Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität

Institut für Informatik

Algorithmen und Datenstrukturen

Wintersemester 2020/21
17. Vorlesung

Amortisierte Analyse

Prof. Dr. Alexander Wolff

Lehrstuhl für Informatik I

Einstiegsbeispiel: Hash-Tabellen

Frage: Wie groß macht man eine Hash-Tabelle?

Idee: Charles E. Leiserson
http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/LectureNotes/lec13.pdf

Einstiegsbeispiel: Hash-Tabellen

Frage: Wie groß macht man eine Hash-Tabelle?

Ziel: So groß wie nötig, so klein wie möglich...

Idee: Charles E. Leiserson
http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/LectureNotes/lec13.pdf

Einstiegsbeispiel: Hash-Tabellen

Frage: Wie groß macht man eine Hash-Tabelle?

Ziel: So groß wie nötig, so klein wie möglich...

Idee: Charles E. Leiserson
http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/LectureNotes/lec13.pdf

Einstiegsbeispiel: Hash-Tabellen

Frage: Wie groß macht man eine Hash-Tabelle?

Ziel: So groß wie nötig, so klein wie möglich...

Verhindere, dass die Tabelle überläuft
oder dass Operationen ineffizient werden.

Idee: Charles E. Leiserson
http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/LectureNotes/lec13.pdf

Einstiegsbeispiel: Hash-Tabellen

Frage: Wie groß macht man eine Hash-Tabelle?

Ziel: So groß wie nötig, so klein wie möglich...

Verhindere, dass die Tabelle überläuft
oder dass Operationen ineffizient werden.

Problem: Was tun, wenn man die maximale Anzahl zu speichernder Elemente vorab nicht kennt?

Idee: Charles E. Leiserson
http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/LectureNotes/lec13.pdf

Einstiegsbeispiel: Hash-Tabellen

Frage: Wie groß macht man eine Hash-Tabelle?

Ziel: So groß wie nötig, so klein wie möglich...

Verhindere, dass die Tabelle überläuft
oder dass Operationen ineffizient werden.

Problem: Was tun, wenn man die maximale Anzahl zu speichernder Elemente vorab nicht kennt?

Lösung:

Idee: Charles E. Leiserson
http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/LectureNotes/lec13.pdf

Einstiegsbeispiel: Hash-Tabellen

Frage: Wie groß macht man eine Hash-Tabelle?

Ziel: So groß wie nötig, so klein wie möglich...

Verhindere, dass die Tabelle überläuft
oder dass Operationen ineffizient werden.

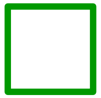
Problem: Was tun, wenn man die maximale Anzahl zu speichernder Elemente vorab nicht kennt?

Lösung: *Dynamische* Tabellen!

Idee: Charles E. Leiserson
http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/LectureNotes/lec13.pdf

Dynamische Tabellen

Idee:



Dynamische Tabellen

Idee:

Insert(1) $\square$

Dynamische Tabellen

Idee:

Insert(1) $\boxed{1}$

Dynamische Tabellen

Idee:

Insert(1) $\boxed{1}$

Insert(2)

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).

Insert(1)

Insert(2)

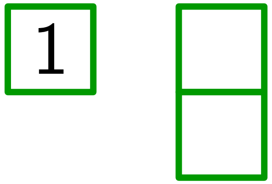
$\boxed{1}$

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).

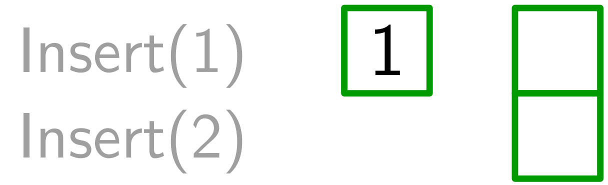
Insert(1)
Insert(2)



Dynamische Tabellen

Idee:

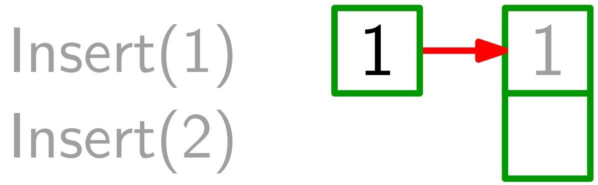
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.



Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.

