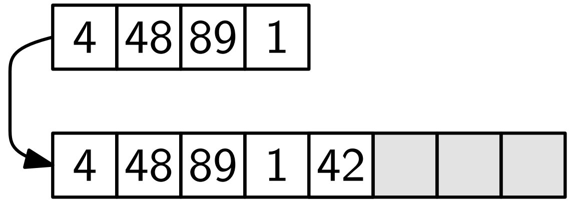
Verschiedene Techniken zur Analyse

- Aggregation

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
  - z.B.: Operation meist günstig, seltener teurer

Beispiel: unbeschränktes Array



Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

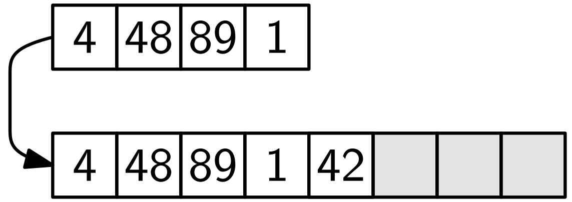
Verschiedene Techniken zur Analyse

- Aggregation
- Charging

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
  - z.B.: Operation meist günstig, seltener teurer

Beispiel: unbeschränktes Array



Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

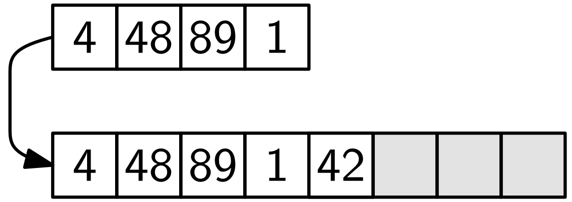
Verschiedene Techniken zur Analyse

- Aggregation
- Charging
- Konto

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
  - z.B.: Operation meist günstig, seltener teurer

Beispiel: unbeschränktes Array



Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

Verschiedene Techniken zur Analyse

- Aggregation
- Charging
- Konto
- Potential

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits

0	0	0	0	0	0	0	0

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	1

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	1

0	0	0	0	0	0	1	0

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	1

0	0	0	0	0	0	1	0

0	0	0	0	0	0	1	1

Beispiel: Binärzähler

Karlsruher Institut für Technologie

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	1

0	0	0	0	0	0	1	0

0	0	0	0	0	0	1	1

0	0	0	0	0	1	0	0

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Karlsruher Institut für Technologie

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

. . .

Beispiel: Binärzähler

Karlsruher Institut für Technologie

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Karlsruher Institut für Technologie

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Karlsruher Institut für Technologie

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee
- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus
- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	$\boxed{0}$
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	$\boxed{0}$	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

$\ldots$

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus in Pseudocode

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

$\dots$

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus in Pseudocode

increment()

$i := 0$

while $A[i] = 1$ **do**

  $A[i] := 0$

  i := i+1

$A[i] := 1$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus in Pseudocode

increment()

$i := 0$
while $A[i] = 1$ **do**
  $A[i] := 0$
  i := i+1
$A[i] := 1$

Laufzeit?

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i}$$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Karlsruher Institut für Technologie

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \underbrace{\sum_{i=0}^{\infty} \frac{1}{2^i} = 2n}_{\boxed{}}$$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

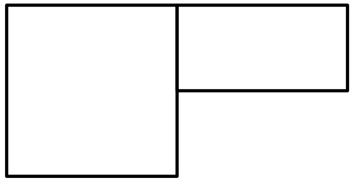
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

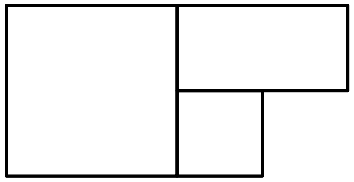
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \underbrace{\sum_{i=0}^{\infty} \frac{1}{2^i}}_{} = 2n$$



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

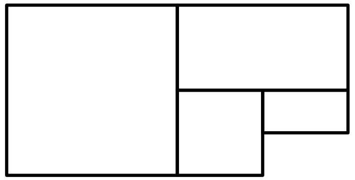
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

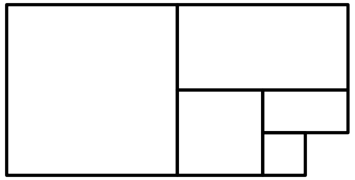
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

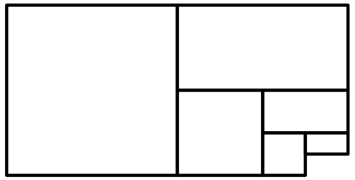
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \underbrace{\sum_{i=0}^{\infty} \frac{1}{2^i}} = 2n$$



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

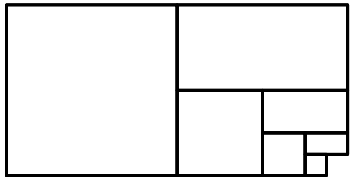
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

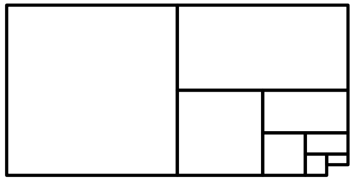
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

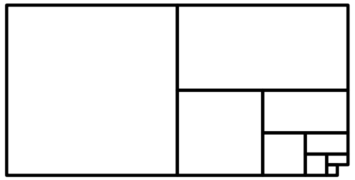
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \underbrace{\sum_{i=0}^{\infty} \frac{1}{2^i}} = 2n$$



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Karlsruher Institut für Technologie

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

D.h.: pro Operation nur 2 Bit-Flips

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i-ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

D.h.: pro Operation nur 2 Bit-Flips

Somit: **amortisierte Kosten** in $O(1)$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse mittels Charging

Analyse mittels Charging

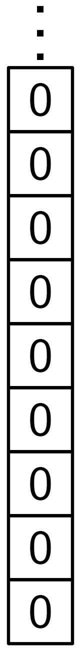
Idee

- teure Operationen legen Kosten um auf günstige Operationen

Analyse mittels Charging

Idee

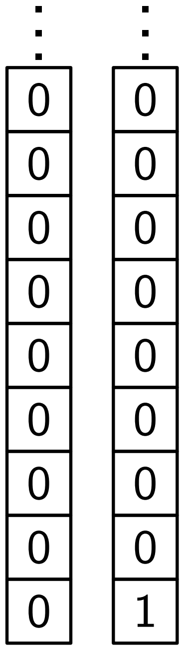
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

⋮	⋮	⋮
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	1
0	1	0

Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

⋮	⋮	⋮	⋮
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	1	1
0	1	0	1

Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

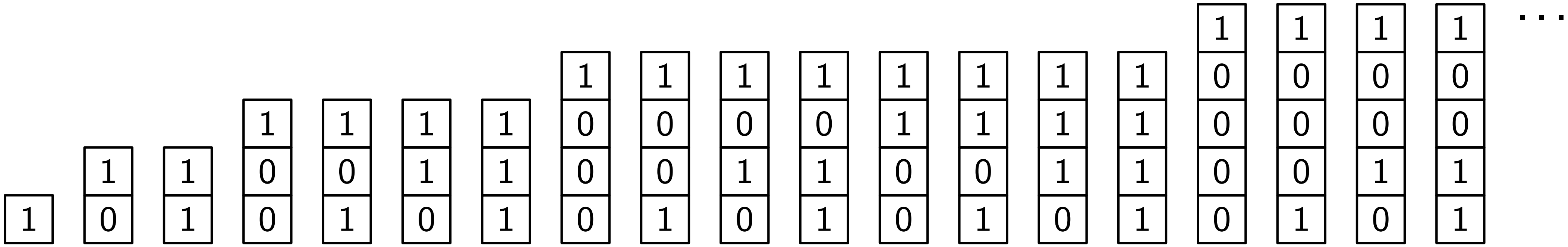
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

…

Analyse mittels Charging

Idee

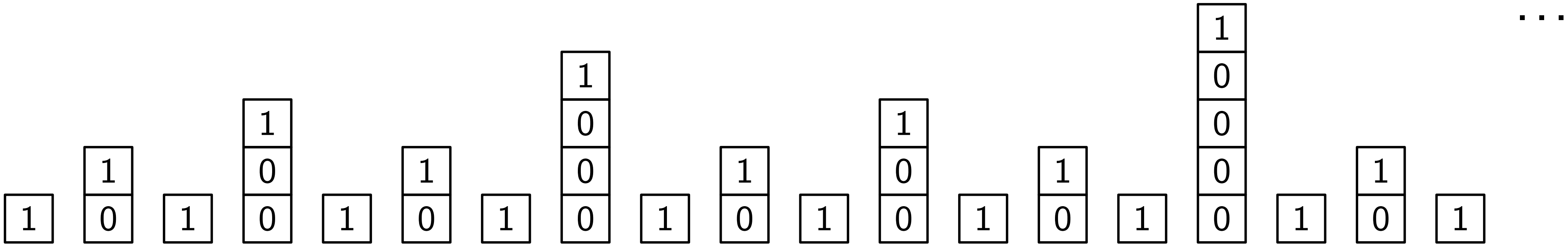
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

