

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

- 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

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( $x, y-1$ )
```

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:

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

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

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)
  
```

```

algTwo(x, y) // Annahme: initial x ≤ y
| if x ≥ y then
| | return x
| total := algTwo(x, ⌊ $\frac{y+3x}{4}$ ⌋)
| total += algTwo(⌊ $\frac{y+3x}{4}$ ⌋, ⌊ $\frac{y+x}{2}$ ⌋)
| total += algTwo(⌊ $\frac{y+x}{2}$ ⌋, ⌊ $\frac{3y+x}{4}$ ⌋)
| return total
  
```

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)
  
```

Gesucht:

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

```

algTwo(x, y) // Annahme: initial x ≤ y
| if x ≥ y then
| | return x
| total := algTwo(x, ⌊ $\frac{y+3x}{4}$ ⌋)
| total += algTwo(⌊ $\frac{y+3x}{4}$ ⌋, ⌊ $\frac{y+x}{2}$ ⌋)
| total += algTwo(⌊ $\frac{y+x}{2}$ ⌋, ⌊ $\frac{3y+x}{4}$ ⌋)
| return total
  
```

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

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)
|

```

Gesucht:

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

```

algTwo(x, y) // Annahme: initial x ≤ y
|
|   if x ≥ y then
|   |   return x
|   |
|   |   total := algTwo(x, ⌊ $\frac{y+3x}{4}$ ⌋)
|   |   total += algTwo(⌊ $\frac{y+3x}{4}$ ⌋, ⌊ $\frac{y+x}{2}$ ⌋)
|   |   total += algTwo(⌊ $\frac{y+x}{2}$ ⌋, ⌊ $\frac{3y+x}{4}$ ⌋)
|   |   return total
|

```

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

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)
  
```

Gesucht:

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

```

algTwo(x, y) // Annahme: initial x ≤ y
| if x ≥ y then
| | return x
| total := algTwo(x, ⌊ $\frac{y+3x}{4}$ ⌋)
| total += algTwo(⌊ $\frac{y+3x}{4}$ ⌋, ⌊ $\frac{y+x}{2}$ ⌋)
| total += algTwo(⌊ $\frac{y+x}{2}$ ⌋, ⌊ $\frac{3y+x}{4}$ ⌋)
| 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

Gegeben:

```

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

Gesucht:

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

```

algTwo(x, y) // Annahme: initial x ≤ y
| if x ≥ y then
| | return x
| total := algTwo(x, ⌊ $\frac{y+3x}{4}$ ⌋)
| total += algTwo(⌊ $\frac{y+3x}{4}$ ⌋, ⌊ $\frac{y+x}{2}$ ⌋)
| total += algTwo(⌊ $\frac{y+x}{2}$ ⌋, ⌊ $\frac{3y+x}{4}$ ⌋)
| 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:  $\mathbb{N}$ , 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:

(4, 8)

```
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:

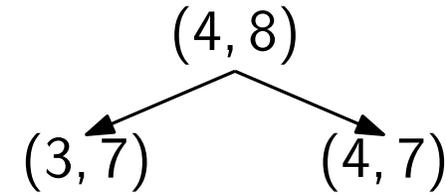
- „Problemgröße“ n
- Rekurrenz in Abh. von n
- Laufzeit in Abh. von n

Und bei **algOne**?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

```
algOne(x: ℕ, y: 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

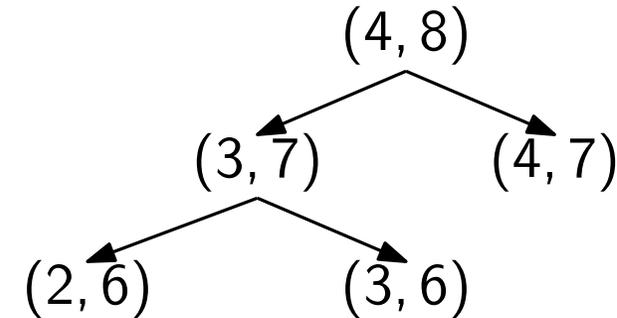
Und bei **algOne**?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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

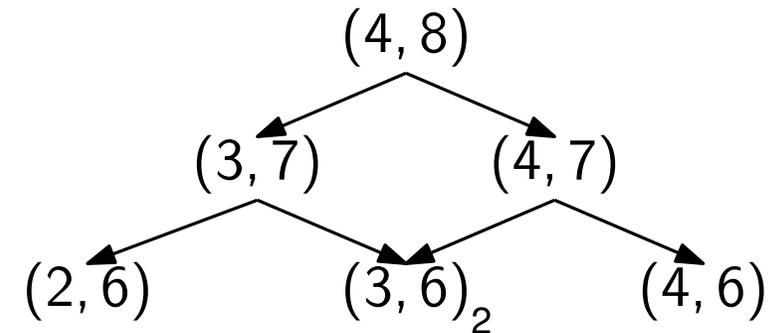
Und bei `algOne`?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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

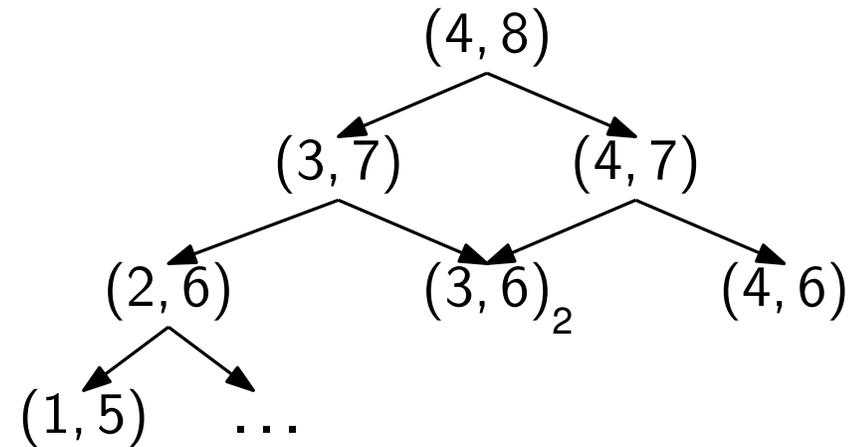
Und bei `algOne`?

Übung 2: Aufgabe 1: ALGONE

Gegeben:

```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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:

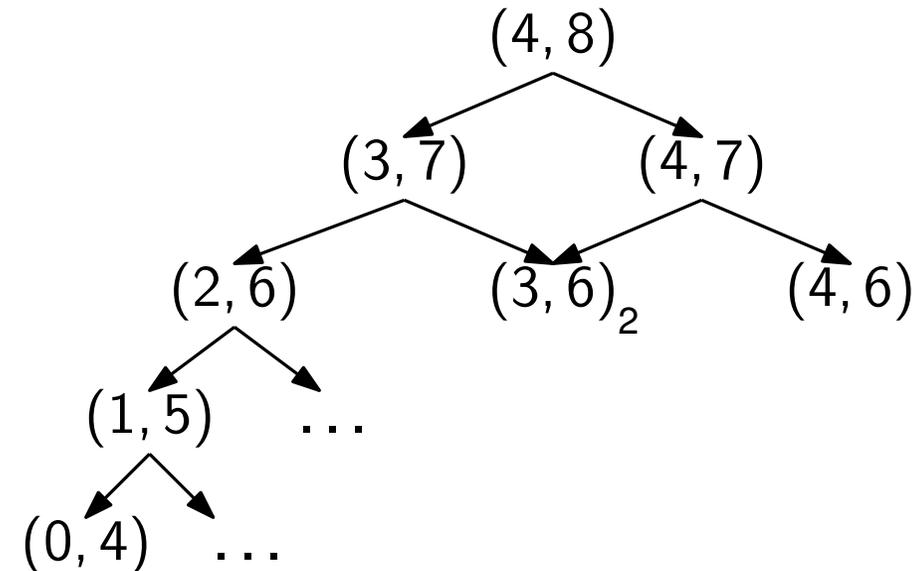
```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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:

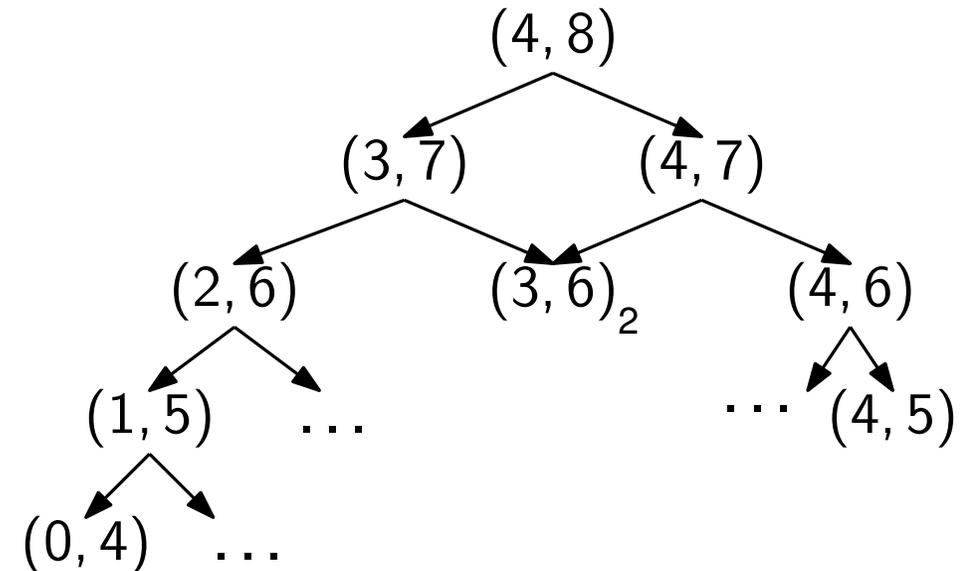
```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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:

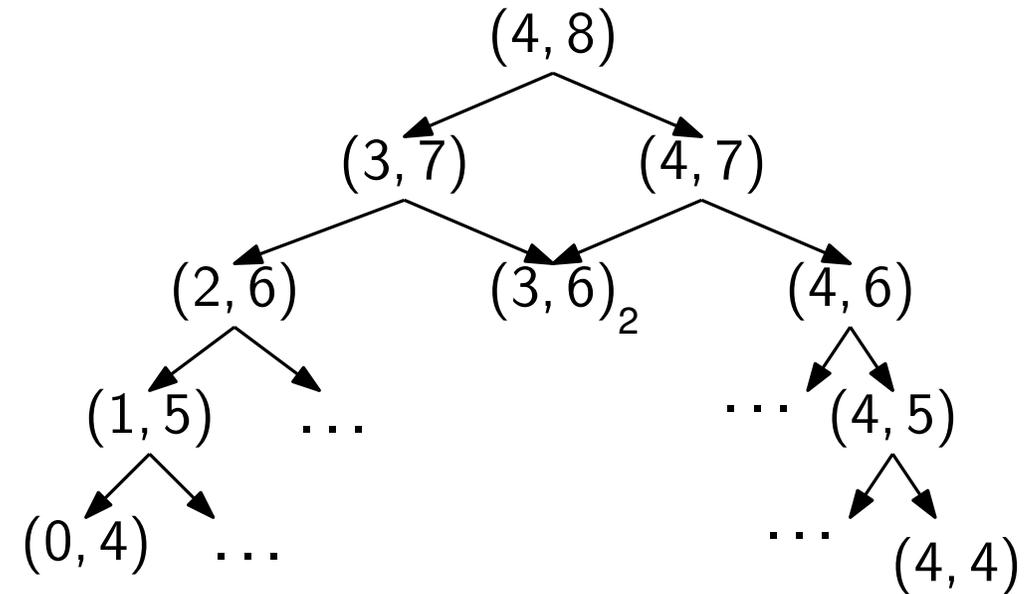
```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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:

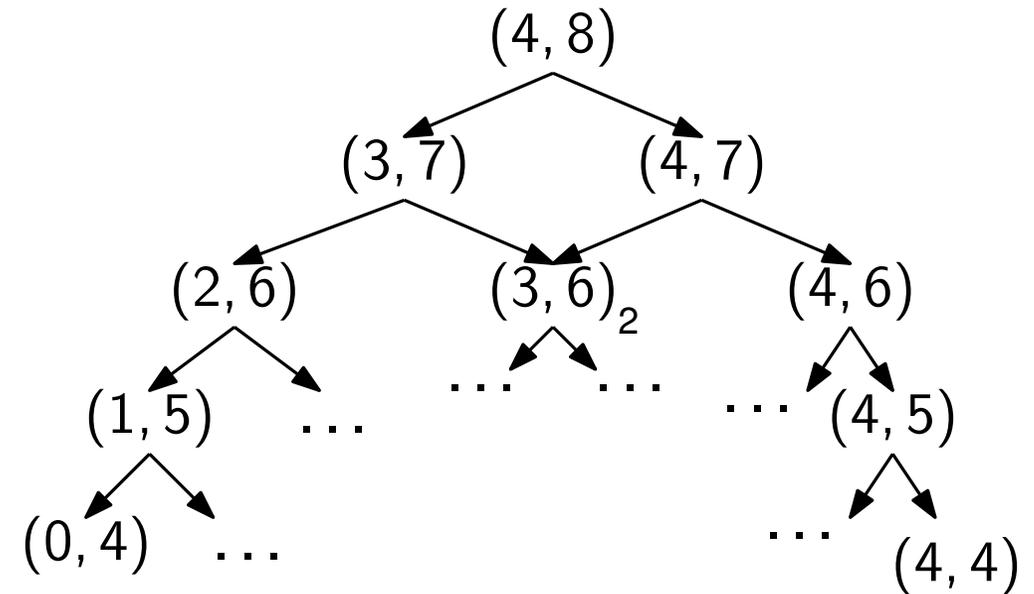
```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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:

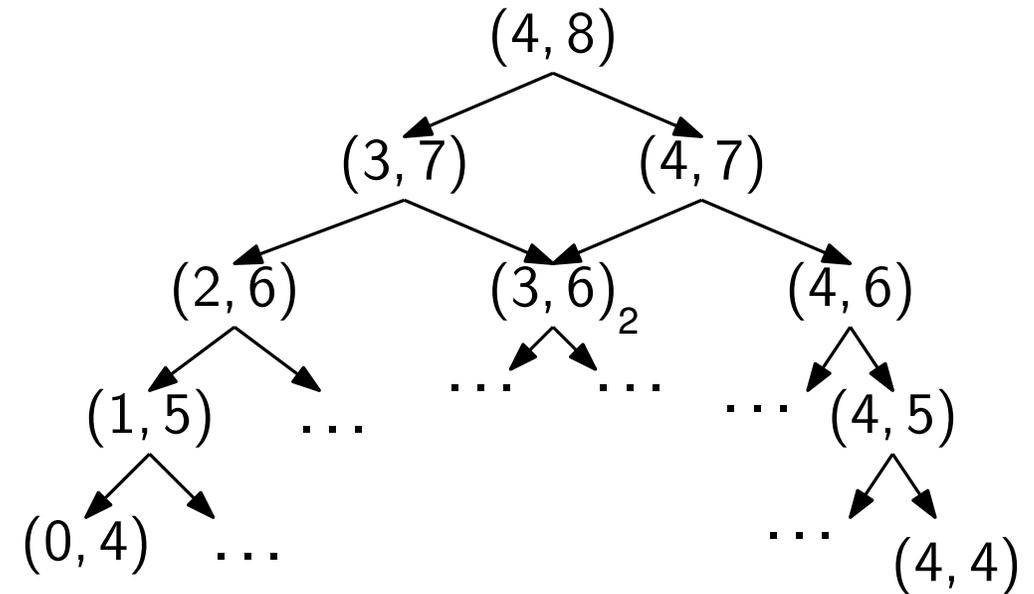
```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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 :(

Übung 2: Aufgabe 1: ALGONE

Gegeben:

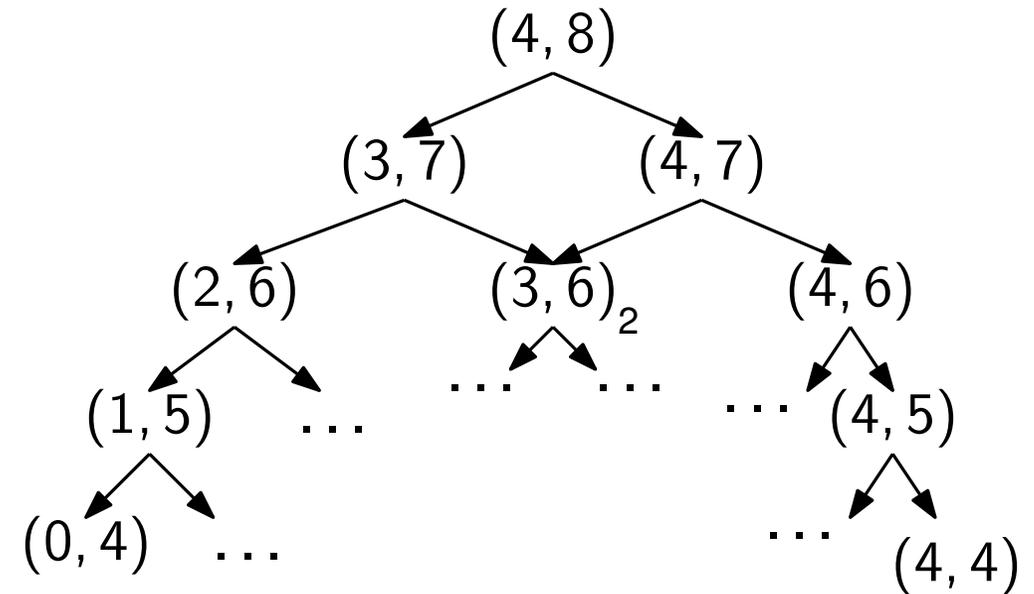
```

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:

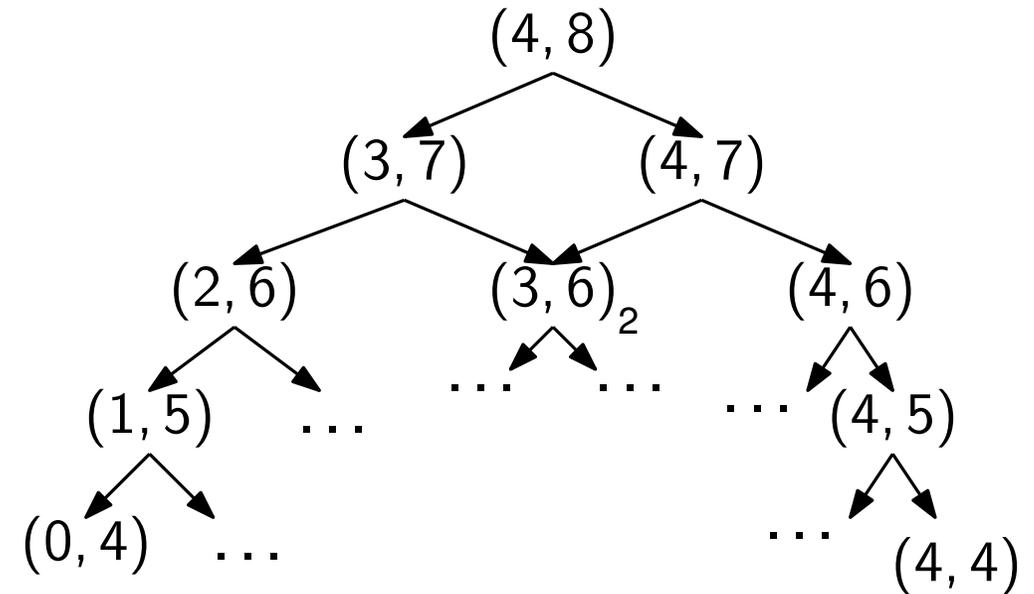
```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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