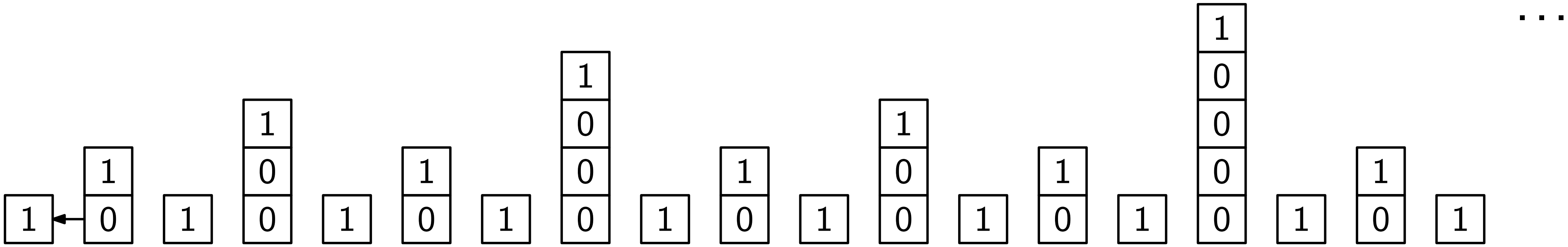
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

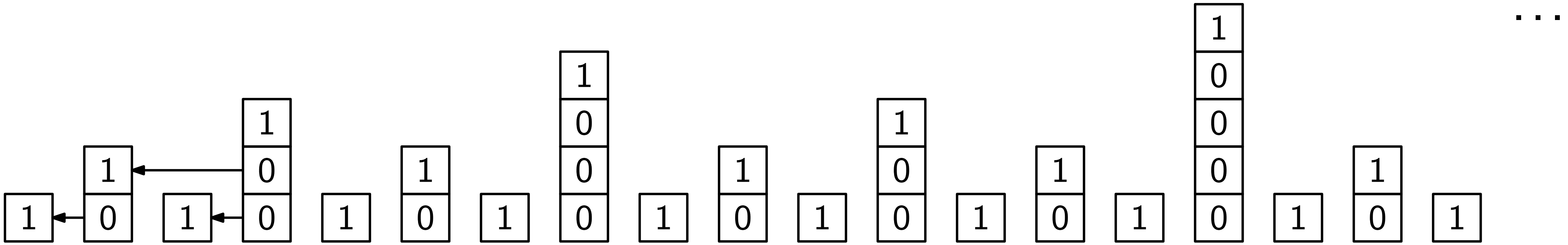
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

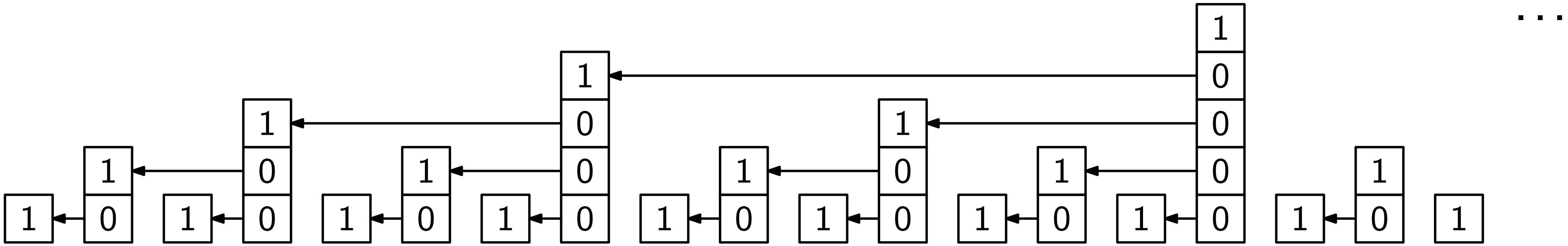
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

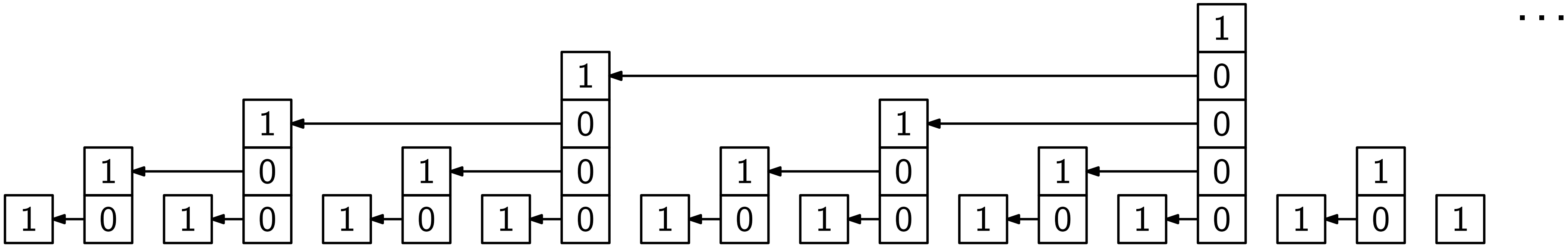


Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:



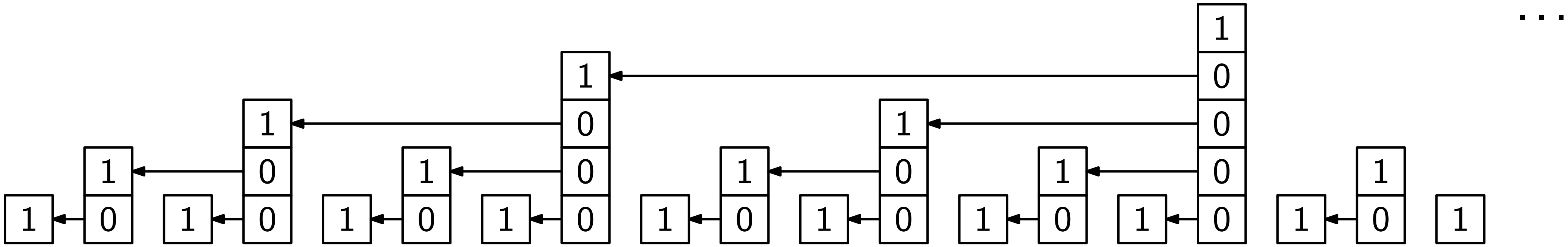
Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:

- lege Kosten von 10-flip auf letzte Operation welche zuvor 01-flip ausgeführt hat



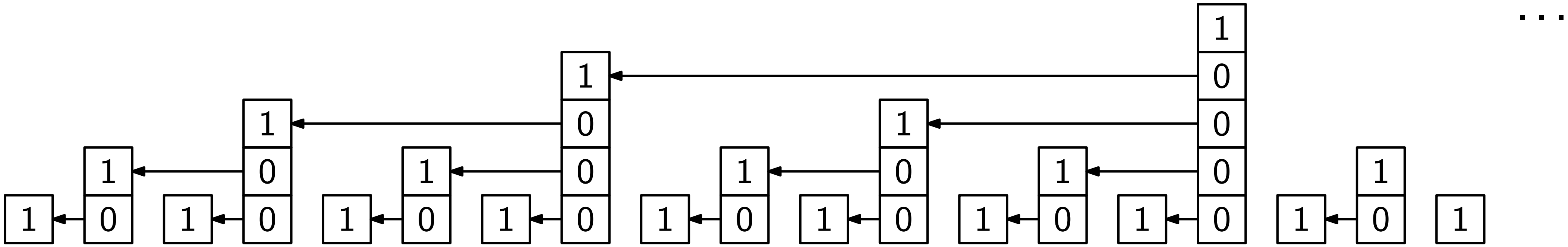
Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:

- lege Kosten von 10-flip auf letzte Operation welche zuvor 01-flip ausgeführt hat
  - das geht für jeden 10-flip



Analyse mittels Charging

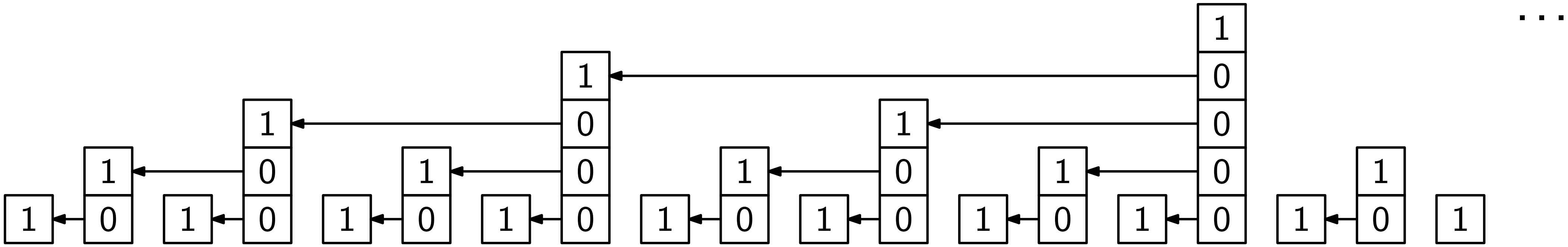
Karlsruher Institut für Technologie

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:

- lege Kosten von 10-flip auf letzte Operation welche zuvor 01-flip ausgeführt hat
  - das geht für jeden 10-flip
- Pro Operation übrig:



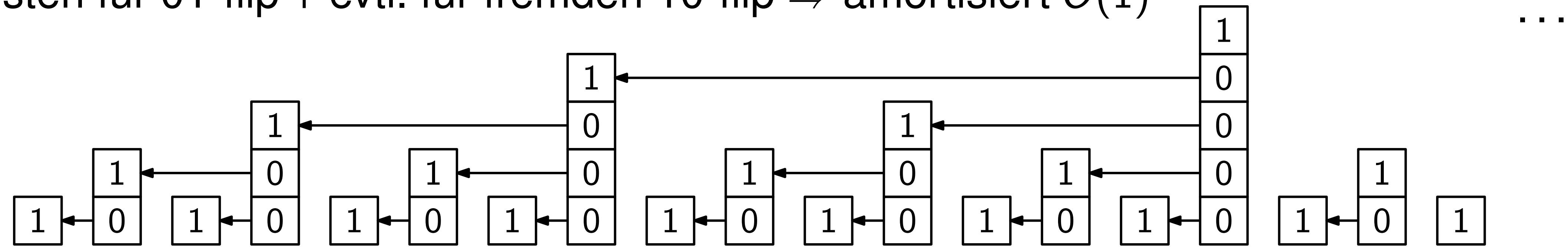
Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:

- lege Kosten von 10-flip auf letzte Operation welche zuvor 01-flip ausgeführt hat
  - das geht für jeden 10-flip
- Pro Operation übrig:
  - Kosten für 01-flip + evtl. für fremden 10-flip $\Rightarrow$ amortisiert $O(1)$



Analyse: Potentialmethode

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i

Analyse: Potentialmethode

Karlsruher Institut für Technologie

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

tatsächliche
Kosten

Analyse: Potentialmethode

Karlsruher Institut für Technologie

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\text{Anstieg des Potenzials}}$

tatsächliche Kosten

Analyse: Potentialmethode

Karlsruher Institut für Technologie

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\sum_{i=1}^{n} \hat{c}_i$$

Analyse: Potentialmethode

Karlsruher Institut für Technologie

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} (c_i) + \sum_{i=1}^{n} (\Phi(D_i) - \Phi(D_{i-1}))$$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} (c_i) + \sum_{i=1}^{n} (\Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} (c_i) + \Phi(D_n) - \Phi(D_0)$$

Analyse: Potentialmethode

Karlsruher Institut für Technologie

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) &= \sum_{i=1}^{n} (c_i) + \sum_{i=1}^{n} (\Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^{n} (c_i) + \Phi(D_n) - \Phi(D_0)\end{aligned}$$

Falls $\Phi(D_n) \geq \Phi(D_0)$: amortisierte Kosten obere Schranke für tatsächliche Kosten

Analyse: Potentialmethode

Karlsruher Institut für Technologie

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) &= \sum_{i=1}^{n} (c_i) + \sum_{i=1}^{n} (\Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^{n} (c_i) + \Phi(D_n) - \Phi(D_0)\end{aligned}$$

Falls $\Phi(D_n) \geq \Phi(D_0)$: amortisierte Kosten obere Schranke für tatsächliche Kosten

$\Rightarrow$ fordere $\Phi(D_i) \geq \Phi(D_0)$ f.a. $i > 0$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
  - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} (c_i) + \sum_{i=1}^{n} (\Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} (c_i) + \Phi(D_n) - \Phi(D_0)$$

Falls $\Phi(D_n) \geq \Phi(D_0)$: amortisierte Kosten obere Schranke für tatsächliche Kosten

$\Rightarrow$ fordere $\Phi(D_i) \geq \Phi(D_0)$ f.a. $i > 0$ oder: $\Phi(D_0) = 0$ und $\Phi(D_i) \geq 0$ f.a. $i > 0$

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

$\dots$

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$

gültige Pot.fun. :)

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tätsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tätsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun.* :)
- tätsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$
- amort. Kosten: $\hat{c}_i =$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tätsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$
- amort. Kosten: $\hat{c}_i =$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tätsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$
- amort. Kosten: $\hat{c}_i = 2 \cdot \#01\text{-flips}$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tätsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$
- amort. Kosten: $\hat{c}_i = 2 \cdot \#01\text{-flips} \leq 2$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	0	0	0

...

Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Beispiel 2: Stacks und Queues

Beispiel 2: Stacks und Queues

Wiederholung

Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste

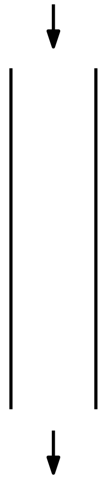
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



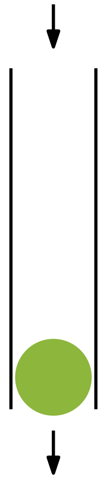
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



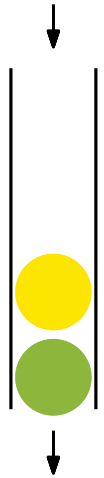
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



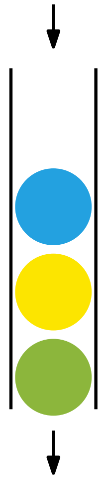
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



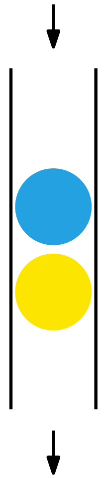
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



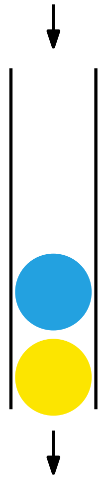
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



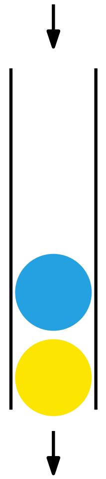
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste

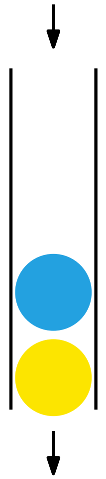
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

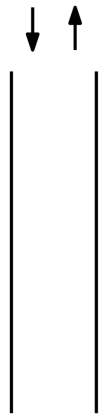
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



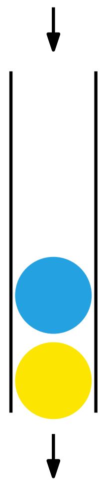
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

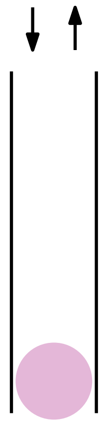
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



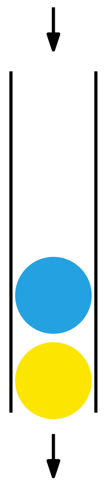
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

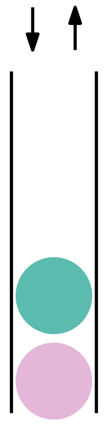
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



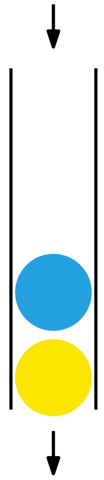
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