- (grobe) obere Schranke in y

Übung 2: Aufgabe 1: ALGONE

Gegeben:

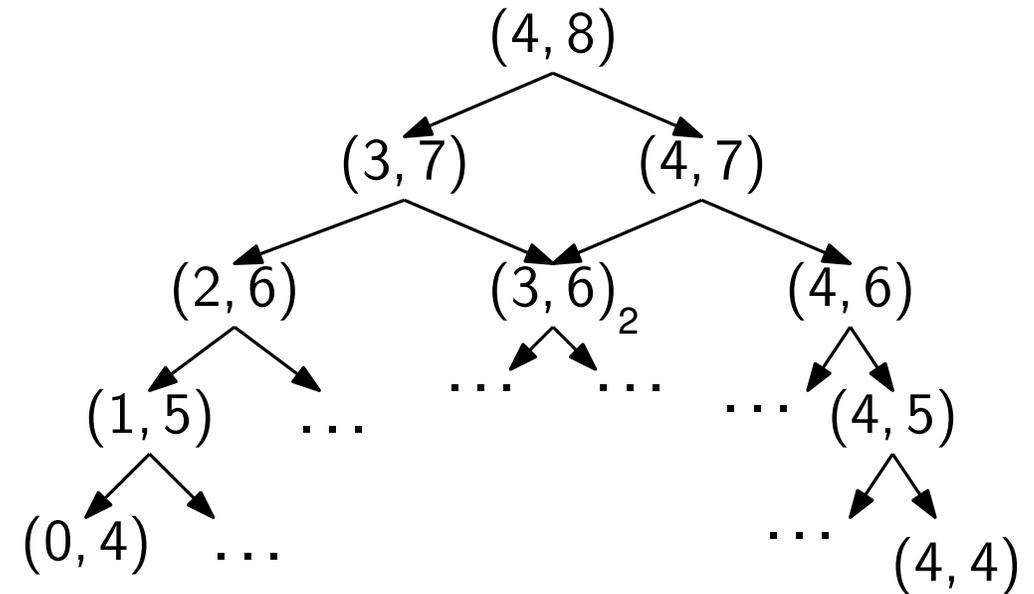
```

algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  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 :(

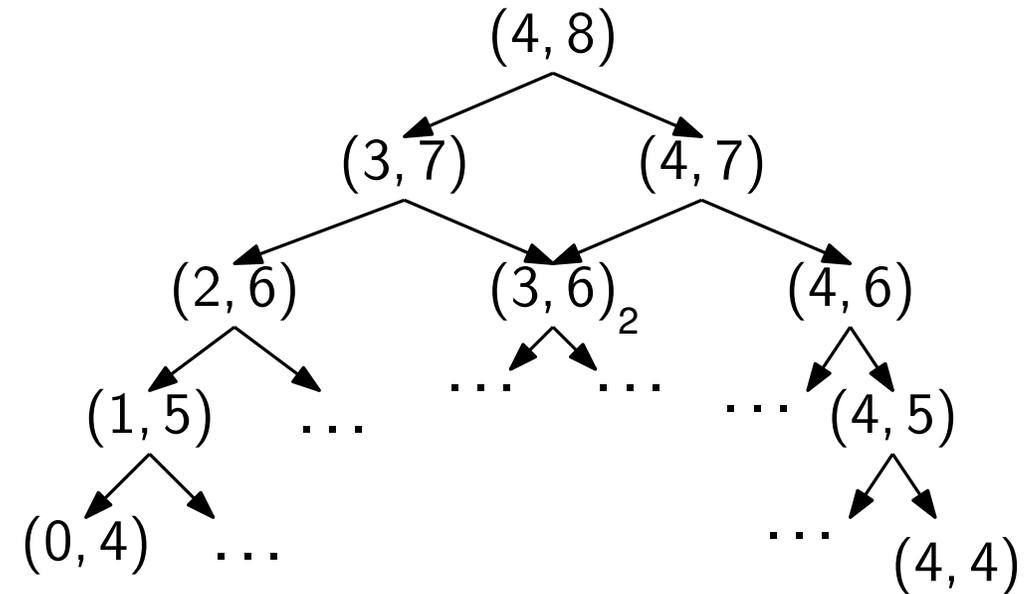
Plan

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

Ungenauere obere Schranke ALGONE

```

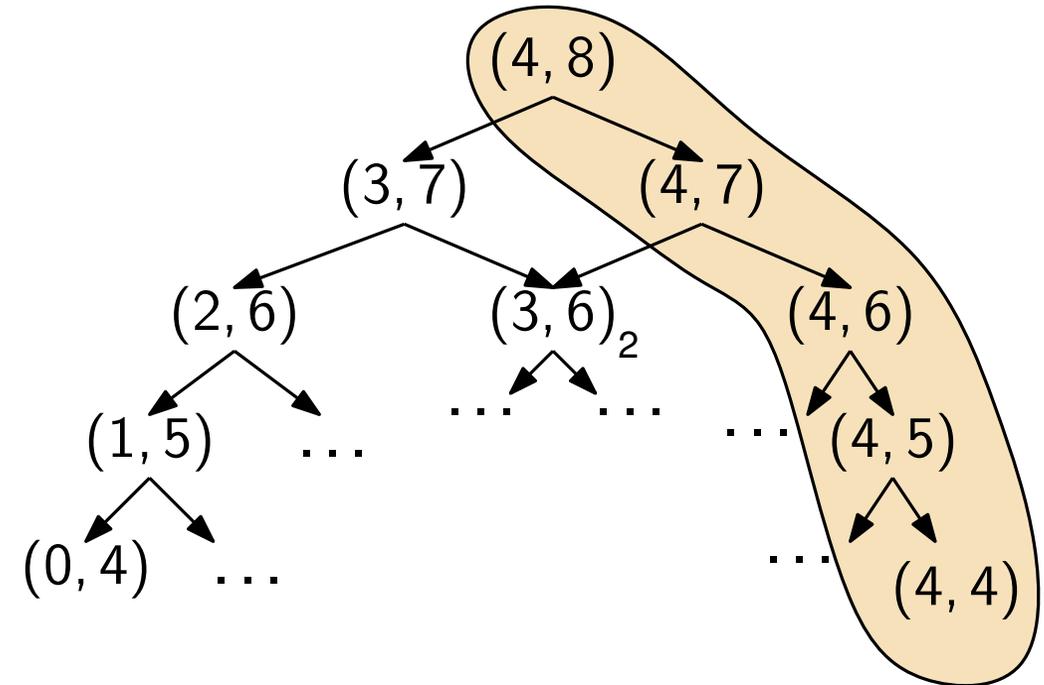
algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  then
    return 1
  return algOne( $x-1, y-1$ ) + algOne( $x, y-1$ )
  
```



Ungenauere obere Schranke ALGONE

```

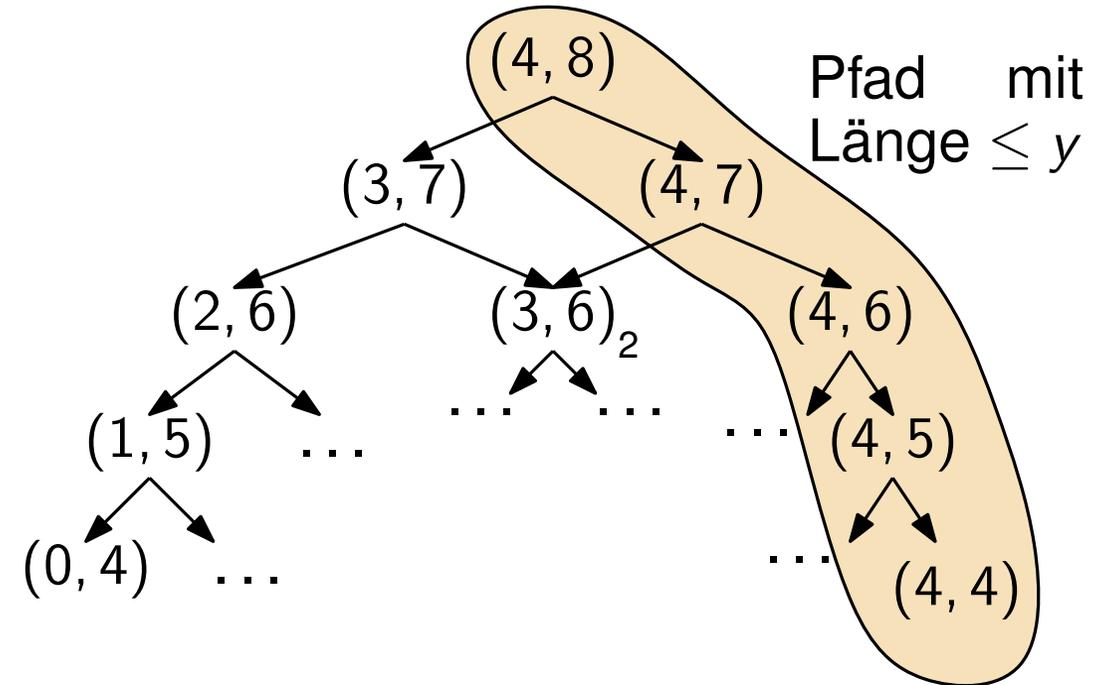
algOne( $x: \mathbb{N}, y: \mathbb{N}$ ) // Annahme: initial  $x \leq y$ 
  if  $x = 0 \vee x = y$  then
    return 1
  return algOne( $x-1, y-1$ ) + algOne( $x, y-1$ )
  
```



Ungenauere 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( $x, y-1$ )
  
```

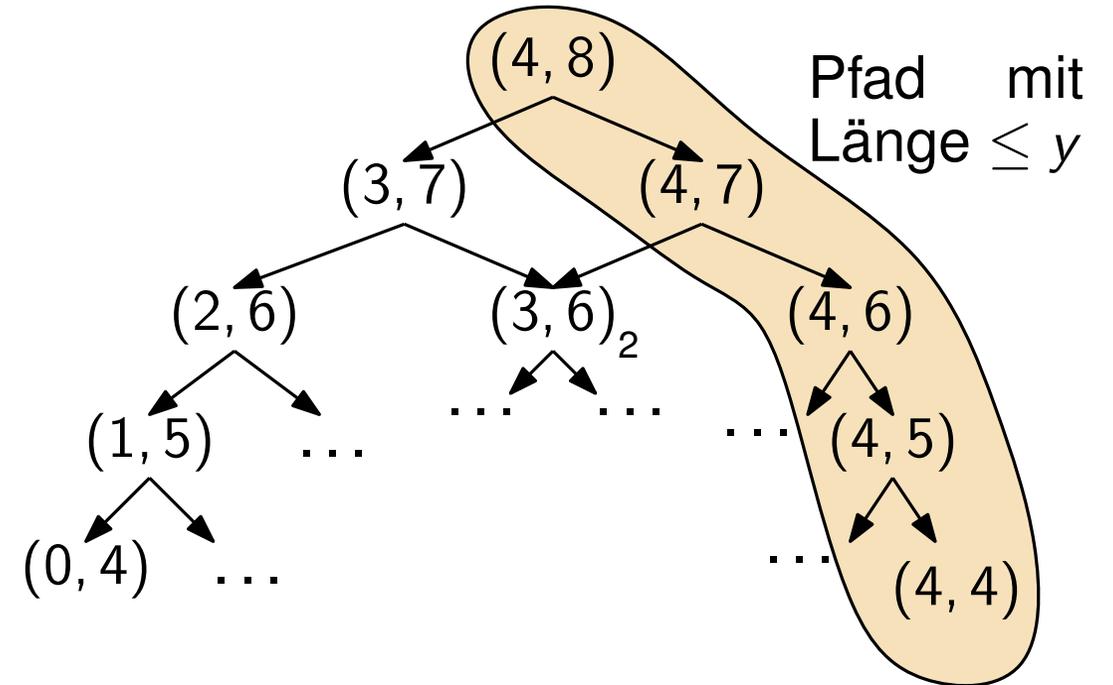


Ungenauere obere Schranke ALGONE

```

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



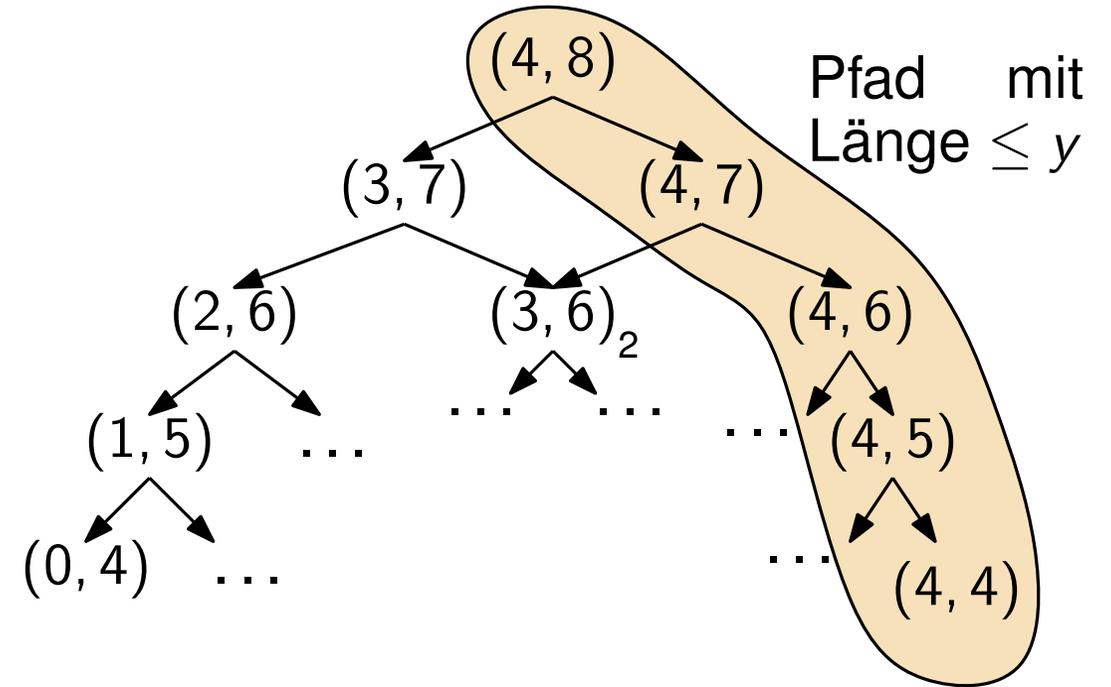
Ungenauere obere Schranke ALGONE