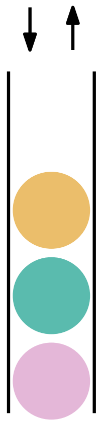
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



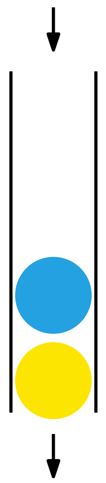
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

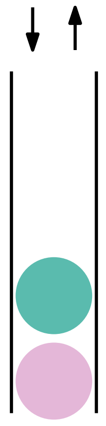
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



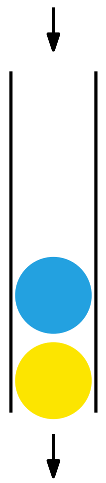
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

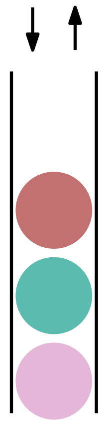
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



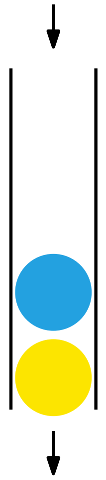
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

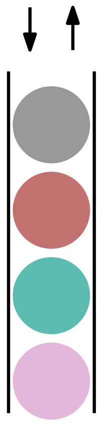
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



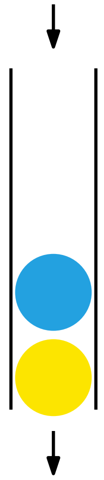
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

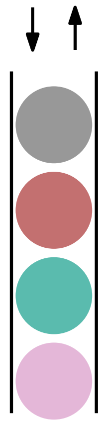
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Frage

Beispiel 2: Stacks und Queues

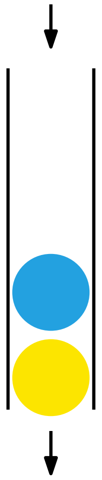
Karlsruher Institut für Technologie

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

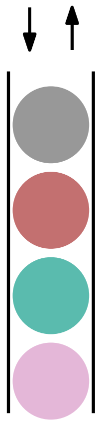
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste

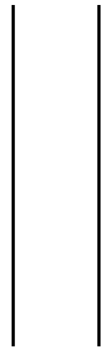


Frage

Angenommen wir können Stacks (als Blackbox) verwenden.
(Wie) können wir daraus eine Queue bauen?

Eine Queue aus Stacks

Operationen:

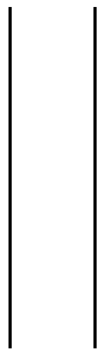


Stack A

Eine Queue aus Stacks

Operationen:

- **push**: 3, 1, 4

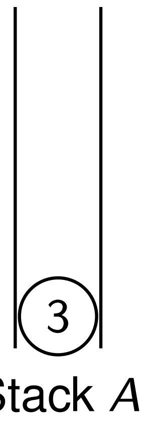


Stack A

Eine Queue aus Stacks

Operationen:

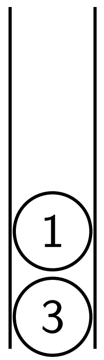
- **push**: 3, 1, 4



Eine Queue aus Stacks

Operationen:

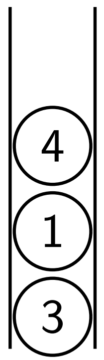
- **push**: 3, 1, 4



Eine Queue aus Stacks

Operationen:

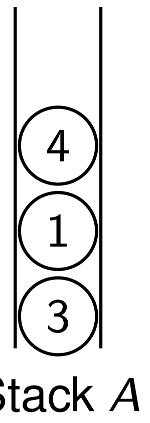
- **push**: 3, 1, 4



Eine Queue aus Stacks

Operationen:

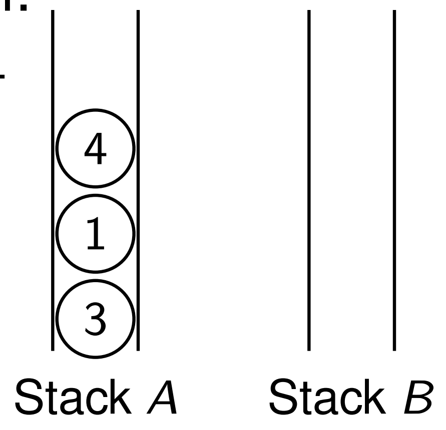
- **push**: 3, 1, 4
- **pop**()



Eine Queue aus Stacks

Operationen:

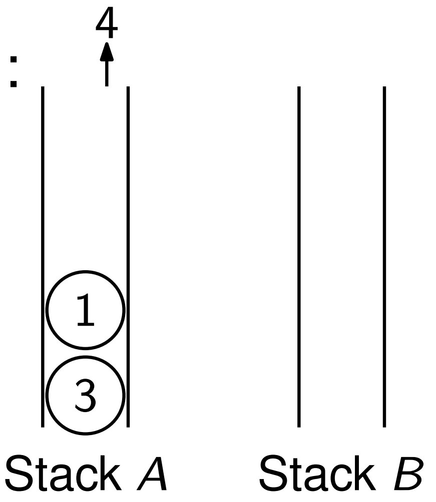
- **push**: 3, 1, 4
- **pop**()



Eine Queue aus Stacks

Operationen:

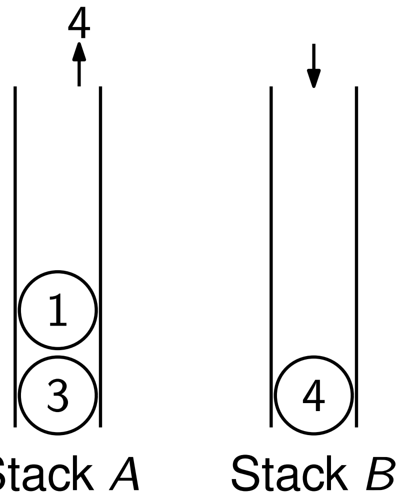
- **push**: 3, 1, 4
- **pop**()



Eine Queue aus Stacks

Operationen:

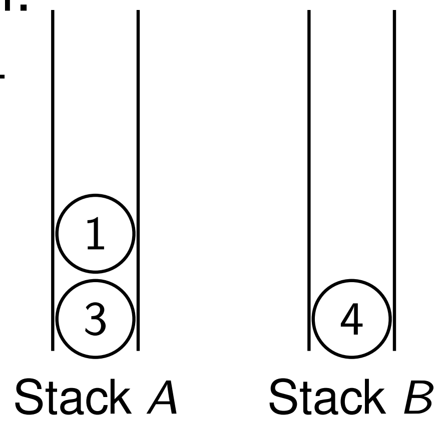
- **push**: 3, 1, 4
- **pop**()



Eine Queue aus Stacks

Operationen:

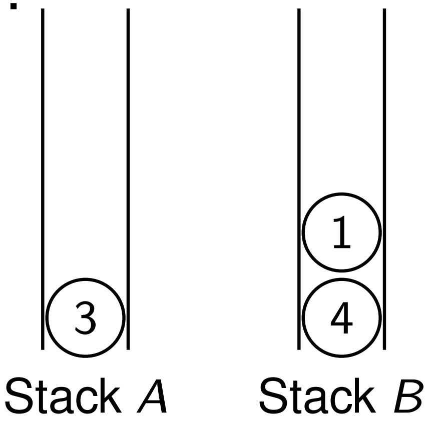
- **push**: 3, 1, 4
- **pop**()



Eine Queue aus Stacks

Operationen:

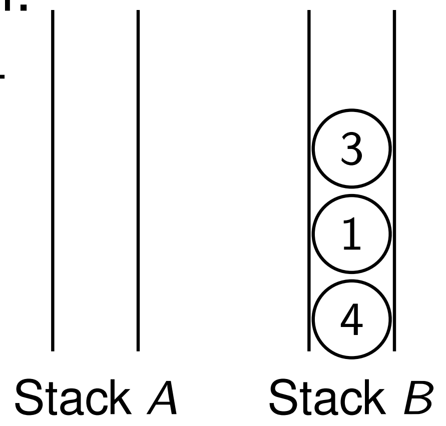
- **push**: 3, 1, 4
- **pop**()



Eine Queue aus Stacks

Operationen:

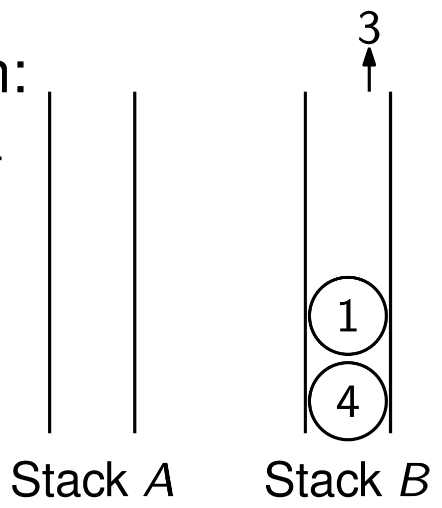
- **push**: 3, 1, 4
- **pop**()



Eine Queue aus Stacks

Operationen:

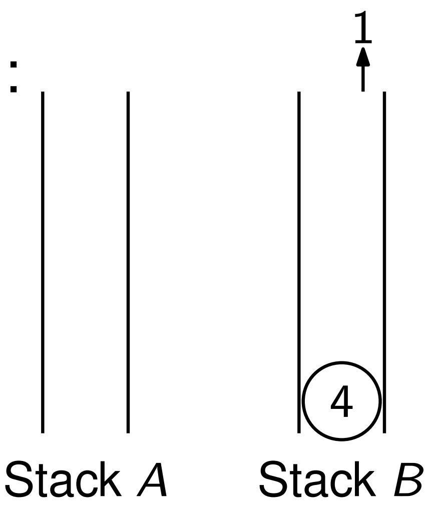
- **push**: 3, 1, 4
- **pop**()



Eine Queue aus Stacks

Operationen:

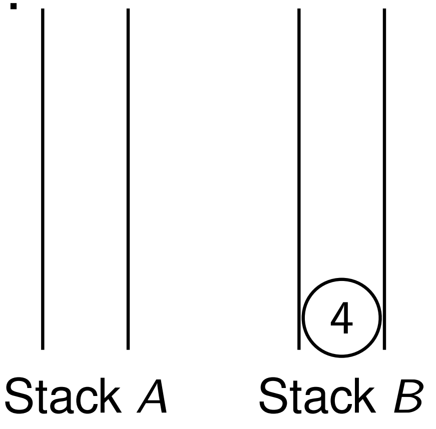
- **push**: 3, 1, 4
- **pop**()
- **pop**()



Eine Queue aus Stacks

Operationen:

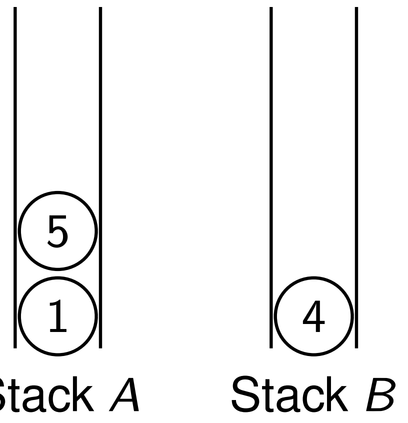
- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



Eine Queue aus Stacks

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5

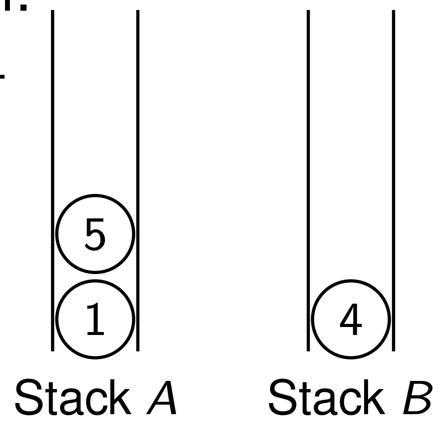


Eine Queue aus Stacks

Algorithmische Umsetzung

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



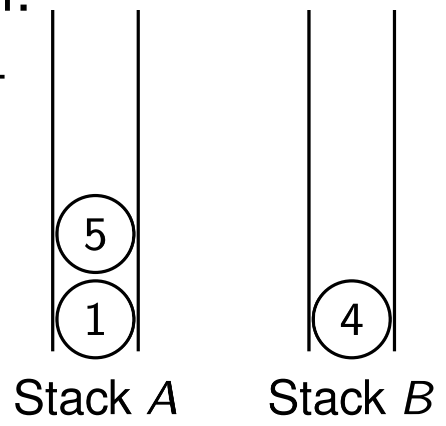
Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



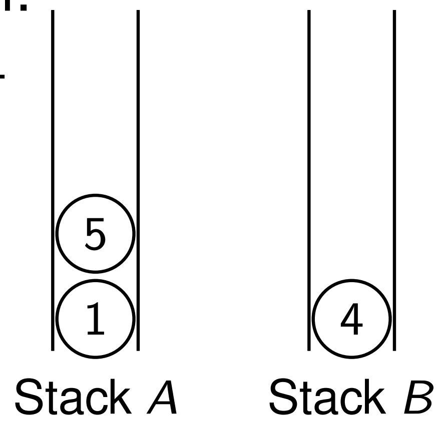
Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



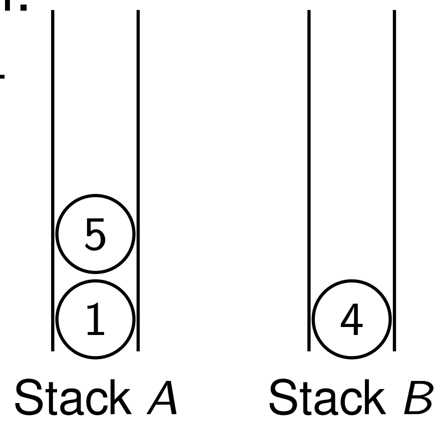
Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:
  - falls B voll: von B poppen

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



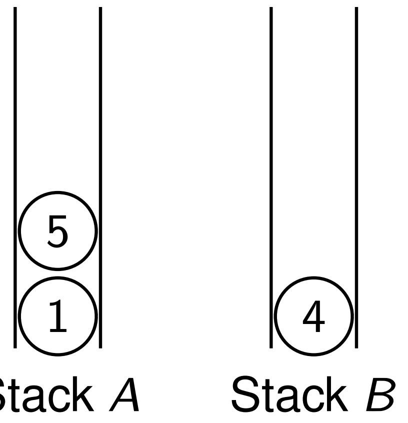
Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:
  - falls B voll: von B poppen
  - falls B leer: alles von A nach B verschieben

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



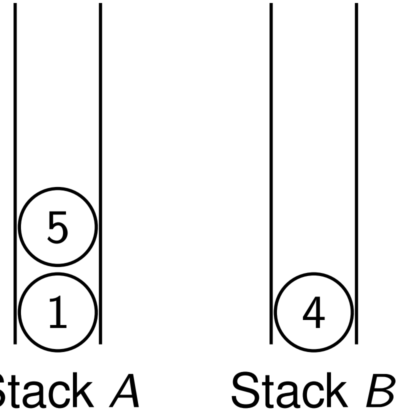
Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen $\quad O(1)$
- **pop**:
  - falls B voll: von B poppen
  - falls B leer: alles von A nach B verschieben

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



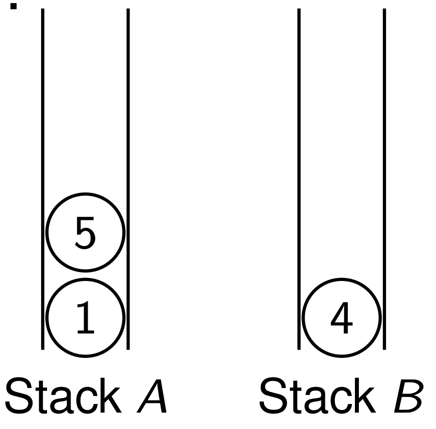
Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen $O(1)$
- **pop**:
  - falls B voll: von B poppen $O(1)$
  - falls B leer: alles von A nach B verschieben

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



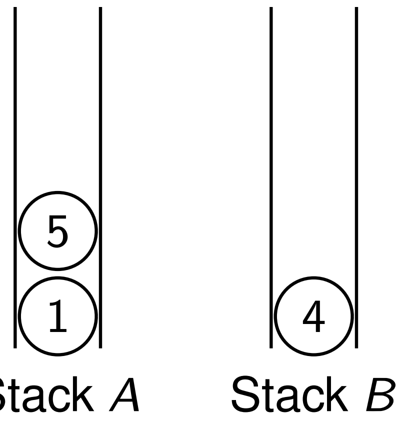
Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen $O(1)$
- **pop**:
  - falls B voll: von B poppen $O(1)$
  - falls B leer: alles von A nach B verschieben $O(|A|)$