```

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



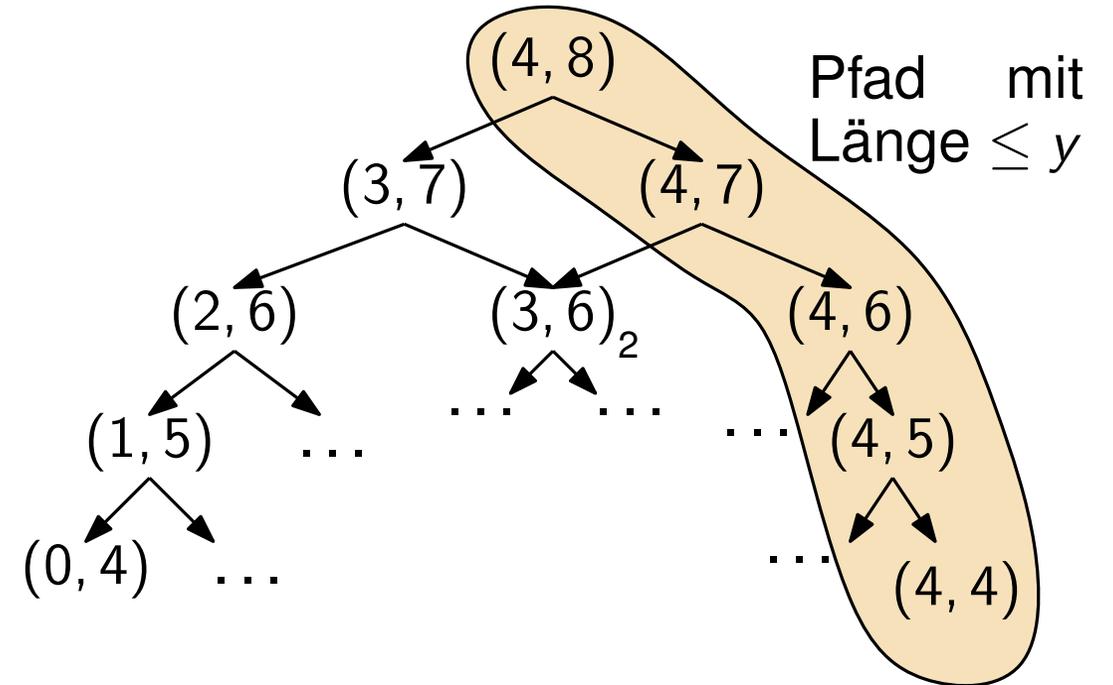
Ungenauere obere Schranke ALGONE

```

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



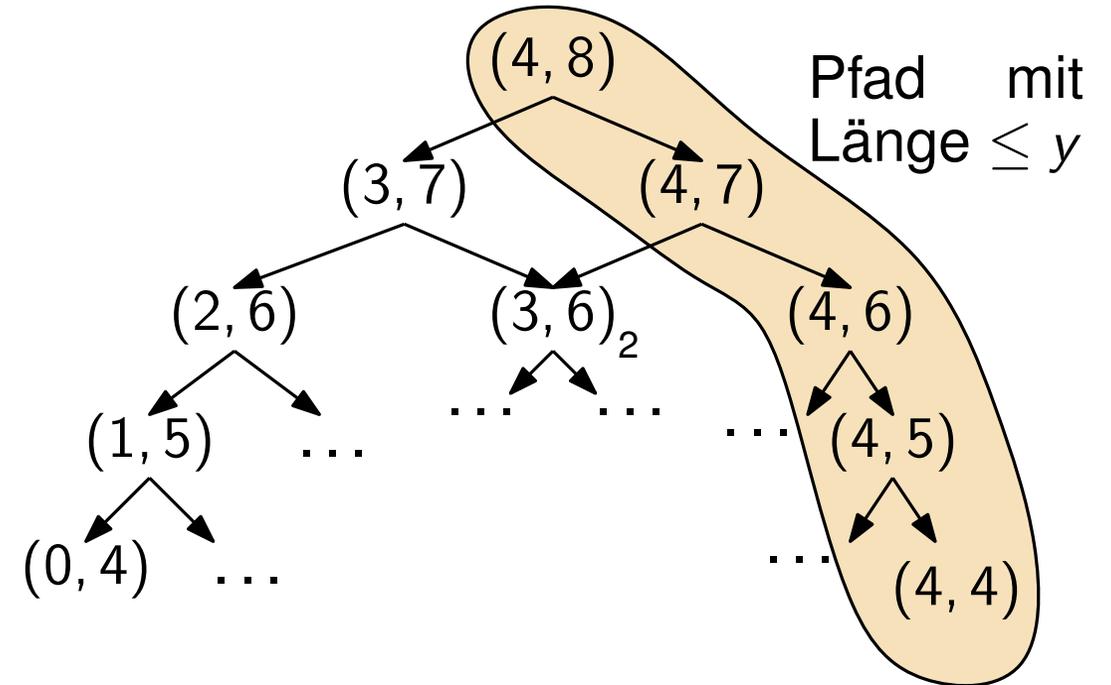
Ungenaue obere Schranke ALGONE

```

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



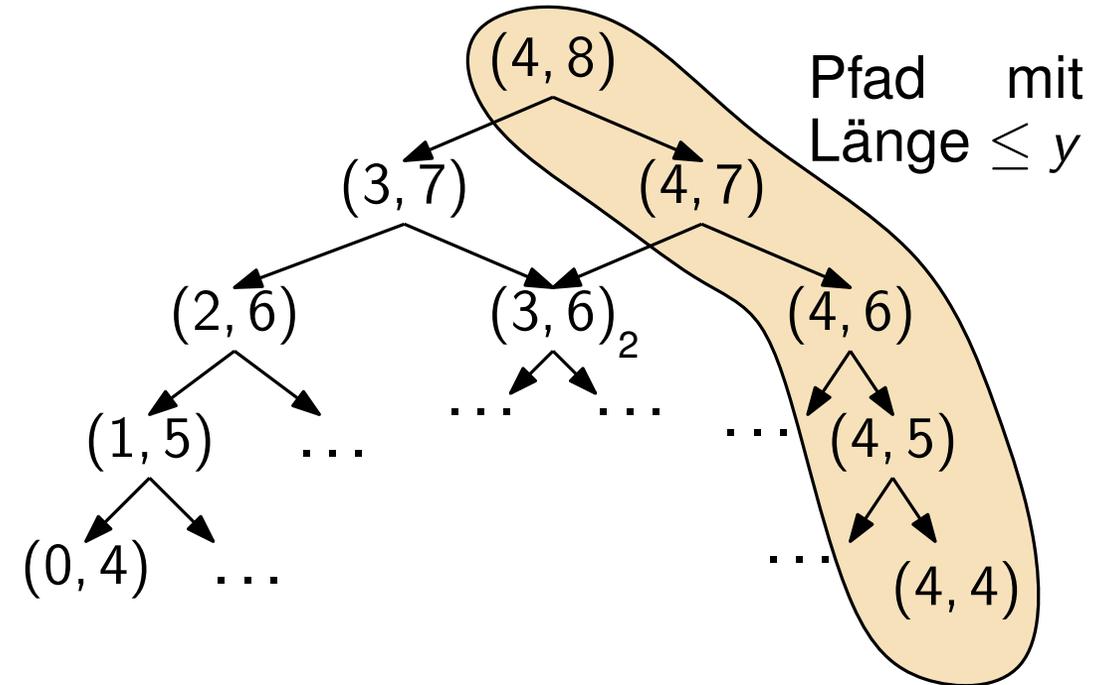
Ungenauere obere Schranke ALGONE

```

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 $\leq 2 \cdot 2^y$



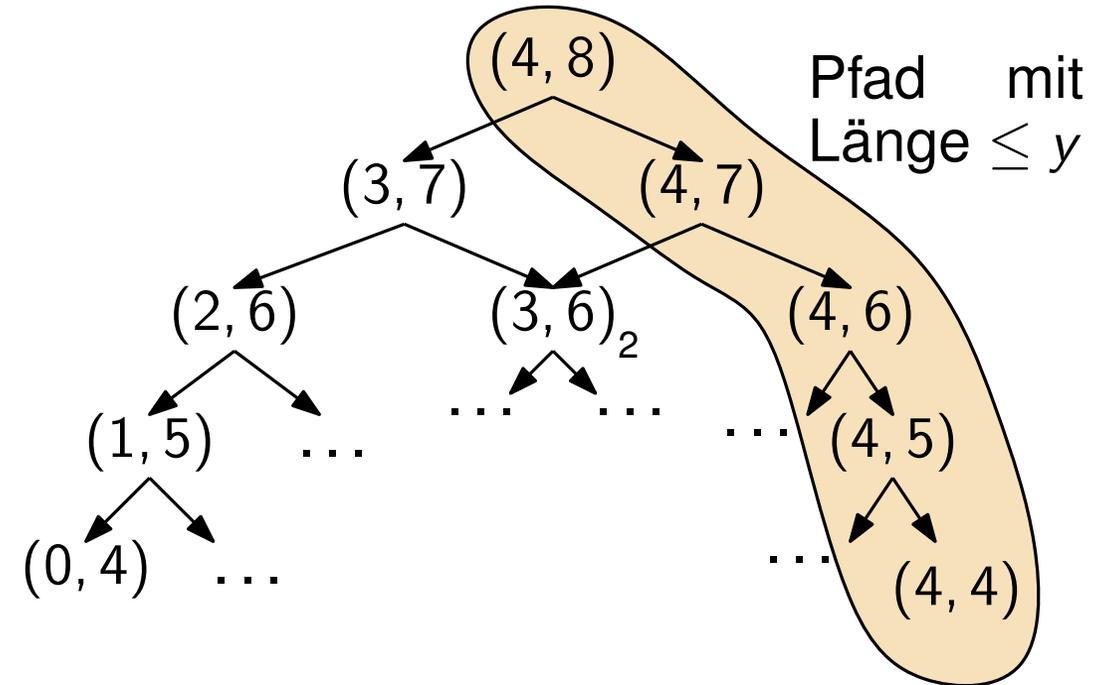
Ungenauere obere Schranke ALGONE

```

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 $O(2^n)$



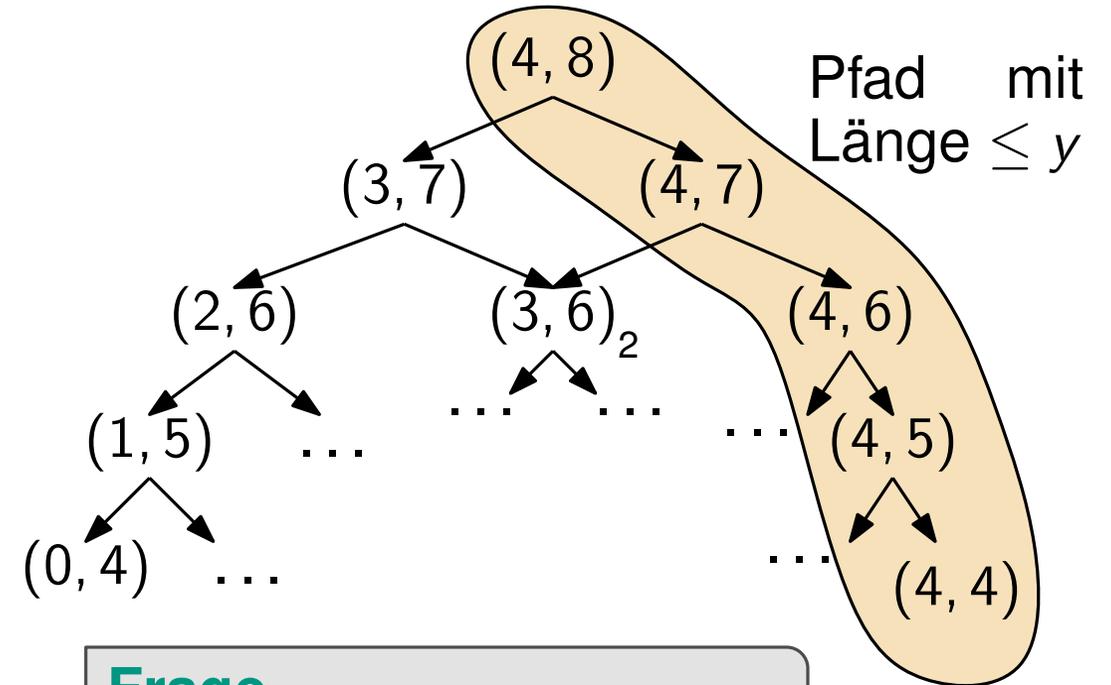
Ungenauere obere Schranke ALGONE

```

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?

Ungenauere obere Schranke ALGONE

```
algOne(x: ℕ, y: 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)
```

(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 $O(2^n)$

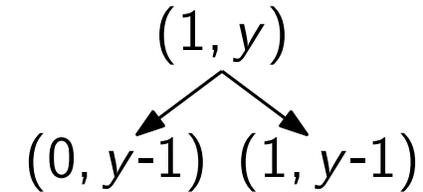
Frage

Ist diese Analyse gut?

Ungenauere 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 $O(2^n)$

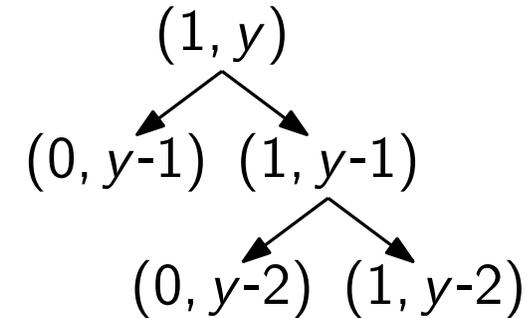
Frage

Ist diese Analyse gut?

Ungenauere 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?

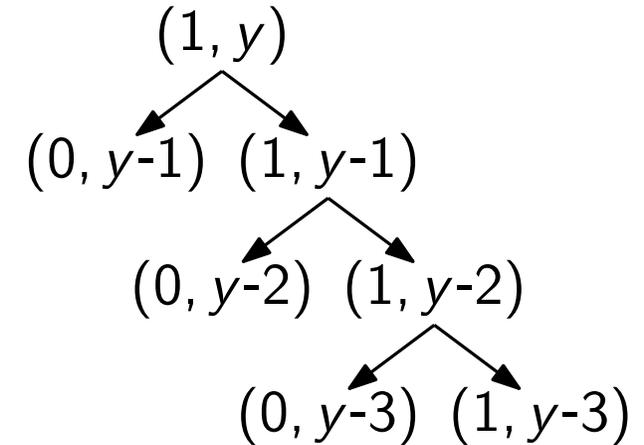
Ungenauere obere Schranke ALGONE

```

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?

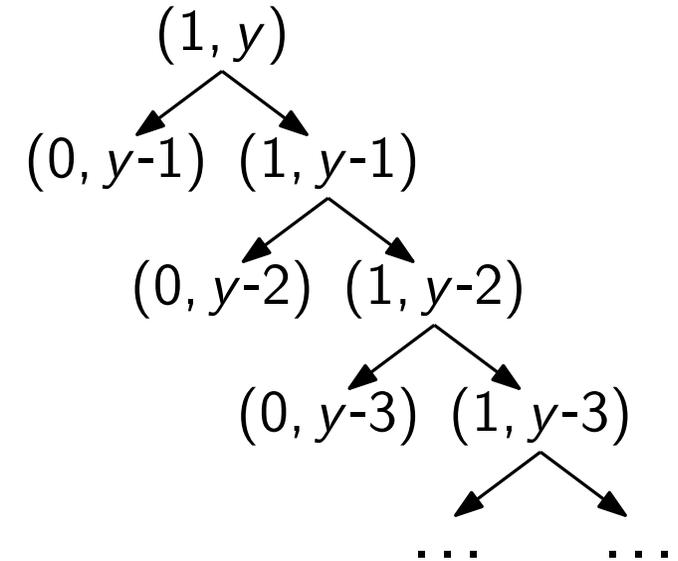
Ungenauere obere Schranke ALGONE

```

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?

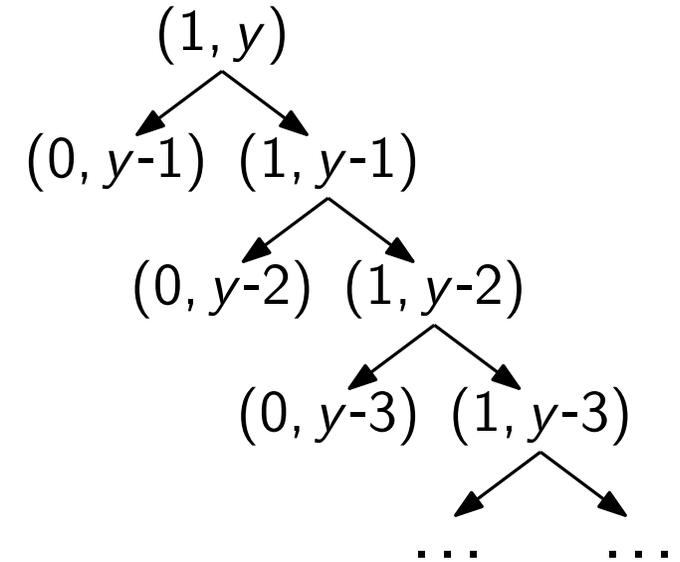
Ungenauere obere Schranke ALGONE

```

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

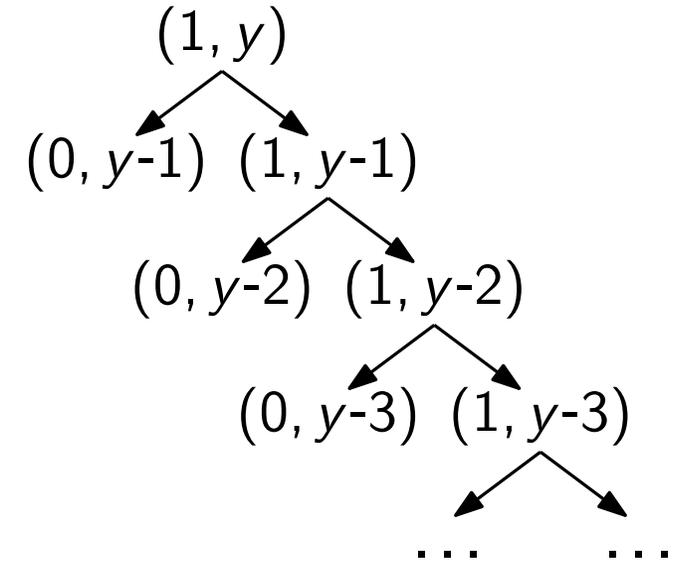
Ungenauere obere Schranke ALGONE

```

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 , große x

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)
```

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...

Genauere Analyse ALGONE

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

So
■

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

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  $x \leq y$   
  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

Genauere Analyse ALGONE

`algOne(x: ℕ, y: ℕ) // Annahme: initial $x \leq y$`

So

■

■

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

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									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](#).

0 1 3 6 2 7
: 13
: 20
23 15
10 22 11 21

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

Sort: [relevance](#) | [references](#) | [number](#) | [modified](#) | [created](#) Format: [long](#) | [short](#) | [data](#)

[A000217](#) Triangular numbers: $a(n) = \text{binomial}(n+1,2) = n*(n+1)/2 = 0 + 1 + 2 + \dots + n$.
 (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 $\text{binomial}(n+1, 2)$ (preferred).
 Also generalized hexagonal numbers: $n*(2*n-1)$, $n=0, +1, +2, +3, \dots$ Generalized k-gonal numbers are second k-gonal numbers and positive terms of k-gonal numbers interleaved, $k \geq 5$. 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 $n \geq 1$, $a(n) = n*(n+1)/2$ 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								5	220	286	364	455
4												0	495	715	1001	1365
5												2	792	1287	2002	3003
6												2	924	1716	3003	5005
7												0	792	1716	3432	6435
8												5	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

The OEIS is supported by [the many generous donors to the OEIS Foundation](#).

0 1 3 6 2 7
: 13
: 20
23 12
10 22 11 21

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES[®]

founded in 1964 by N. J. A. Sloane

1, 5, 15, 35, 70, 126, 210, 330, 495, 715, 1001, 1365 [Hints](#)

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

Search: **seq:1,5,15,35,70,126,210,330,495,715,1001,1365**

Displaying 1-1 of 1 result found. page 1

Sort: [relevance](#) | [references](#) | [number](#) | [modified](#) | [created](#) Format: [long](#) | [short](#) | [data](#)

A000332 Binomial coefficient binomial(n,4) = n*(n-1)*(n-2)*(n-3)/24. +30
360
(Formerly M3853 N1578)

0, 0, 0, 0, 1, 5, 15, 35, 70, 126, 210, 330, 495, 715, 1001, 1365, 1820, 2380, 3060, 3876, 4845, 5985, 7315, 8855, 10626, 12650, 14950, 17550, 20475, 23751, 27405, 31465, 35960, 40920, 46376, 52360, 58905, 66045, 73815, 82251, 91390, 101270, 111930, 123410 ([list](#); [graph](#); [refs](#); [listen](#); [history](#); [text](#); [internal format](#))

OFFSET 0,6

COMMENTS Number of intersection points of diagonals of convex n-gon where no more than two diagonals intersect at any point in the interior.
Also the number of equilateral triangles with vertices in an equilateral triangular array of points with n rows (offset 1), with any orientation. - [Ignacio Larrosa Cañestro](#), Apr 09 2002. [See [Les Reid](#) link for proof. - [N. J. A. Sloane](#), Apr 02 2016]

Start from cubane and attach amino acids according to the reaction scheme that describes the reaction between the active sites. See the hyperlink on chemistry. - [Robert G. Wilson v](#), Aug 02 2002

For n>0, a(n) = (-1/8)*(coefficient of x in Zagier's polynomial P(2n,n)). (Zagier's polynomials are used by PARI/GP for acceleration of alternating or positive series.)

Figurate numbers based on the 4-dimensional regular convex polytope called the regular 4-simplex, pentachoron, 5-cell, pentaptope or 4-hypertetrahedron with Schläfli symbol {3,3,3}. a(n) = ((n*(n-1)*(n-2)*(n-3))/4!). - [Michael J. Welch](#) (mjwl(AT)ntlworld.com), Apr 01 2004, [R. J. Mathar](#), Jul 07 2009

Maximal number of crossings that can be created by connecting n vertices with straight lines. [Amir D. Faris](#) (redsell(AT)univuln.ca), Jan 30 2019

([ahmedfares\(AT\)my-deja.com](#)), Feb 21 2001

The OEIS is supported by [the many generous donors to the OEIS Foundation](#).

0 1 3 6 2 7
: 13
: 20
23 12
10 22 11 21

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES[®]

founded in 1964 by N. J. A. Sloane

1, 5, 15, 35, 70, 126, 210, 330, 495, 715, 1001, 1365 [Hints](#)

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

Search: **seq:1,3,6,10,15,21,28,3**

Displaying 1-10 of 20 results found

Sort: [relevance](#) | [references](#) | [number](#)

A000217 Triangular numbers:
+ n.
(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, 407, 437, 468, 500, 533, 567, 603, 640, 679, 719, 761, 805, 851, 899, 949, 1001, 1055, 1111, 1169, 1229, 1291, 1355, 1421, 1489, 1559, 1631, 1705, 1781, 1859, 1939, 2021, 2105, 2191, 2279, 2369, 2461, 2555, 2651, 2749, 2849, 2951, 3055, 3161, 3269, 3379, 3491, 3605, 3721, 3839, 3959, 4081, 4205, 4331, 4459, 4589, 4721, 4855, 4991, 5129, 5269, 5411, 5555, 5701, 5849, 5999, 6151, 6305, 6461, 6619, 6779, 6941, 7105, 7271, 7439, 7609, 7781, 7955, 8131, 8309, 8489, 8671, 8855, 9041, 9229, 9419, 9611, 9805, 10001, 10201, 10403, 10607, 10813, 11021, 11231, 11443, 11657, 11873, 12091, 12311, 12533, 12757, 12983, 13211, 13441, 13673, 13907, 14143, 14381, 14621, 14863, 15107, 15353, 15601, 15851, 16103, 16357, 16613, 16871, 17131, 17393, 17657, 17923, 18191, 18461, 18733, 19007, 19283, 19561, 19841, 20123, 20407, 20693, 20981, 21271, 21563, 21857, 22153, 22451, 22751, 23053, 23357, 23663, 23971, 24281, 24593, 24907, 25223, 25541, 25861, 26183, 26507, 26833, 27161, 27491, 27823, 28157, 28493, 28831, 29171, 29513, 29857, 30203, 30551, 30901, 31253, 31607, 31963, 32321, 32681, 33043, 33407, 33773, 34141, 34511, 34883, 35257, 35633, 36011, 36391, 36773, 37157, 37543, 37931, 38321, 38713, 39107, 39503, 39901, 40301, 40703, 41107, 41513, 41921, 42331, 42743, 43157, 43573, 43991, 44411, 44833, 45257, 45683, 46111, 46541, 46973, 47407, 47843, 48281, 48721, 49163, 49607, 50053, 50501, 50951, 51403, 51857, 52313, 52771, 53231, 53693, 54157, 54623, 55091, 55561, 56033, 56507, 56983, 57461, 57941, 58423, 58907, 59393, 59881, 60371, 60863, 61357, 61853, 62351, 62851, 63353, 63857, 64363, 64871, 65381, 65893, 66407, 66923, 67441, 67961, 68483, 69007, 69533, 70061, 70591, 71123, 71657, 72193, 72731, 73271, 73813, 74357, 74903, 75451, 76001, 76553, 77107, 77663, 78221, 78781, 79343, 79907, 80473, 81041, 81611, 82183, 82757, 83333, 83911, 84491, 85073, 85657, 86243, 86831, 87421, 88013, 88607, 89203, 89801, 90401, 91003, 91607, 92213, 92821, 93431, 94043, 94657, 95273, 95891, 96511, 97133, 97757, 98383, 99011, 99641, 100273, 100907, 101543, 102181, 102821, 103463, 104107, 104753, 105401, 106051, 106703, 107357, 108013, 108671, 109331, 110093, 110857, 111623, 112391, 113161, 113933, 114707, 115483, 116261, 117041, 117823, 118607, 119393, 120181, 120971, 121763, 122557, 123353, 124151, 124951, 125753, 126557, 127363, 128171, 128981, 129793, 130607, 131423, 132241, 133061, 133883, 134707, 135533, 136361, 137191, 138023, 138857, 139693, 140531, 141371, 142213, 143057, 143903, 144751, 145601, 146453, 147307, 148163, 149021, 149881, 150743, 151607, 152473, 153341, 154211, 155083, 155957, 156833, 157711, 158591, 159473, 160357, 161243, 162131, 163021, 163913, 164807, 165703, 166601, 167501, 168403, 169307, 170213, 171121, 172031, 172943, 173857, 174773, 175691, 176611, 177533, 178457, 179383, 180311, 181241, 182173, 183107, 184043, 184981, 185921, 186863, 187807, 188753, 189701, 190651, 191603, 192557, 193513, 194471, 195431, 196393, 197357, 198323, 199291, 200261, 201233, 202207, 203183, 204161, 205141, 206123, 207107, 208093, 209081, 210071, 211063, 212057, 213053, 214051, 215051, 216053, 217057, 218063, 219071, 220081, 221093, 222107, 223123, 224141, 225161, 226183, 227207, 228233, 229261, 230291, 231323, 232357, 233393, 234431, 235471, 236513, 237557, 238603, 239651, 240701, 241753, 242807, 243863, 244921, 245981, 247043, 248107, 249173, 250241, 251311, 252383, 253457, 254533, 255611, 256691, 257773, 258857, 259943, 261031, 262121, 263213, 264307, 265403, 266501, 267601, 268703, 269807, 270913, 272021, 273131, 274243, 275357, 276473, 277591, 278711, 279833, 280957, 282083, 283211, 284341, 285473, 286607, 287743, 288881, 290021, 291163, 292307, 293453, 294601, 295751, 296903, 298057, 299213, 300371, 301531, 302693, 303857, 305023, 306191, 307361, 308533, 309707, 310883, 312061, 313241, 314423, 315607, 316793, 317981, 319171, 320363, 321557, 322753, 323951, 325151, 326353, 327557, 328763, 329971, 331181, 332393, 333607, 334823, 336041, 337261, 338483, 339707, 340933, 342161, 343391, 344623, 345857, 347093, 348331, 349571, 350813, 352057, 353303, 354551, 355801, 357053, 358307, 359563, 360821, 362081, 363343, 364607, 365873, 367141, 368411, 369683, 370957, 372233, 373511, 374791, 376073, 377357, 378643, 379931, 381221, 382513, 383807, 385103, 386401, 387701, 389003, 390307, 391613, 392921, 394231, 395543, 396857, 398173, 399491, 400811, 402133, 403457, 404783, 406111, 407441, 408773, 410107, 411443, 412781, 414121, 415463, 416807, 418153, 419501, 420851, 422203, 423557, 424913, 426271, 427631, 428993, 430357, 431723, 433091, 434461, 435833, 437207, 438583, 439961, 441341, 442723, 444107, 445493, 446881, 448271, 449663, 451057, 452453, 453851, 455251, 456653, 458057, 459463, 460871, 462281, 463693, 465107, 466523, 467941, 469361, 470783, 472207, 473633, 475061, 476491, 477923, 479357, 480793, 482231, 483671, 485113, 486557, 488003, 489451, 490901, 492353, 493807, 495263, 496721, 498181, 499643, 501107, 502573, 504041, 505511, 506983, 508457, 509933, 511411, 512891, 514373, 515857, 517343, 518831, 520321, 521813, 523307, 524803, 526301, 527801, 529303, 530807, 532313, 533821, 535331, 536843, 538357, 539873, 541391, 542911, 544433, 545957, 547483, 549011, 550541, 552073, 553607, 555143, 556681, 558221, 559763, 561307, 562853, 564401, 565951, 567503, 569057, 570613, 572171, 573731, 575293, 576857, 578423, 579991, 581561, 583133, 584707, 586283, 587861, 589441, 591023, 592607, 594193, 595781, 597371, 598963, 600557, 602153, 603751, 605351, 606953, 608557, 610163, 611771, 613381, 614993, 616607, 618223, 619841, 621461, 623083, 624707, 626333, 627961, 629591, 631223, 632857, 634493, 636131, 637771, 639413, 641057, 642703, 644351, 646001, 647653, 649307, 650963, 652621, 654281, 655943, 657607, 659273, 660941, 662611, 664283, 665957, 667633, 669311, 670991, 672673, 674357, 676043, 677731, 679421, 681113, 682807, 684503, 686201, 687901, 689603, 691307, 693013, 694721, 696431, 698143, 699857, 701573, 703291, 705011, 706733, 708457, 710183, 711911, 713641, 715373, 717107, 718843, 720581, 722321, 724063, 725807, 727553, 729301, 731051, 732803, 734557, 736313, 738071, 739831, 741593, 743357, 745123, 746891, 748661, 750433, 752207, 753983, 755761, 757541, 759323, 761107, 762893, 764681, 766471, 768263, 770057, 771853, 773651, 775451, 777253, 779057, 780863, 782671, 784481, 786293, 788107, 789923, 791741, 793561, 795383, 797207, 799033, 800861, 802691, 804523, 806357, 808193, 810031, 811871, 813713, 815557, 817403, 819251, 821101, 822953, 824807, 826663, 828521, 830381, 832243, 834107, 835973, 837841, 839711, 841583, 843457, 845333, 847211, 849091, 850973, 852857, 854743, 856631, 858521, 860413, 862307, 864203, 866101, 868001, 869903, 871807, 873713, 875621, 877531, 879443, 881357, 883273, 885191, 887111, 889033, 890957, 892883, 894811, 896741, 898673, 900607, 902543, 904481, 906421, 908363, 910307, 912253, 914201, 916151, 918103, 920057, 922013, 923971, 925931, 927893, 929857, 931823, 933791, 935761, 937733, 939707, 941683, 943661, 945641, 947623, 949607, 951593, 953581, 955571, 957563, 959557, 961553, 963551, 965551, 967553, 969557, 971563, 973571, 975581, 977593, 979607, 981623, 983641, 985661, 987683, 989707, 991733, 993761, 995791, 997823, 999857, 1001893, 1003931, 1005971, 1008013, 1010057, 1012103, 1014151, 1016201, 1018253, 1020307, 1022363, 1024421, 1026481, 1028543, 1030607, 1032673, 1034741, 1036811, 1038883, 1040957, 1043033, 1045111, 1047191, 1049273, 1051357, 1053443, 1055531, 1057621, 1059713, 1061807, 1063903, 1066001, 1068101, 1070203, 1072307, 1074413, 1076521, 1078631, 1080743, 1082857, 1084973, 1087091, 1089211, 1091333, 1093457, 1095583, 1097711, 1099841, 1101973, 1104107, 1106243, 1108381, 1110521, 1112663, 1114807, 1116953, 1119101, 1121251, 1123403, 1125557, 1127713, 1129871, 1132031, 1134193, 1136357, 1138523, 1140691, 1142861, 1145033, 1147207, 1149383, 1151561, 1153741, 1155923, 1158107, 1160293, 1162481, 1164671, 1166863, 1169057, 1171253, 1173451, 1175651, 1177853, 1180057, 1182263, 1184471, 1186681, 1188893, 1191107, 1193323, 1195541, 1197761, 1200083, 1202307, 1204533, 1206761, 1208991, 1211223, 1213457, 1215693, 1217931, 1220171, 1222413, 1224657, 1226903, 1229151, 1231401, 1233653, 1235907, 1238163, 1240421, 1242681, 1244943, 1247207, 1249473, 1251741, 1254011, 1256283, 1258557, 1260833, 1263111, 1265391, 1267673, 1269957, 1272243, 1274531, 1276821, 1279113, 1281407, 1283703, 1286001, 1288301, 1290603, 1292907, 1295213, 1297521,

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...
- Zahlen anstarren

Genauere Analyse ALGONE

```
algOne(x: ℕ, y: 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

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...
- Zahlen anstarren

Schritt 2

- Vermutung bilden:

Genauere Analyse ALGONE

```
algOne(x: ℕ, y: 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:
 - $T(x, y) \leq 2 \binom{y}{x} - 1$

Genauere Analyse ALGONE

```
algOne(x: ℕ, y: 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: ℕ, y: 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)
```

Beweis per Induktion

Schritt 1

- Kosten ausrechnen...
- Zahlen anstarren

Schritt 2

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

Genauere Analyse ALGONE

```
algOne(x: ℕ, y: 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

Beweis per Induktion

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

Genaue Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

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

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

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

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

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

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

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

Genauere Analyse ALGONE

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

if $x = 0 \vee x = y$ **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 $\binom{n}{k} = 0$.

So

- $\binom{n}{0} = 1 = \binom{n}{n}$

- $\binom{n}{1} = n = \binom{n}{n-1}$

- $\binom{n}{k} = \frac{n-k+1}{k} \binom{n}{k-1}$

- $\binom{n}{k} = \frac{n}{k} \cdot \binom{n-1}{k-1} \Leftrightarrow k \cdot \binom{n}{k} = n \cdot \binom{n-1}{k-1}$

- $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$. Für $n = k$ ist der rechte Summand $\binom{n}{k+1} = 0$.

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

- 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 Werte von x und y

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

$$= 2 \binom{y-1}{x-1} - 1 + 2 \binom{y-1}{x} - 1 + 1$$

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

$$= 2 \binom{y}{x} - 1$$

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

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

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

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

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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: ℕ, y: 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

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...
 - $\binom{y}{x} = \binom{y-1}{x-1} + \binom{y-1}{x} \dots ?$

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

Genauere Analyse ALGONE

```

algOne(x: ℕ, y: 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

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

:-)

Motivation: Datenstrukturen

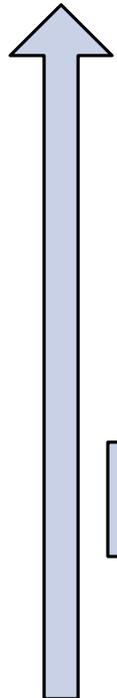
Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher

Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher

hoch



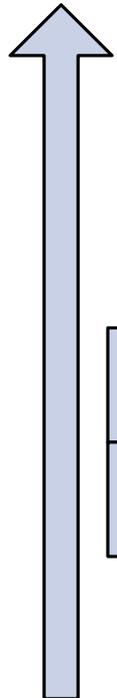
tief

Manuelles Verwalten von Speicheradressen

Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher

hoch



tief

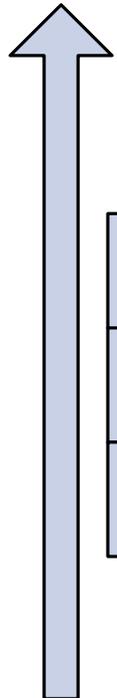
Arrays, verzeigerte Strukturen, Structs / Objekte

Manuelles Verwalten von Speicheradressen

Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher

hoch



Datenstrukturen (z.B. Listen, Queues, etc.)