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



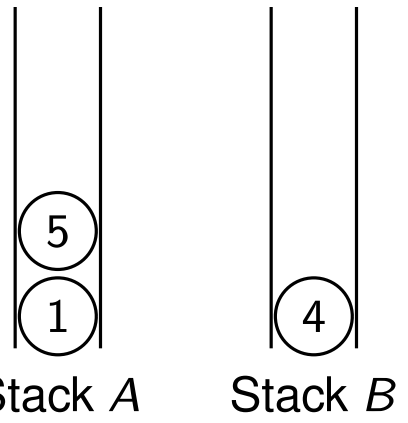
Eine Queue aus Stacks

Algorithmische Umsetzung

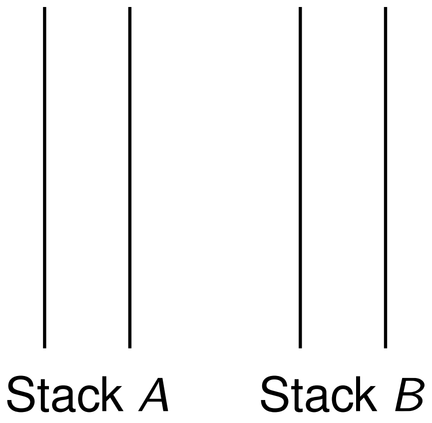
- **push**: auf A pushen $O(1)$
- **pop**: $O(|A|)$
  - falls B voll: von B poppen $O(1)$
  - falls B leer: alles von A nach B verschieben $O(|A|)$

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5

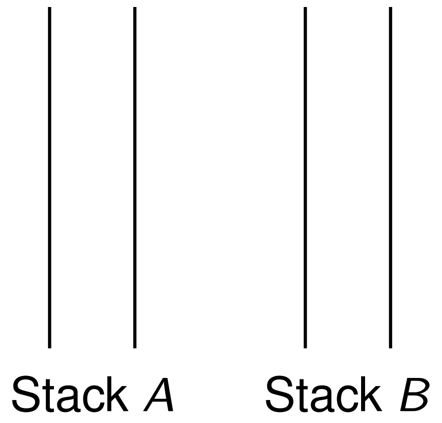


Analyse: Kontomethode



Analyse: Kontomethode

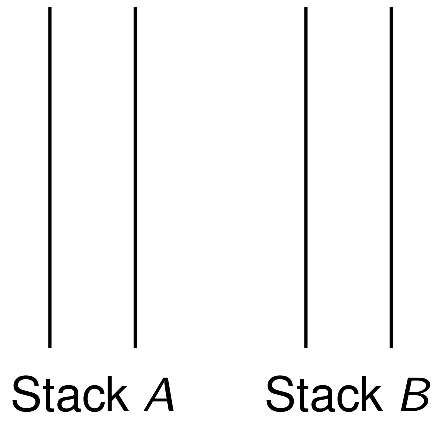
Idee



Analyse: Kontomethode

Idee

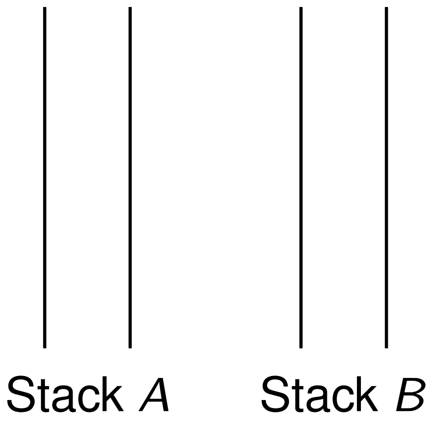
- günstige Operationen bauen Guthaben auf



Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

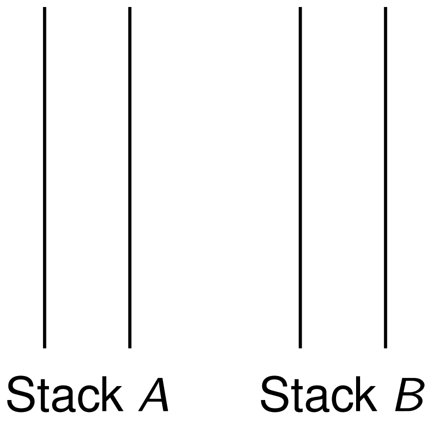


Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier



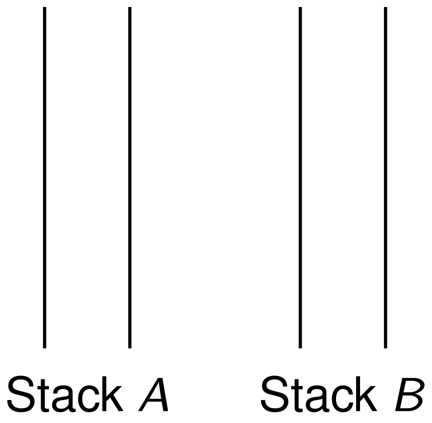
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen



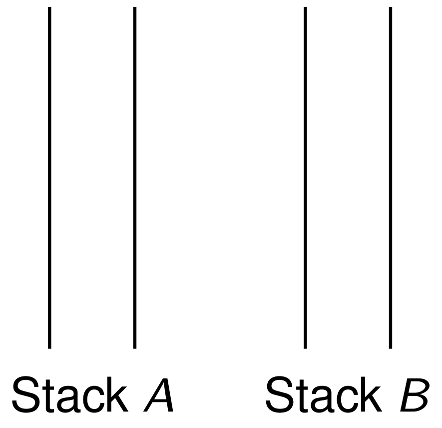
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



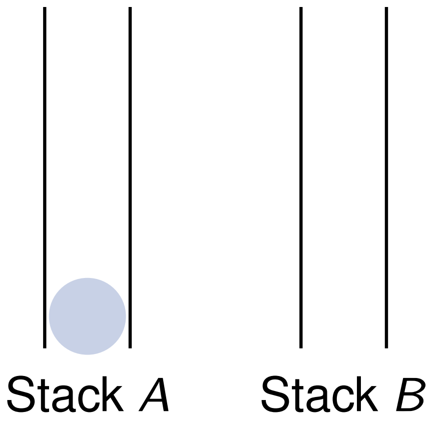
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



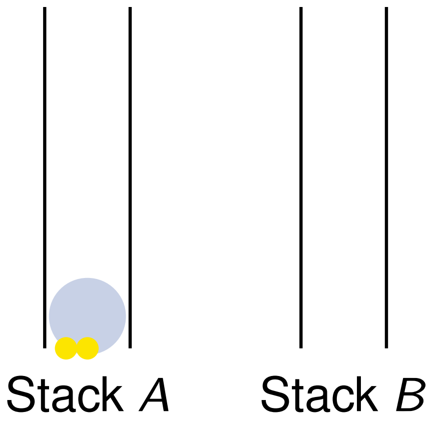
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



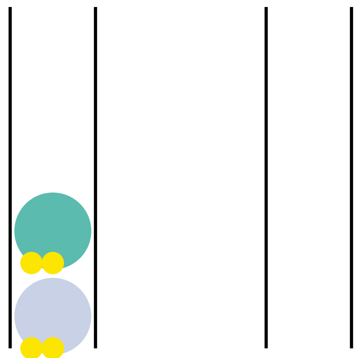
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



Stack A Stack B

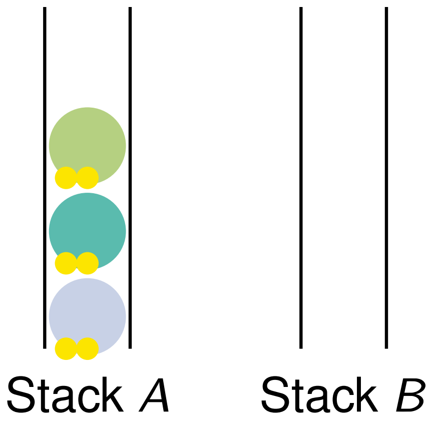
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



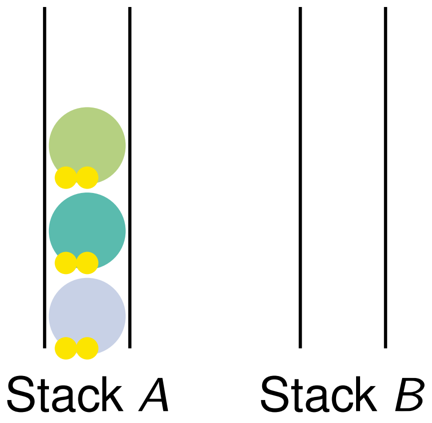
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1