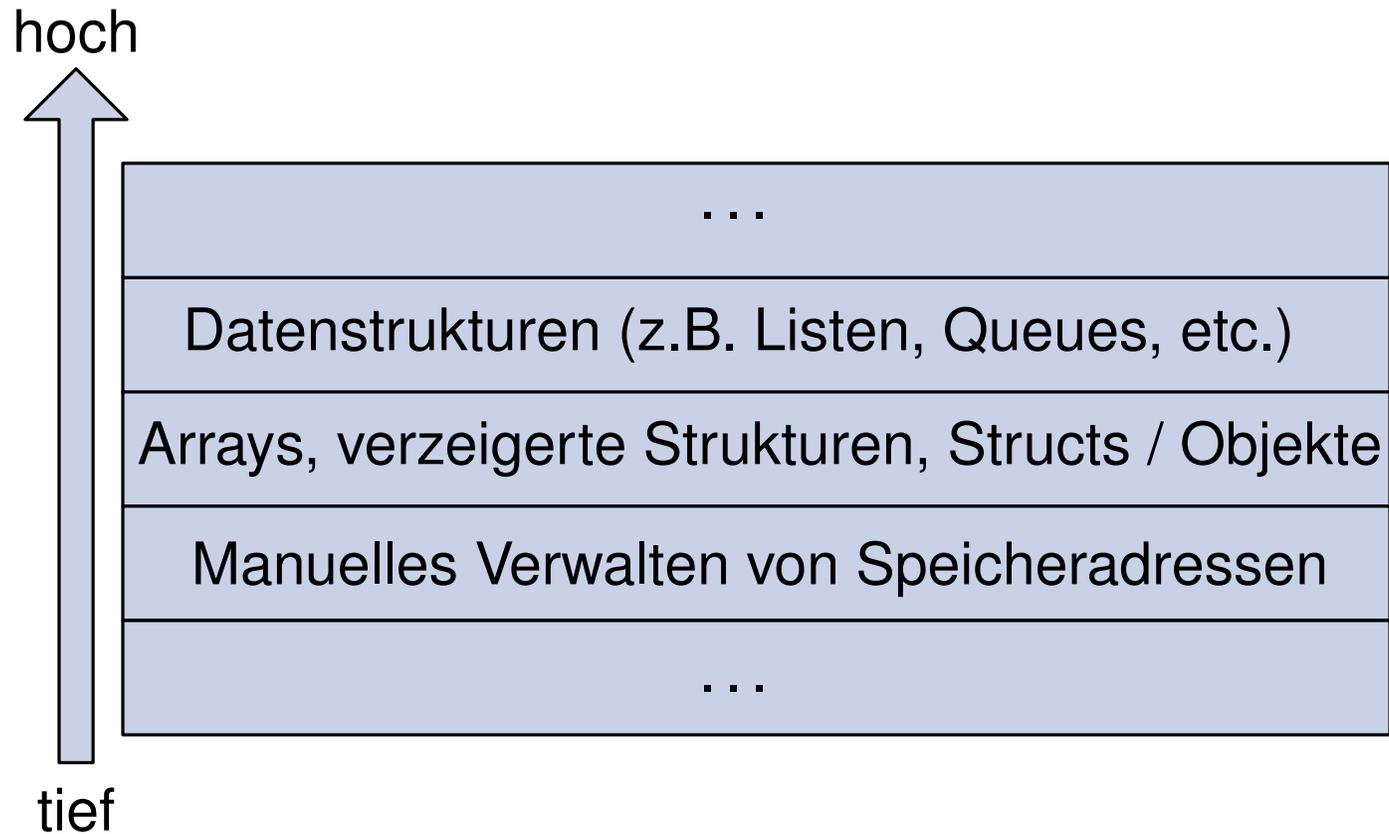
Arrays, verzeigerte Strukturen, Structs / Objekte

Manuelles Verwalten von Speicheradressen

tief

Motivation: Datenstrukturen

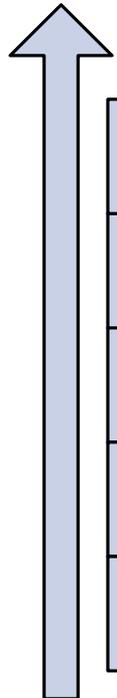
Abstraktionsebenen auf dem Speicher



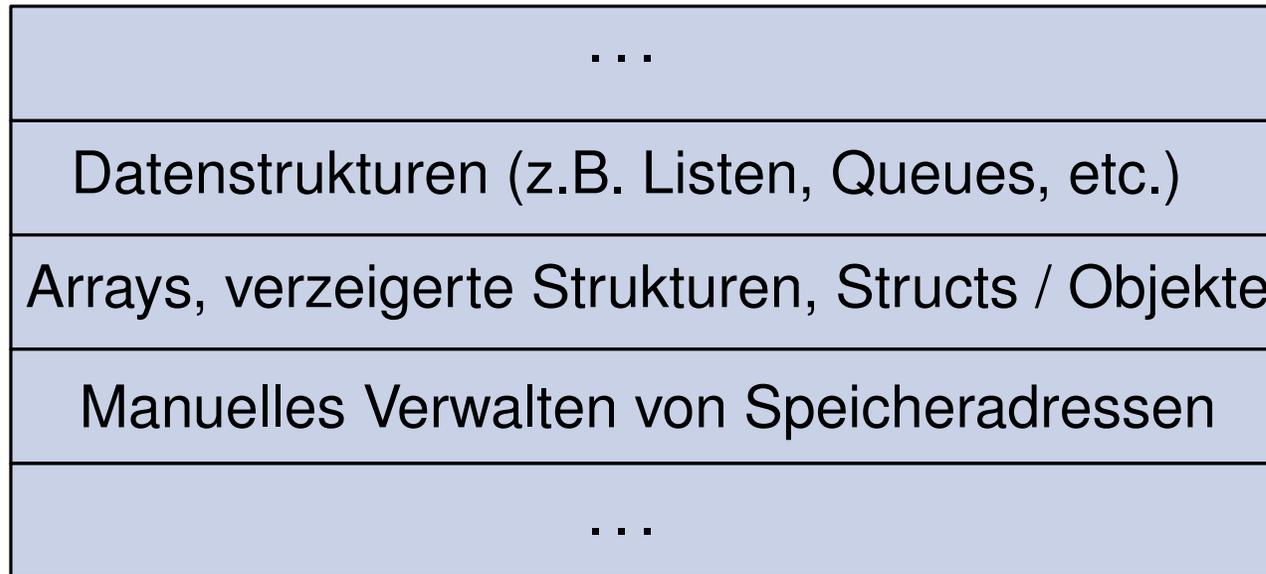
Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher

hoch



tief



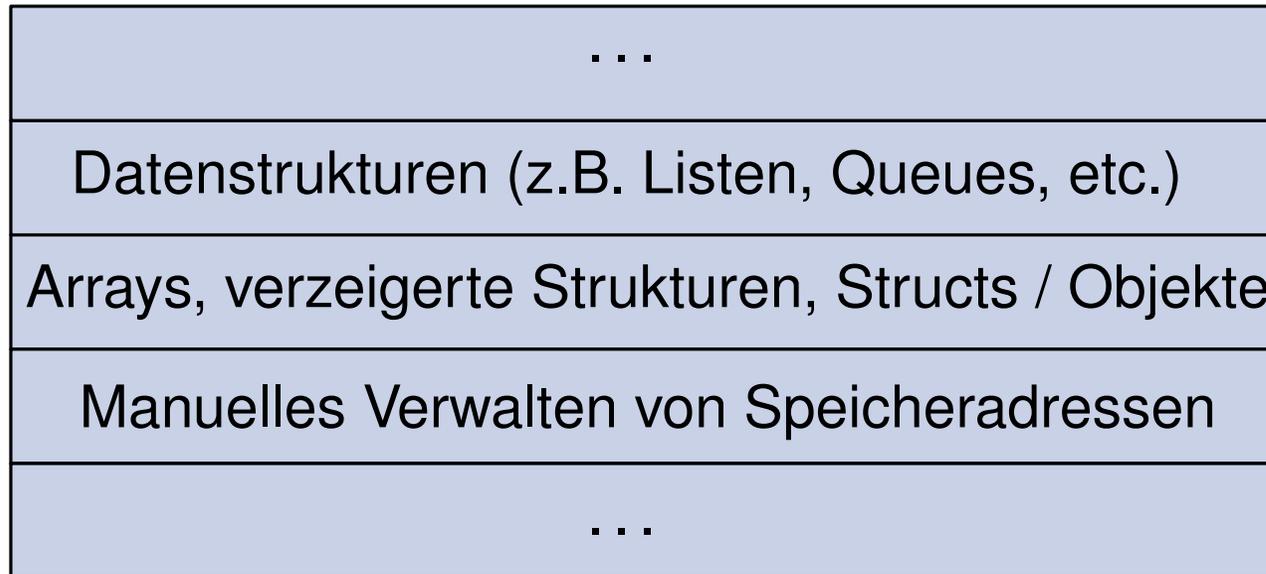
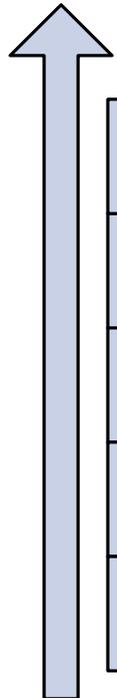
Bsp: Queue

- `pushBack(e)`
- `popFront()`
- `size()`

Motivation: Datenstrukturen

Abstraktionsebenen auf dem Speicher

hoch



Bsp: Queue

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

z.B. mittels verketteter Liste

tief

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

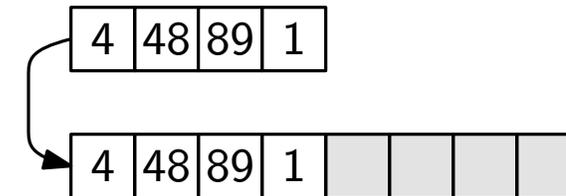
4	48	89	1
---	----	----	---



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

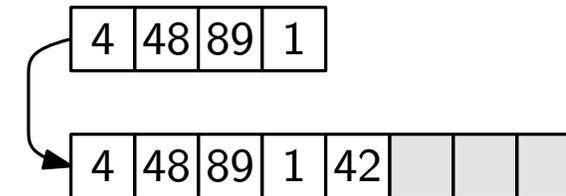
Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

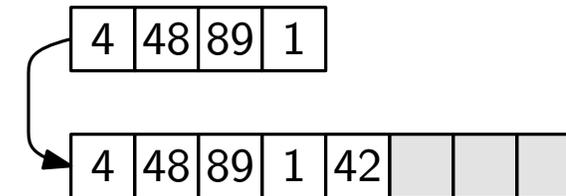
Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

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

Beispiel: unbeschränktes Array



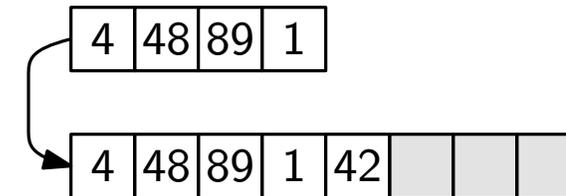
Motivation: Amortisierte Analyse

- 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

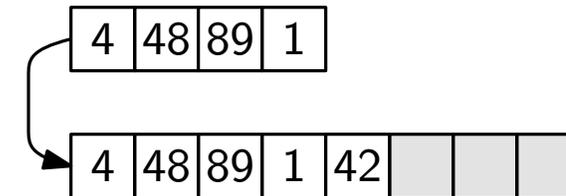
- Laufzeit von Operationen auf DS nicht immer gleich
 - z.B.: Operation meist günstig, seltener teurer

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

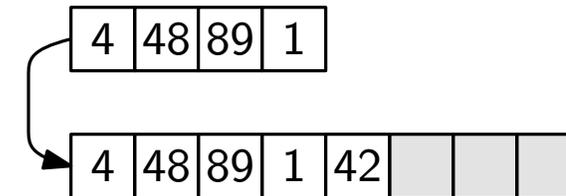
Idee

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

Verschiedene Techniken zur Analyse

- Aggregation

Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

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

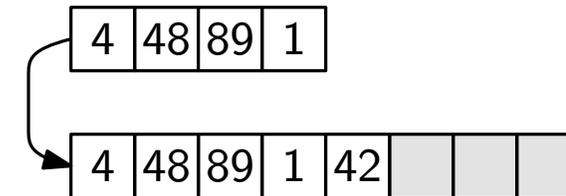
Idee

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

Verschiedene Techniken zur Analyse

- Aggregation
- Charging

Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

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

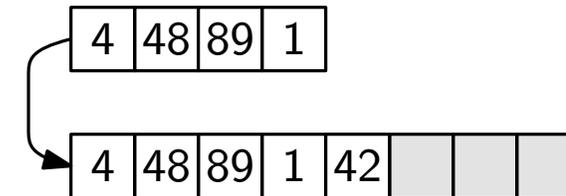
Idee

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

Verschiedene Techniken zur Analyse

- Aggregation
- Charging
- Konto

Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

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

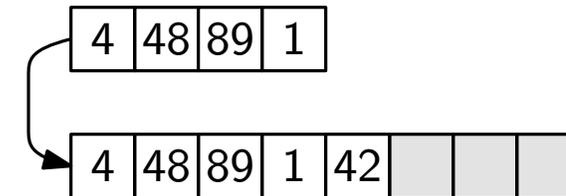
Idee

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

Verschiedene Techniken zur Analyse

- Aggregation
- Charging
- Konto
- Potential

Beispiel: unbeschränktes Array



Beispiel: Binärzähler

Idee

- Array A verwaltet Bits

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits



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

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

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

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

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	0	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	1	0	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	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	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

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
   $A[i] := 0$ 
   $i := i + 1$ 
 $A[i] := 1$ 

```

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	0	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	1	0	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	0	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	1	0	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	0	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	1	0	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	0	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---

...

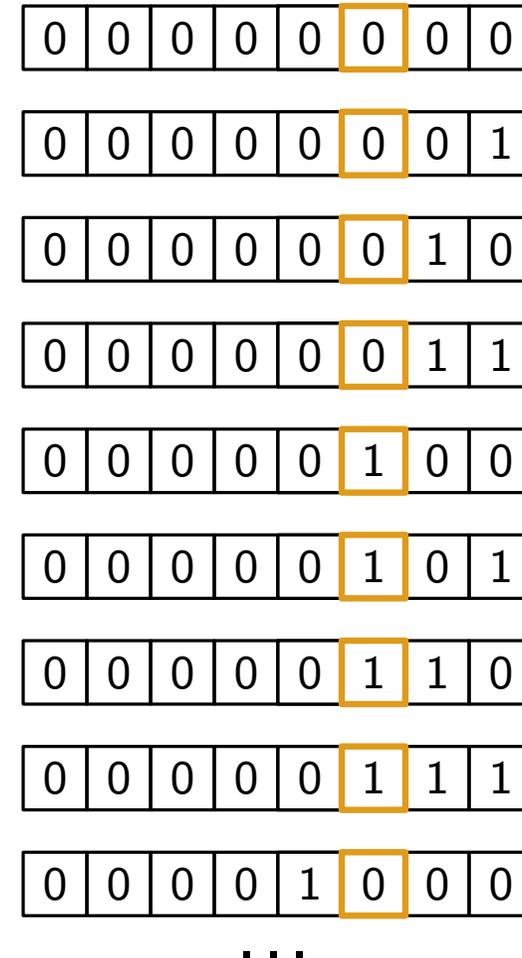
Analyse: Aggregatmethode

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



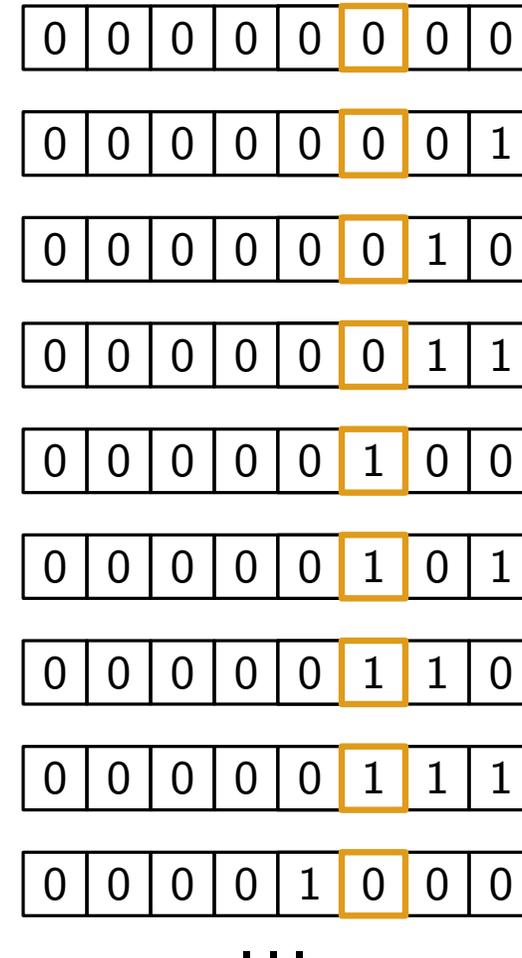
Analyse: Aggregatmethode

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



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:

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

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:

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

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
---	---	---	---	---	---	---	---

...

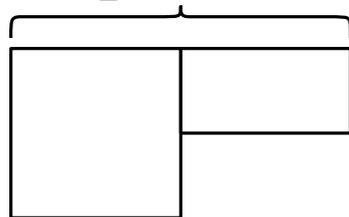
Analyse: Aggregatmethode

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
---	---	---	---	---	---	---	---

...

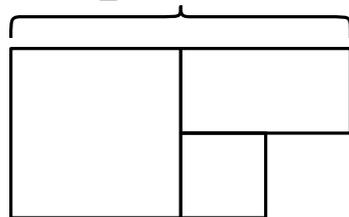
Analyse: Aggregatmethode

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:

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



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
---	---	---	---	---	---	---	---

...

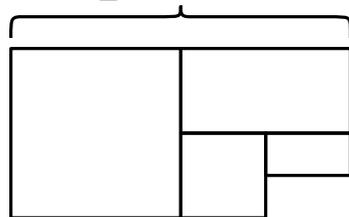
Analyse: Aggregatmethode

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
---	---	---	---	---	---	---	---

...

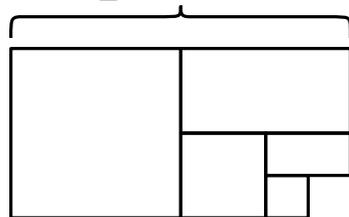
Analyse: Aggregatmethode

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:

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



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
---	---	---	---	---	---	---	---

...

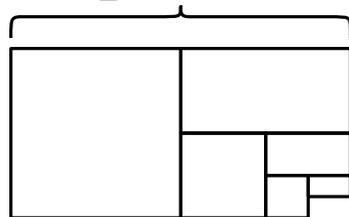
Analyse: Aggregatmethode

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
---	---	---	---	---	---	---	---

...

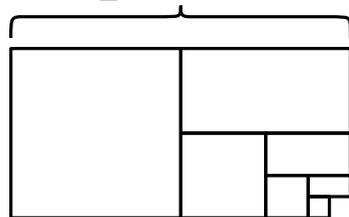
Analyse: Aggregatmethode

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:

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



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
---	---	---	---	---	---	---	---

...

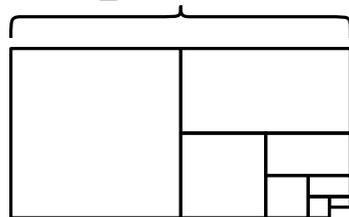
Analyse: Aggregatmethode

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
---	---	---	---	---	---	---	---

...

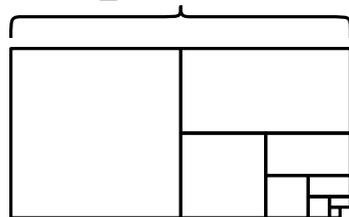
Analyse: Aggregatmethode

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:

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



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:

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

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

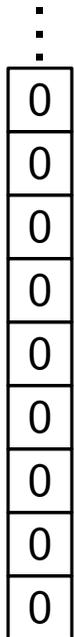
Idee

- teure Operationen legen Kosten um auf günstige Operationen

Analyse mittels Charging

Idee

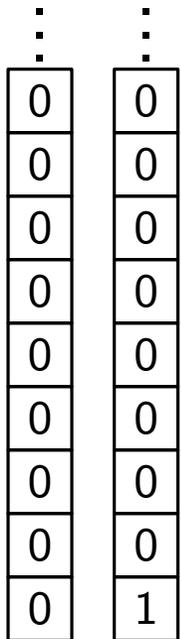
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

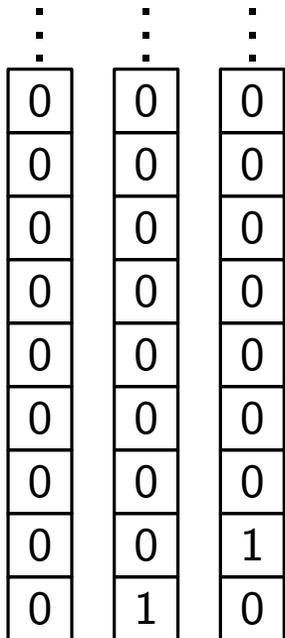
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

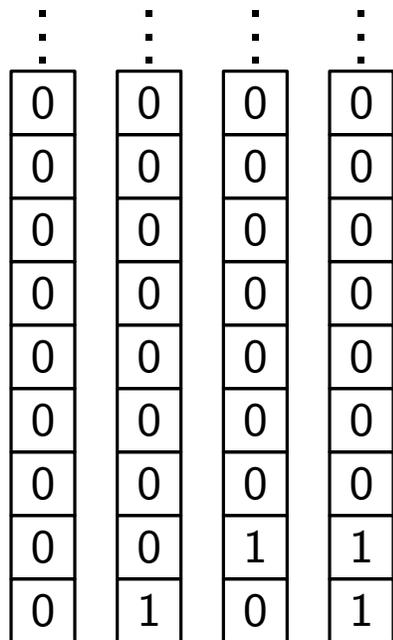
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

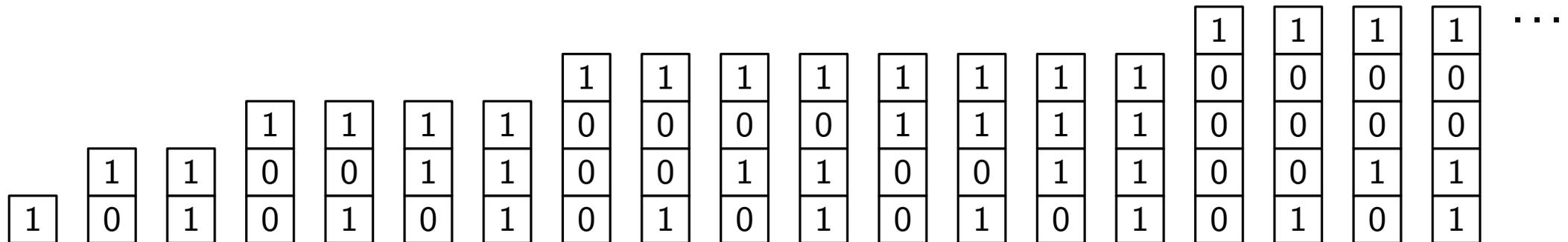
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

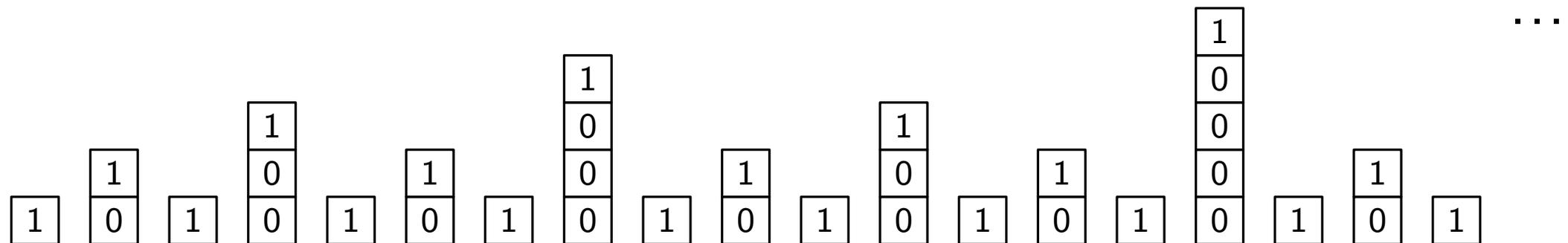
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

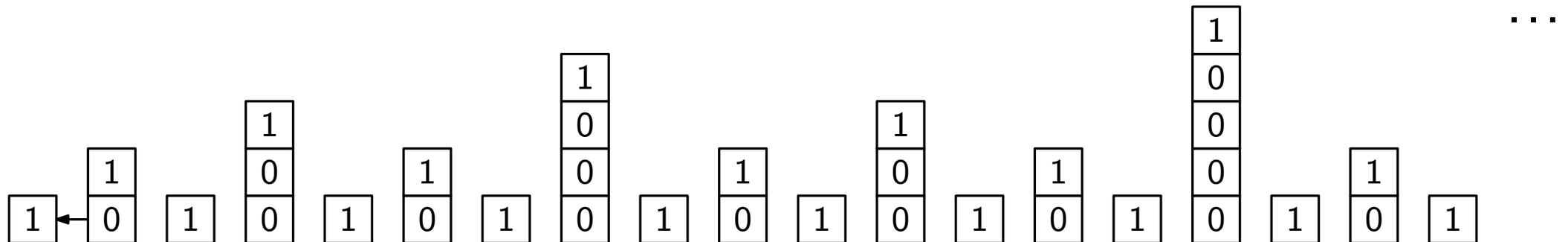
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

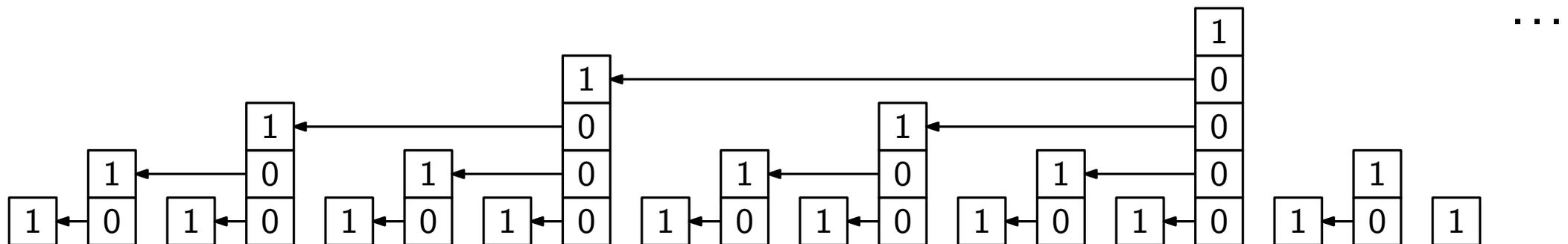
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen



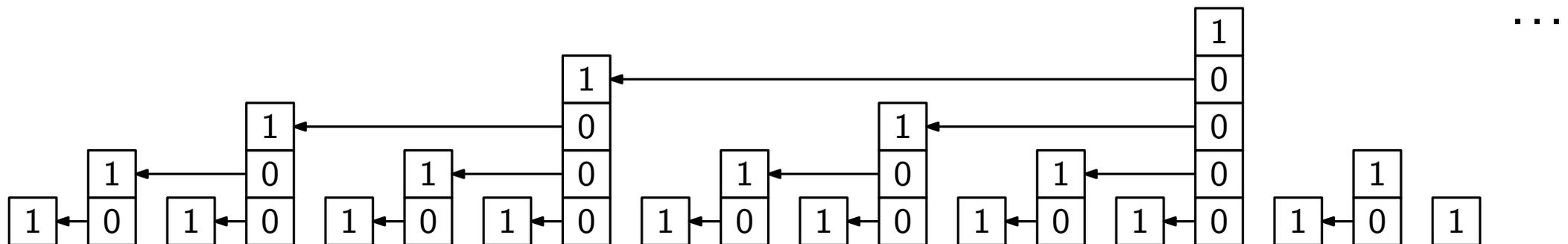
Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:

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



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

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

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

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

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?

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

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?

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

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

Analyse: Potentialmethode

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?

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

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. $i > 0$ 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
---	---	---	---	---	---	---	---

...

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. :)*
- tatsächliche 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. :)*
- tatsächliche 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. :)*