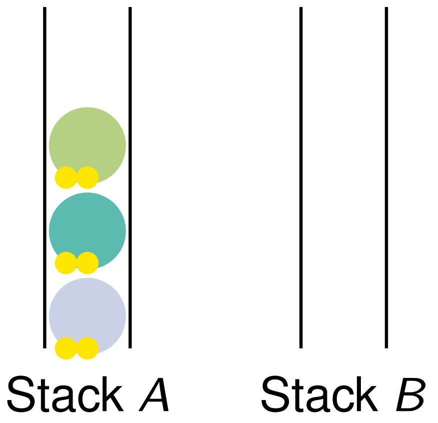
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()**



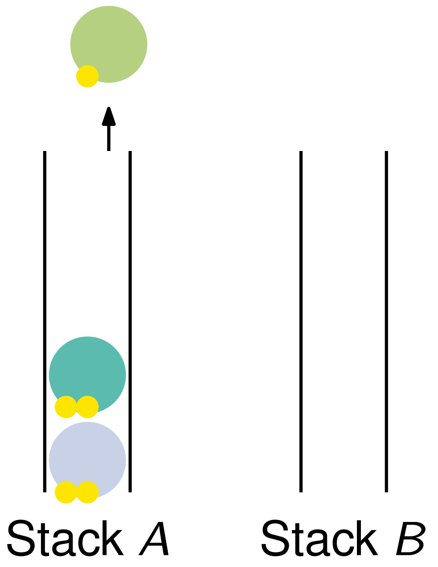
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()**



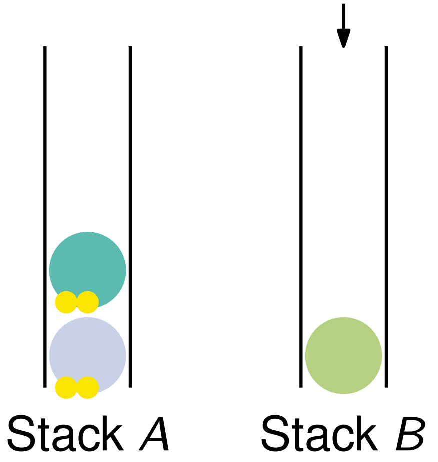
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()**



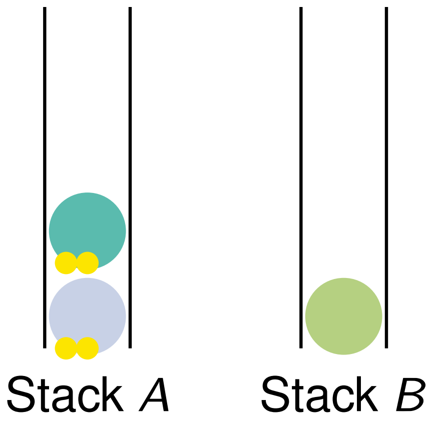
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()**



Analyse: Kontomethode

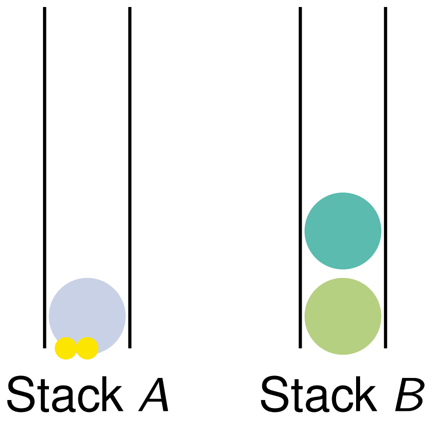
Karlsruher Institut für Technologie

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()**



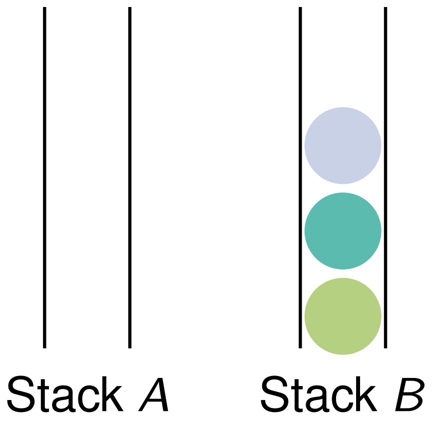
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()**



Analyse: Kontomethode

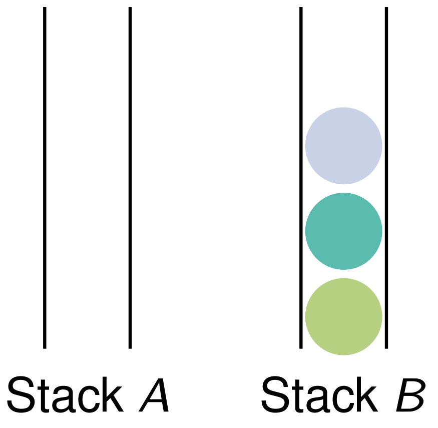
Karlsruher Institut für Technologie

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()** (für alle Elemente in A)



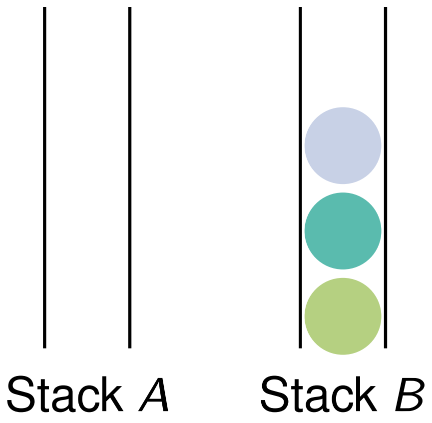
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()** (für alle Elemente in A)
  - Falls B voll: B.**pop**() mit Kosten 1



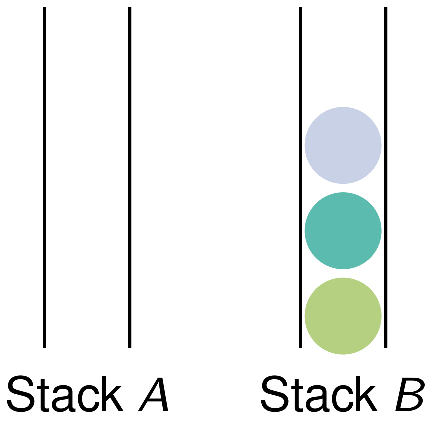
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()** (für alle Elemente in A)
  - Falls B voll: B.**pop**() mit Kosten 1
- Konto wird nie negativ



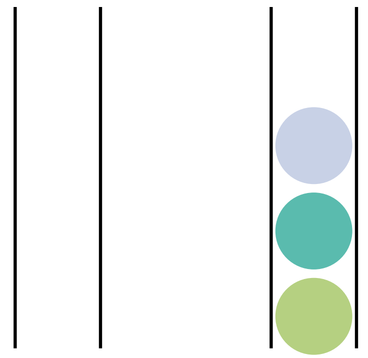
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

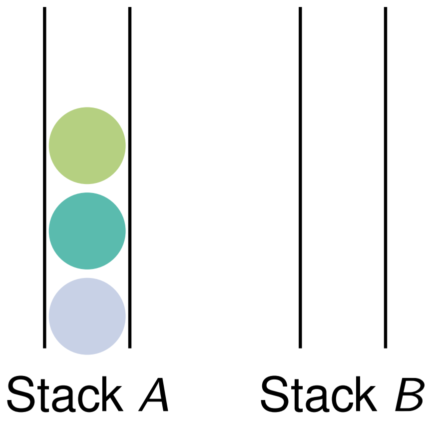
Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
  - Falls B leer: nutze 2 Guthaben für A.**pop()** und B.**push()** (für alle Elemente in A)
  - Falls B voll: B.**pop**() mit Kosten 1
- Konto wird nie negativ
- $3, 1 \in \Theta(1) \Rightarrow$ konstante amortisierte Kosten