- tatsächliche 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. :)*
- tatsächliche 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. :)*
- tatsächliche 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. :)*
- tatsächliche 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 ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



Beispiel 2: Stacks und Queues

Wiederholung

Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



Beispiel 2: Stacks und Queues

Wiederholung

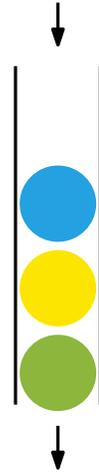
Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



Beispiel 2: Stacks und Queues

Wiederholung

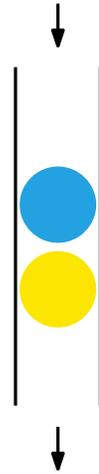
Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



Beispiel 2: Stacks und Queues

Wiederholung

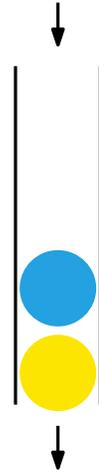
Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



Beispiel 2: Stacks und Queues

Wiederholung

Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



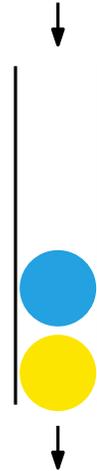
Beispiel 2: Stacks und Queues

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

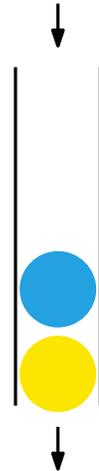
Beispiel 2: Stacks und Queues

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**() $O(1)$

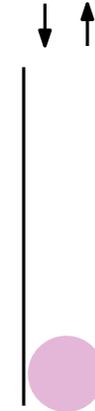
z.B. mittels verketteter Liste



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**() $O(1)$

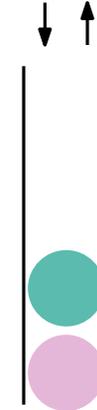
z.B. mittels verketteter Liste



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



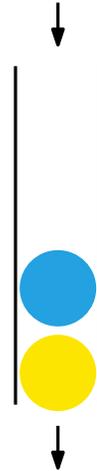
Beispiel 2: Stacks und Queues

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

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

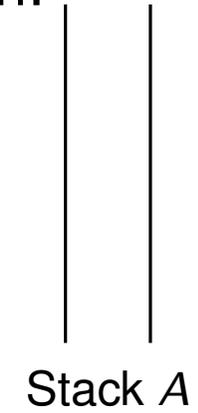


Frage

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

Eine Queue aus Stacks

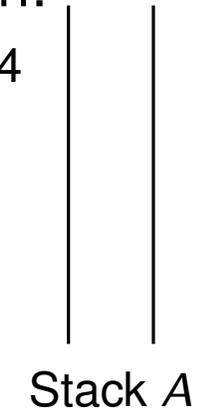
Operationen:



Eine Queue aus Stacks

Operationen:

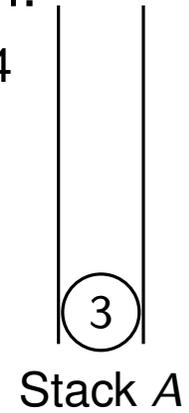
■ **push**: 3, 1, 4



Eine Queue aus Stacks

Operationen:

■ **push:** 3, 1, 4



Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

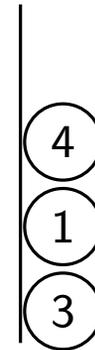


Stack A

Eine Queue aus Stacks

Operationen:

■ **push:** 3, 1, 4



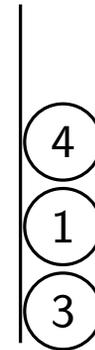
Stack A

Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

■ **pop**()



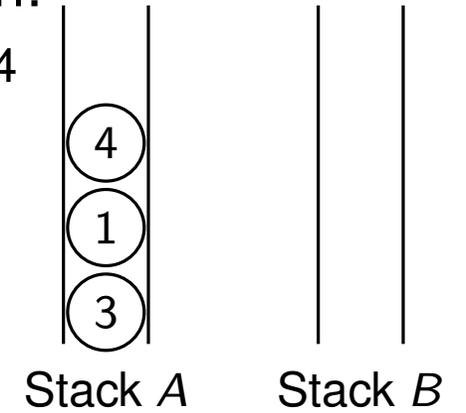
Stack A

Eine Queue aus Stacks

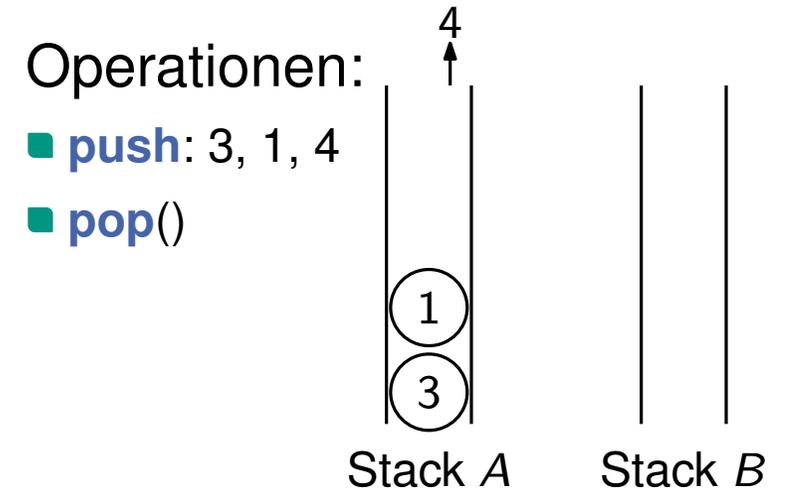
Operationen:

■ **push**: 3, 1, 4

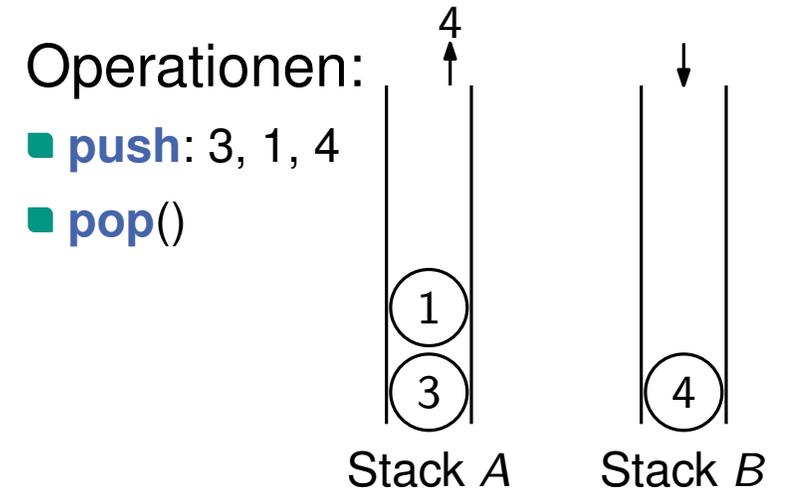
■ **pop**()



Eine Queue aus Stacks



Eine Queue aus Stacks

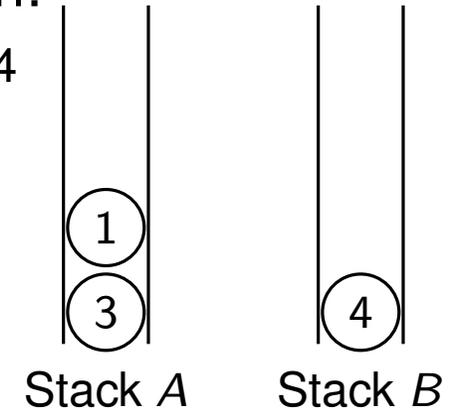


Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

■ **pop**()

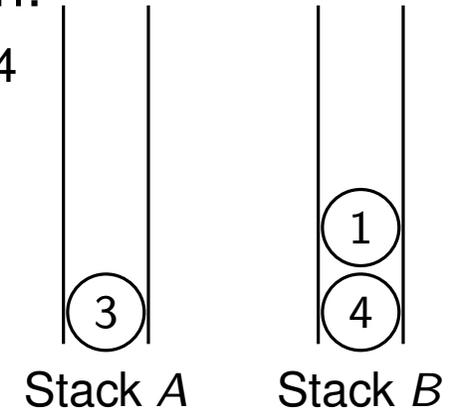


Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

■ **pop**()

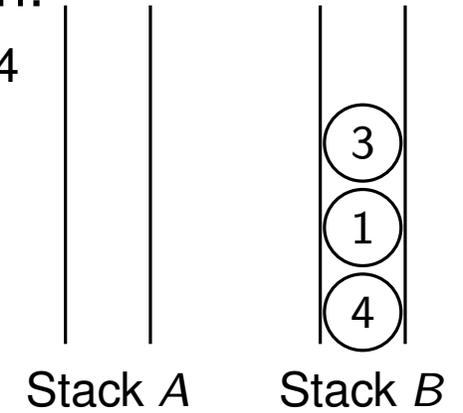


Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

■ **pop**()

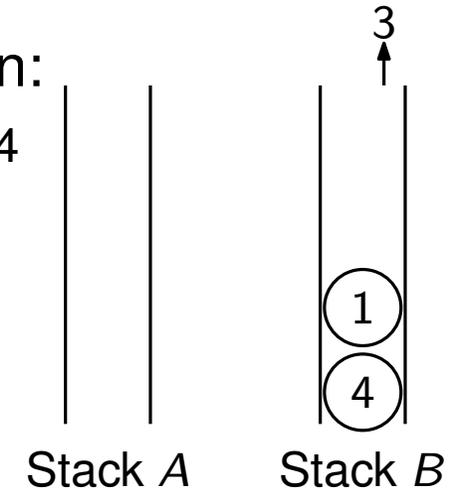


Eine Queue aus Stacks

Operationen:

■ **push:** 3, 1, 4

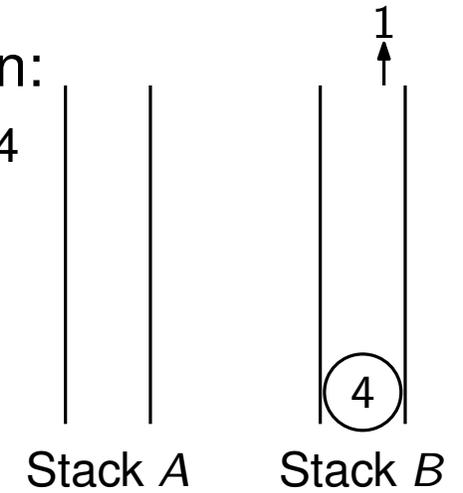
■ **pop()**



Eine Queue aus Stacks

Operationen:

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



Eine Queue aus Stacks

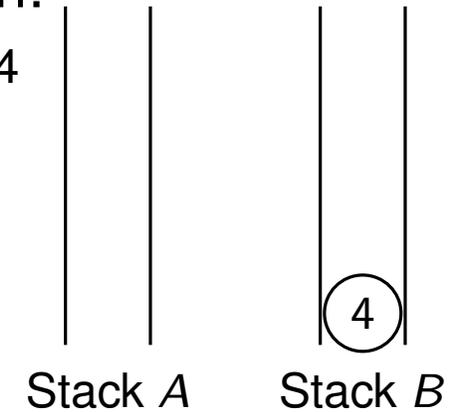
Operationen:

■ **push:** 3, 1, 4

■ **pop()**

■ **pop()**

■ **push:** 1,5



Eine Queue aus Stacks

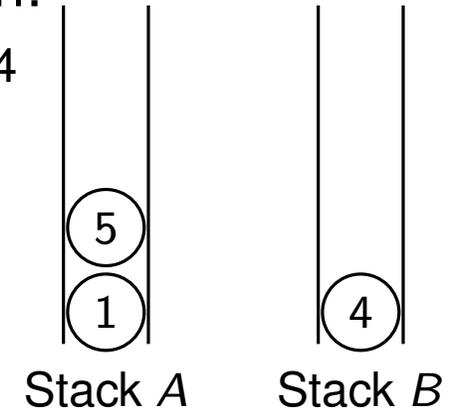
Operationen:

■ **push:** 3, 1, 4

■ **pop()**

■ **pop()**

■ **push:** 1,5



Eine Queue aus Stacks

Algorithmische Umsetzung

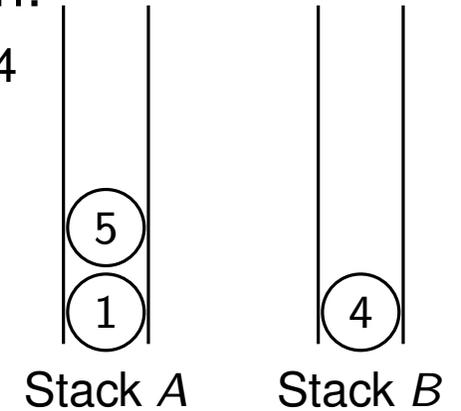
Operationen:

■ **push:** 3, 1, 4

■ **pop()**

■ **pop()**

■ **push:** 1,5



Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen

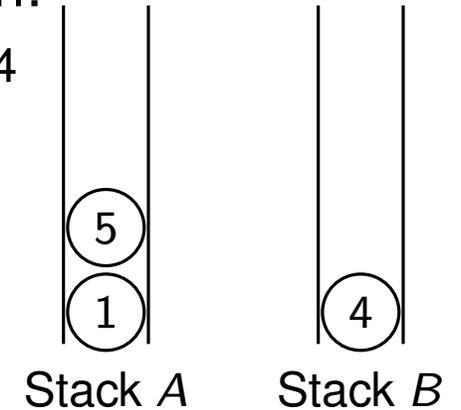
Operationen:

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

- **pop**()

- **pop**()

- **push**: 1,5



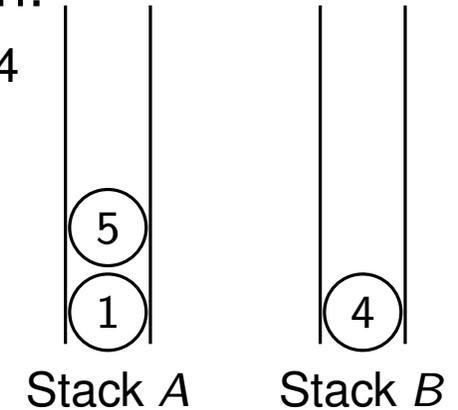
Eine Queue aus Stacks

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

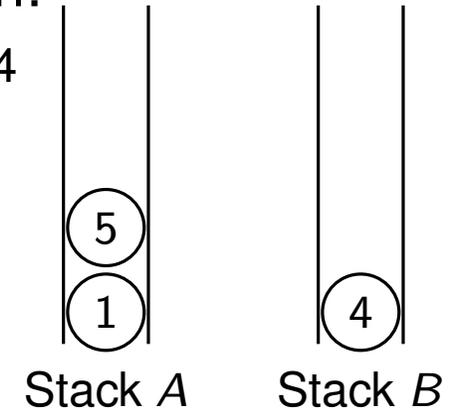
Operationen:

■ **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

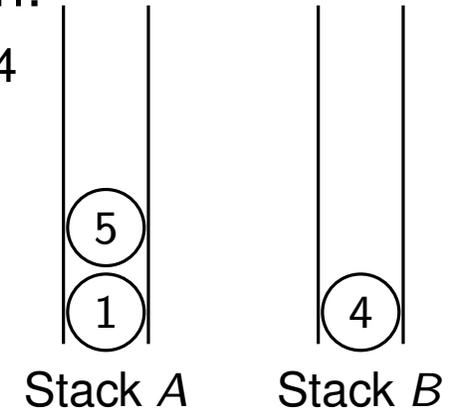
Operationen:

■ **push**: 3, 1, 4

■ **pop**()

■ **pop**()

■ **push**: 1,5



Eine Queue aus Stacks

Algorithmische Umsetzung

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

$O(1)$

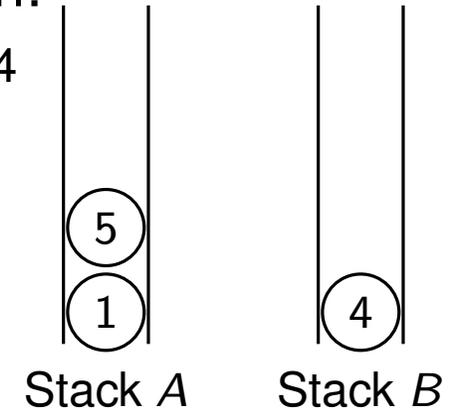
Operationen:

■ **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

$O(1)$

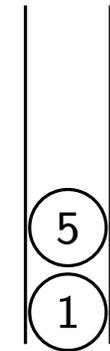
Operationen:

■ **push**: 3, 1, 4

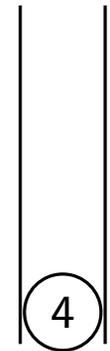
■ **pop**()

■ **pop**()

■ **push**: 1,5



Stack A



Stack B

$O(1)$

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

$O(1)$

$O(1)$

$O(|A|)$

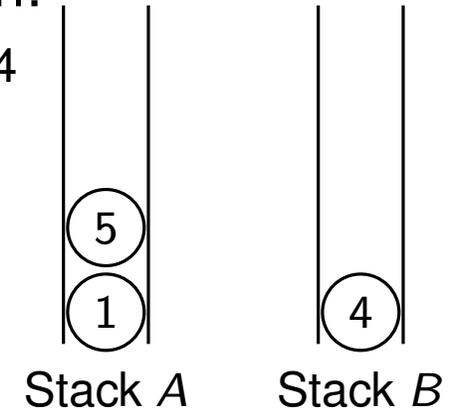
Operationen:

■ **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

$O(1)$

$O(|A|)$

$O(1)$

$O(|A|)$

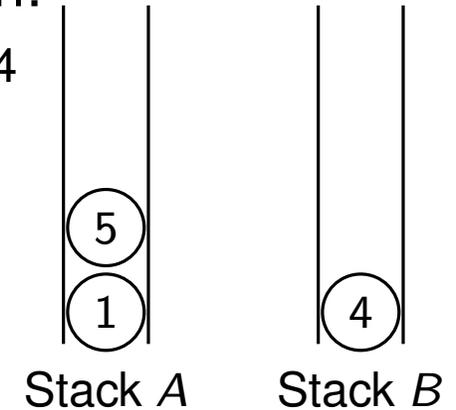
Operationen:

■ **push**: 3, 1, 4

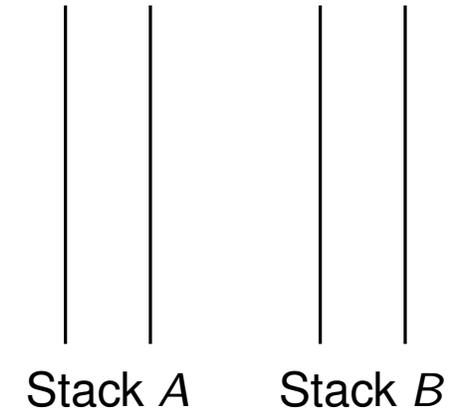
■ **pop**()

■ **pop**()

■ **push**: 1,5

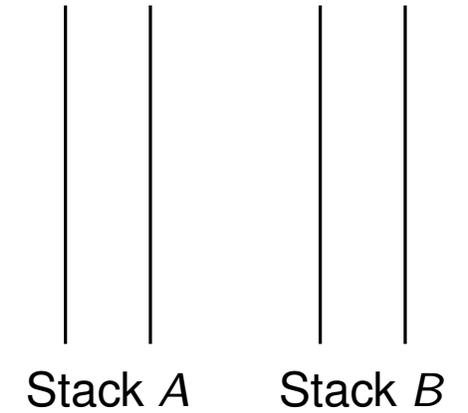


Analyse: Kontomethode



Analyse: Kontomethode

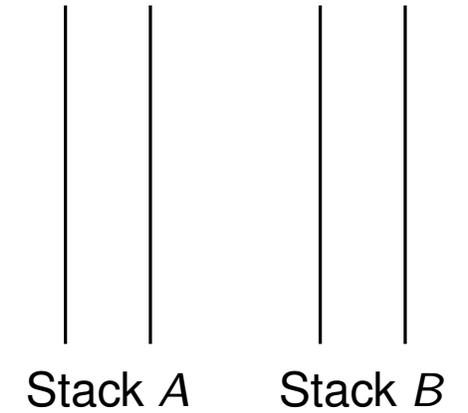
Idee



Analyse: Kontomethode

Idee

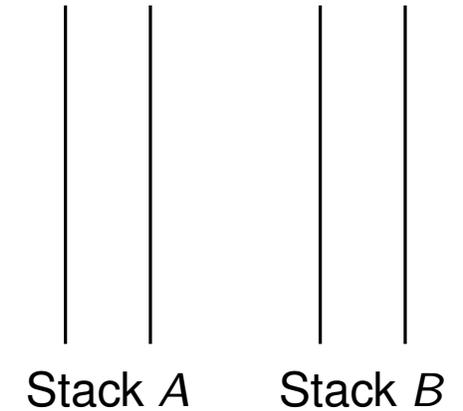
- günstige Operationen bauen Guthaben auf



Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

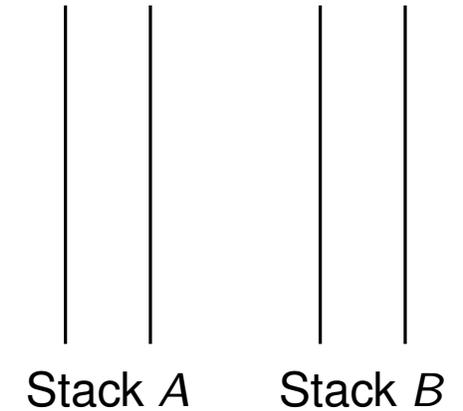


Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier



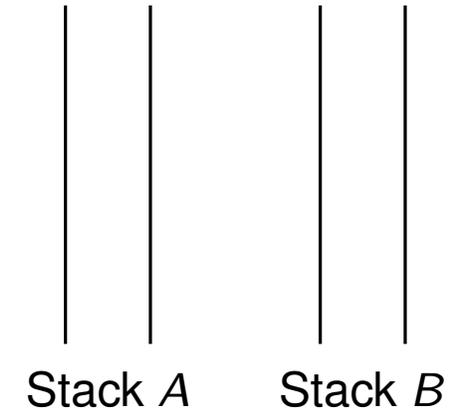
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

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



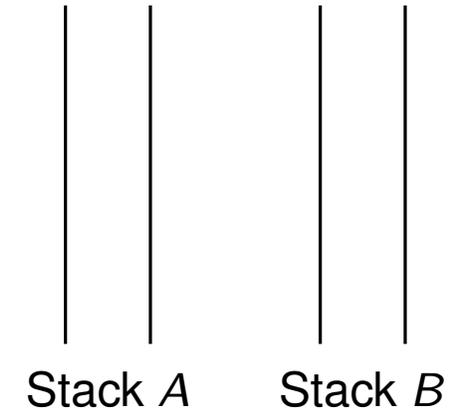
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

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



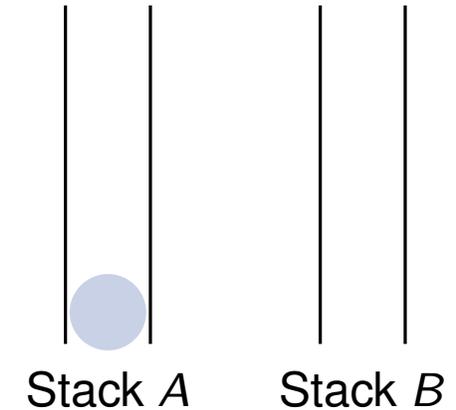
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

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



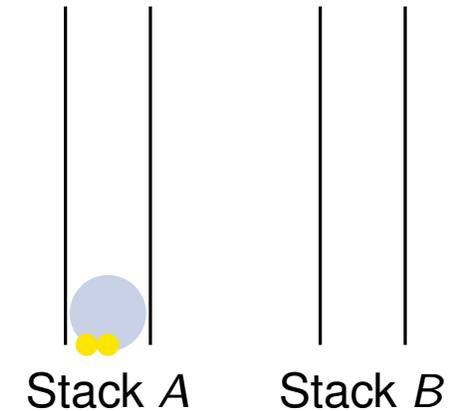
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

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



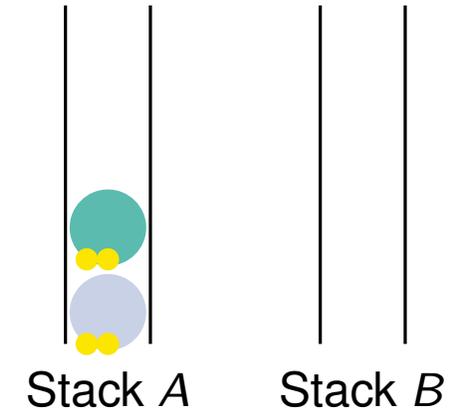
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

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



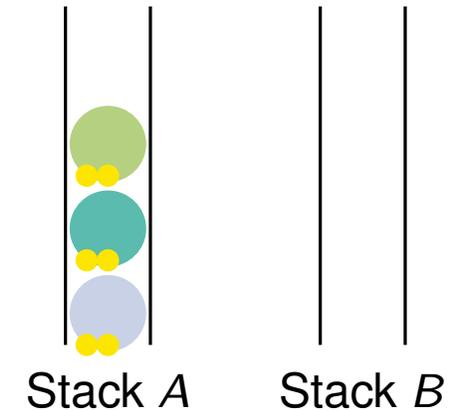
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

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



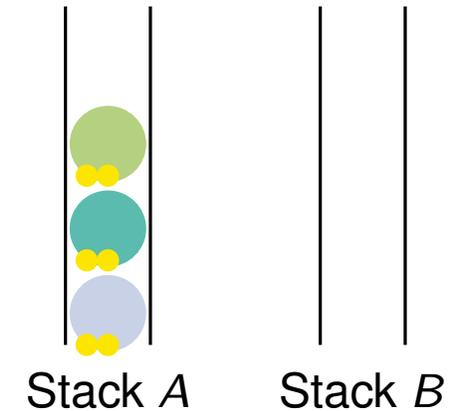
Analyse: Kontomethode

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



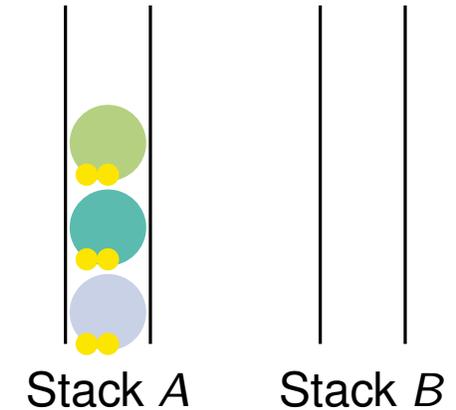
Analyse: Kontomethode

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()*



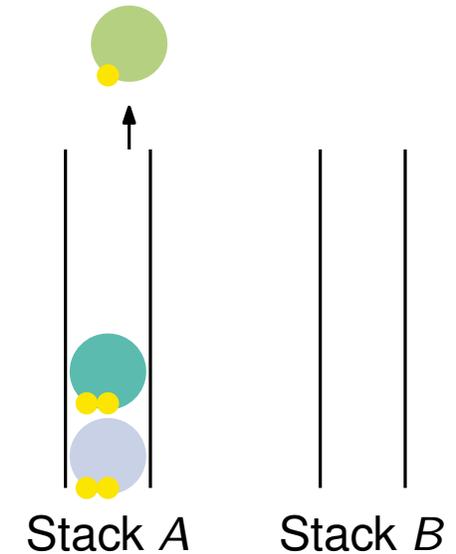
Analyse: Kontomethode

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()*



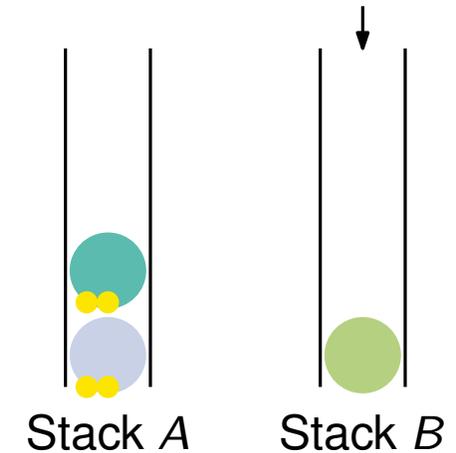
Analyse: Kontomethode

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()*



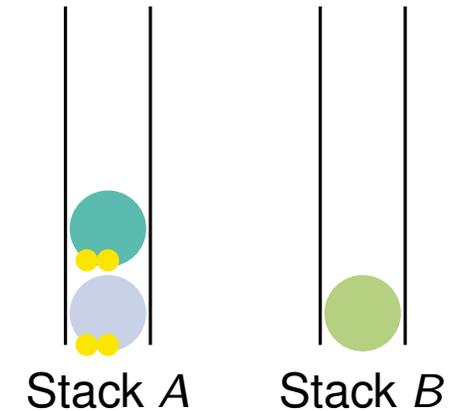
Analyse: Kontomethode

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()*



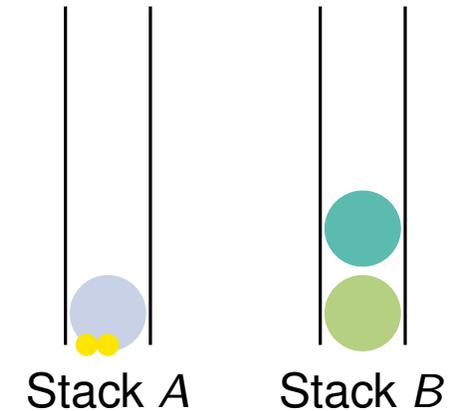
Analyse: Kontomethode

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()*



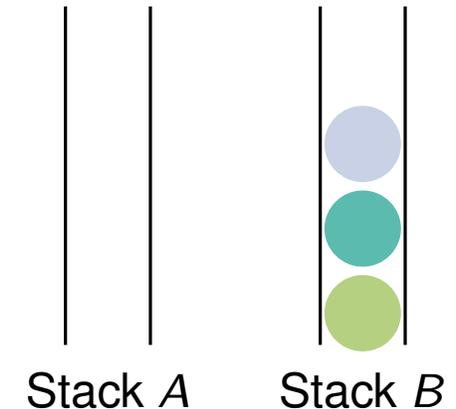
Analyse: Kontomethode

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 $A.\text{pop}()$ und $B.\text{push}()$



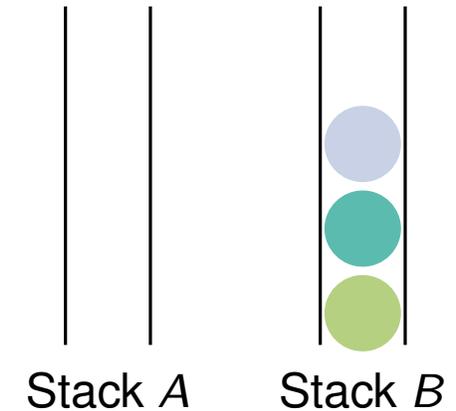
Analyse: Kontomethode

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 $A.\text{pop}()$ und $B.\text{push}()$ (für alle Elemente in A)



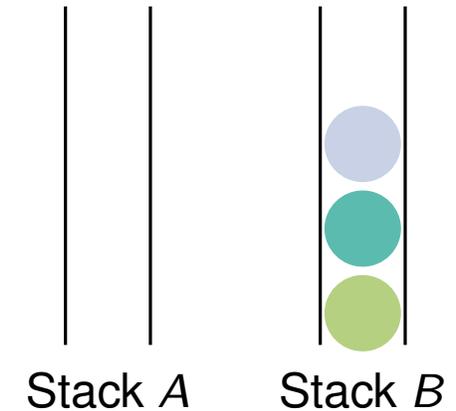
Analyse: Kontomethode

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



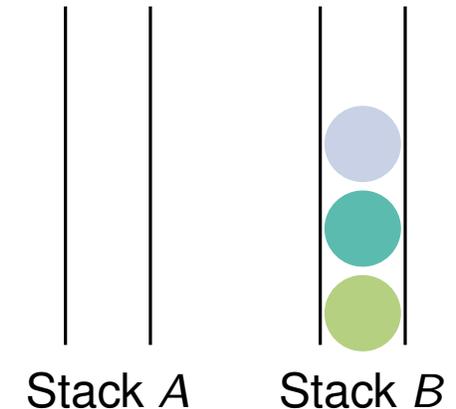
Analyse: Kontomethode

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



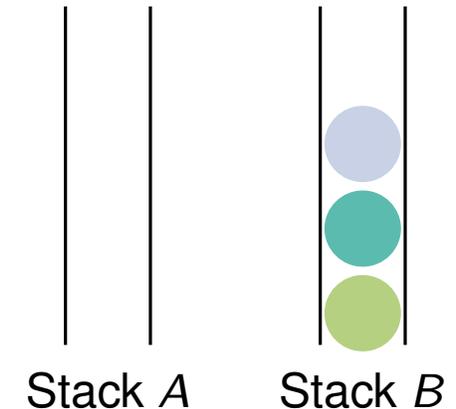
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

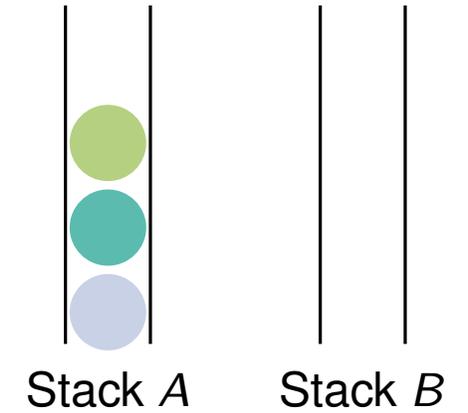
Umsetzung hier

- 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

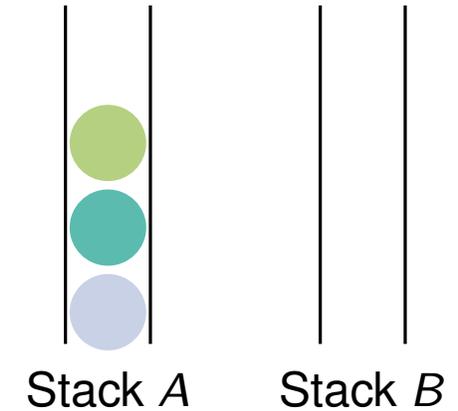
Potentialfunktion



Analyse: Potentialmethode

Potentialfunktion

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

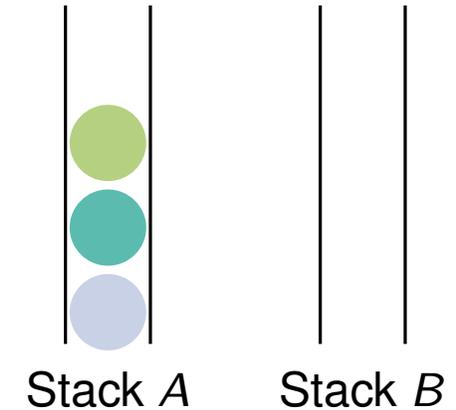


Analyse: Potentialmethode

Potentialfunktion

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

Analyse



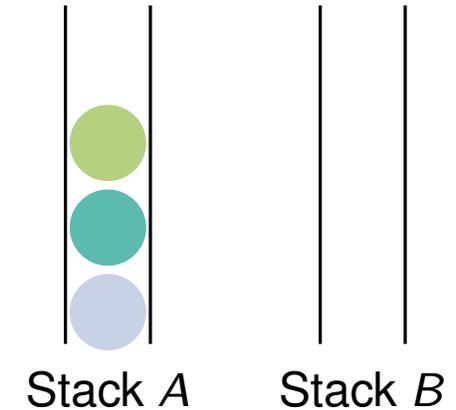
Analyse: Potentialmethode

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. $i > 0$



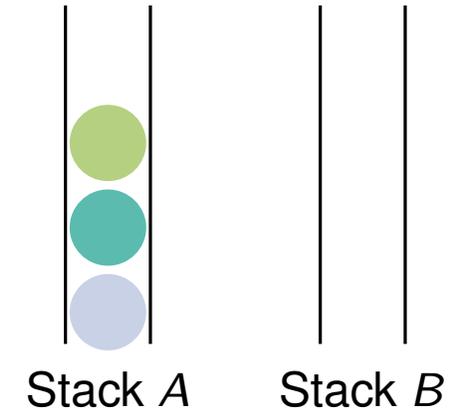
Analyse: Potentialmethode

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)



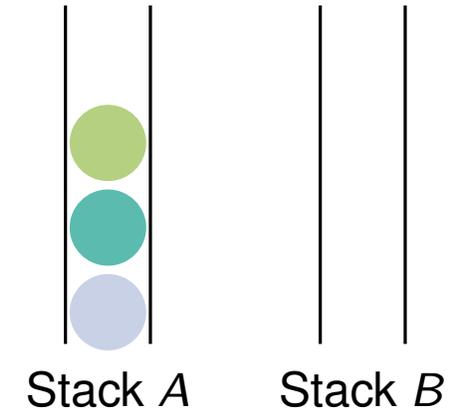
Analyse: Potentialmethode

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**:



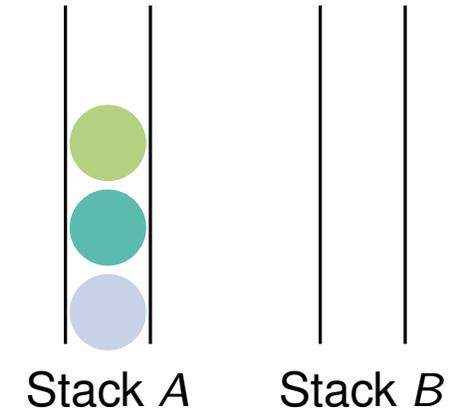
Analyse: Potentialmethode

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

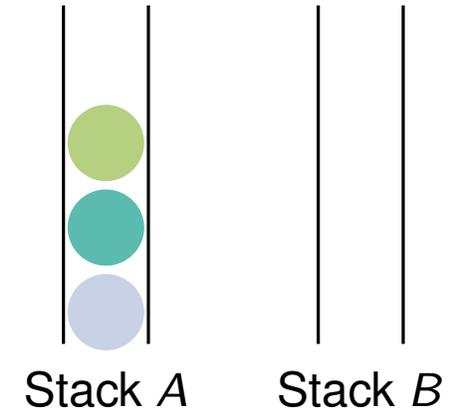
Analyse: Potentialmethode

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

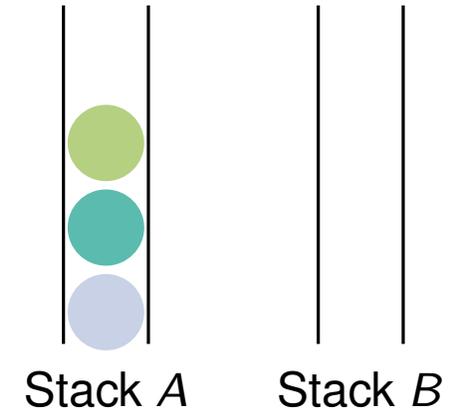
Analyse: Potentialmethode

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

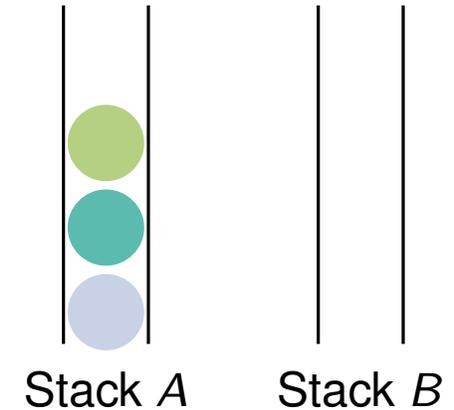
Analyse: Potentialmethode

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

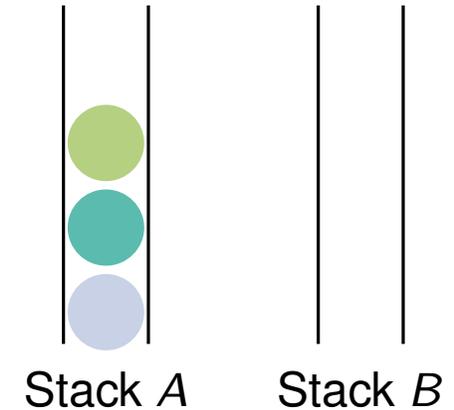
Analyse: Potentialmethode

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

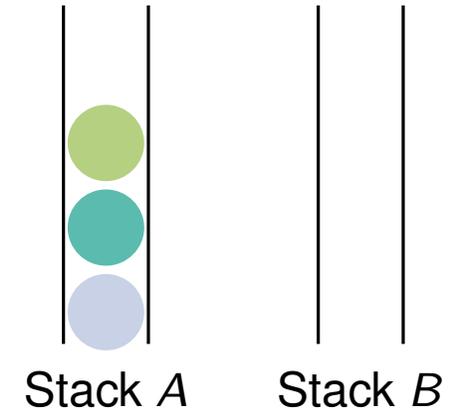
Analyse: Potentialmethode

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

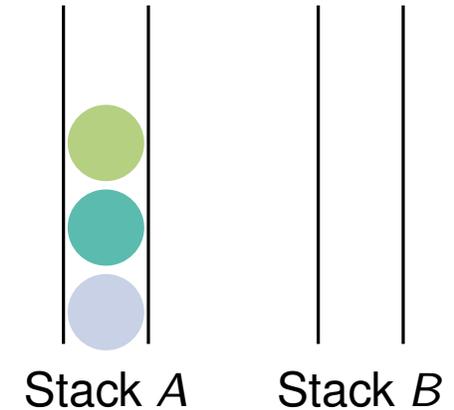
Analyse: Potentialmethode

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

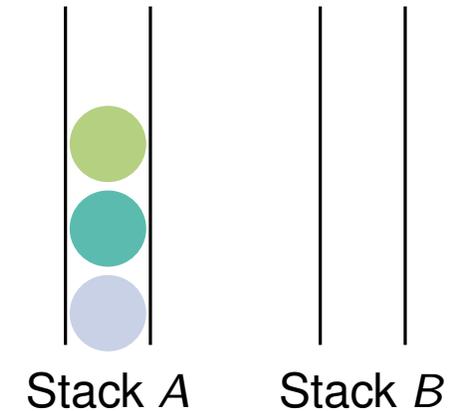
Analyse: Potentialmethode

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**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1$
 - $c_i = 2 \cdot |A_{i-1}| + 1$
 - $\Phi(D_i) - \Phi(D_{i-1}) = -2|A_{i-1}|$



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