Analyse: Potentialmethode

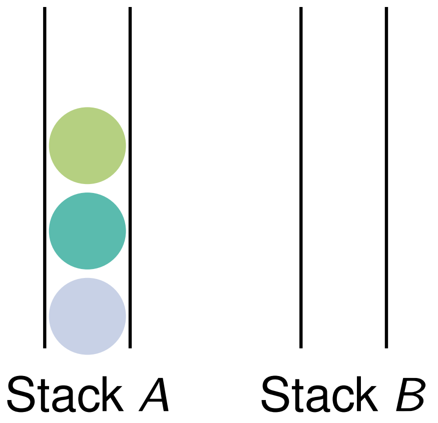
Potentialfunktion



Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

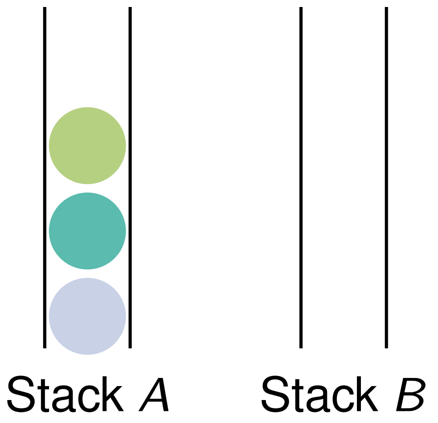


Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse



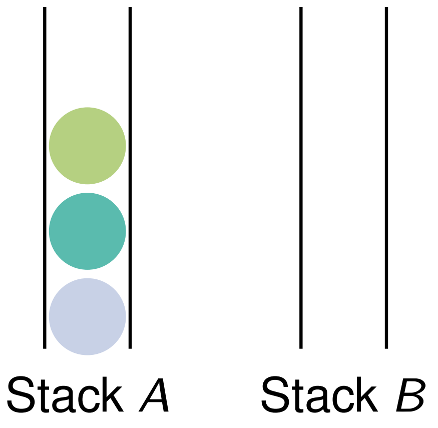
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$



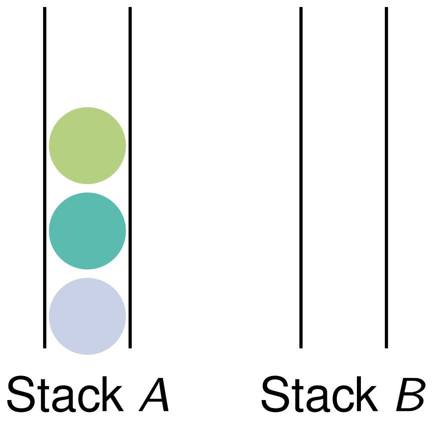
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)



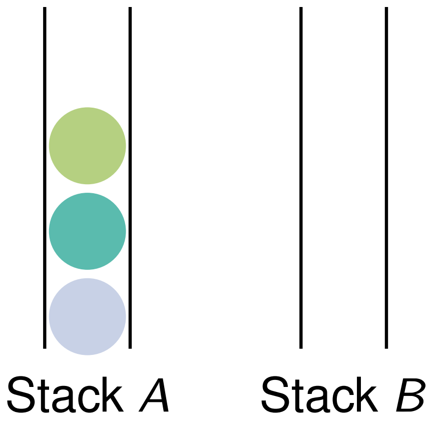
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:



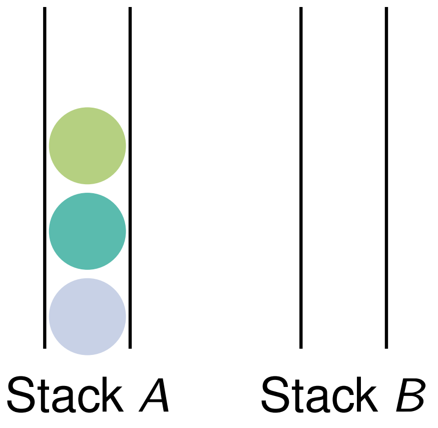
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

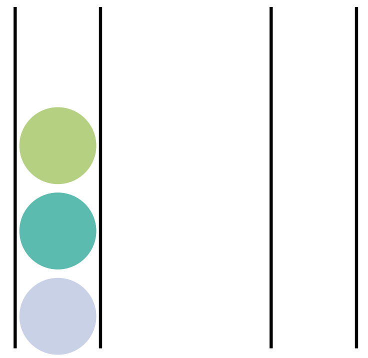
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

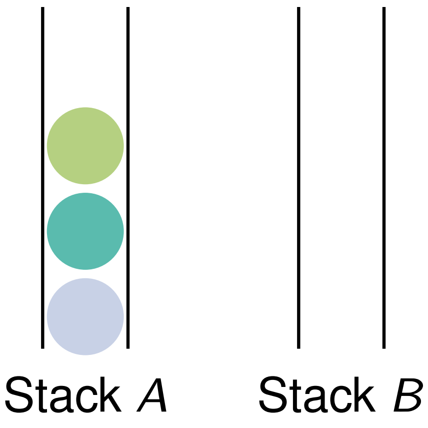
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

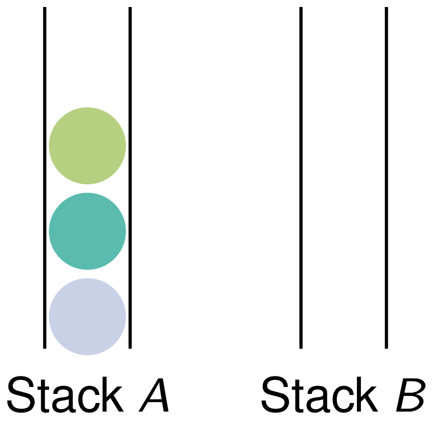
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

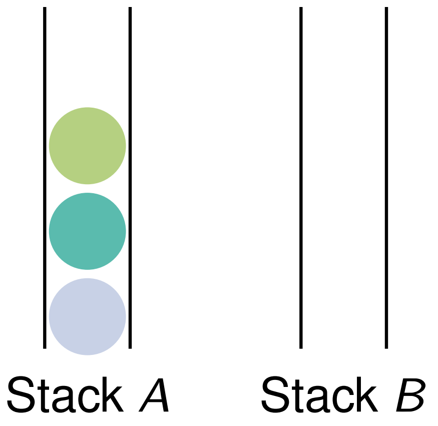


Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)



Analyse

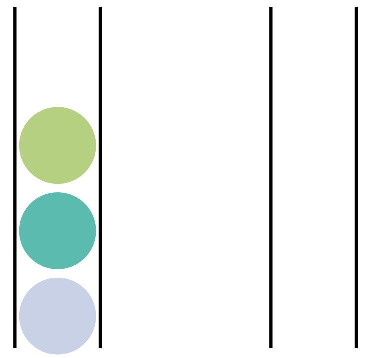
- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
  - $c_i = 2 \cdot |A_{i-1}| + 1$

Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)



Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
  - $c_i = 2 \cdot |A_{i-1}| + 1$
  - $\Phi(D_i) - \Phi(D_{i-1}) = -2|A_{i-1}|$

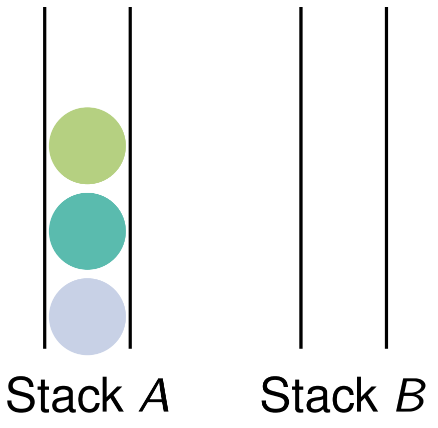
Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

Karlsruher Institut für Technologie

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)



Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1$
  - $c_i = 2 \cdot |A_{i-1}| + 1$
  - $\Phi(D_i) - \Phi(D_{i-1}) = -2|A_{i-1}|$

Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

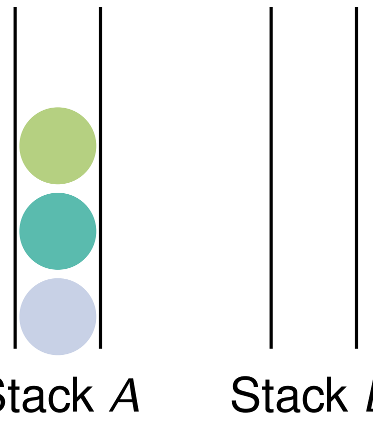
Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
  - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1$
  - $c_i = 2 \cdot |A_{i-1}| + 1$
  - $\Phi(D_i) - \Phi(D_{i-1}) = -2|A_{i-1}|$

amortisierte Kosten in $O(1)$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$