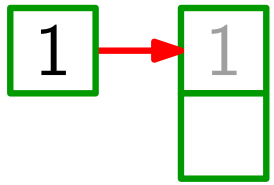


Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

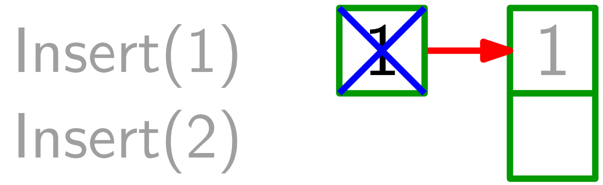
Insert(1)
Insert(2)



Dynamische Tabellen

Idee:

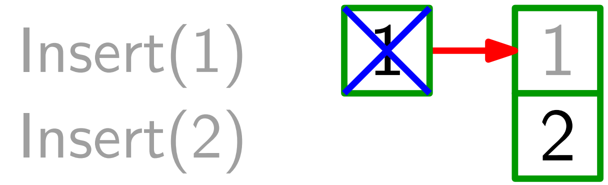
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

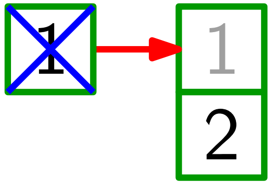


Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

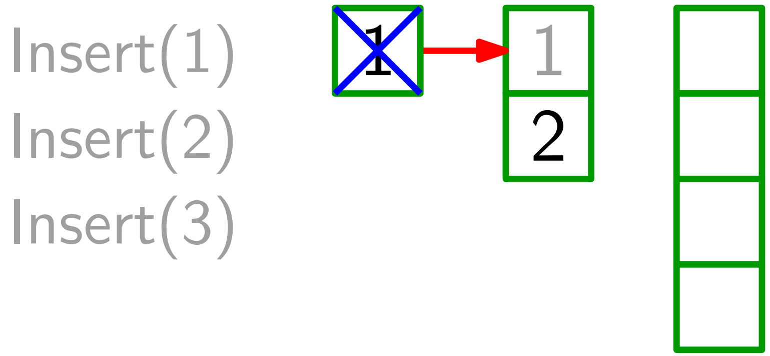
Insert(1)
Insert(2)
Insert(3)



Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

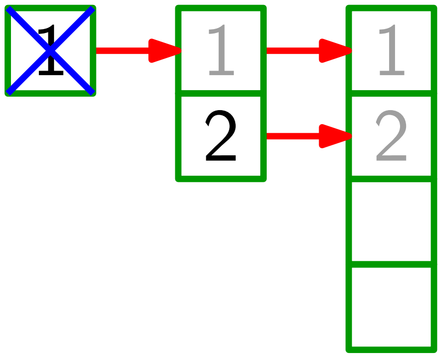


Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)

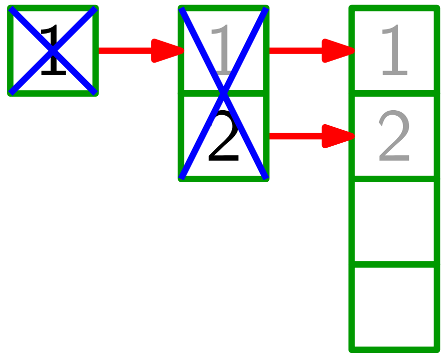


Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

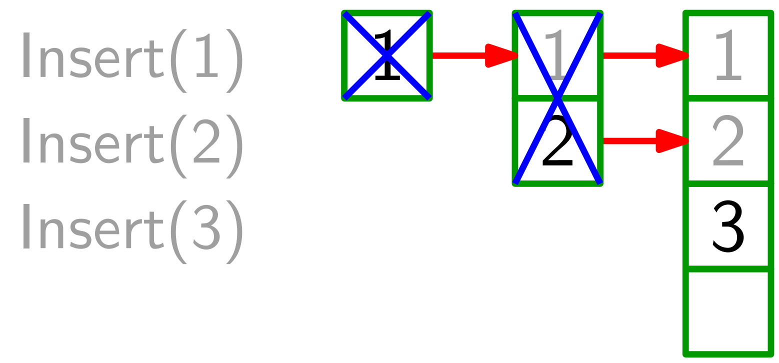
Insert(1)
Insert(2)
Insert(3)



Dynamische Tabellen

Idee:

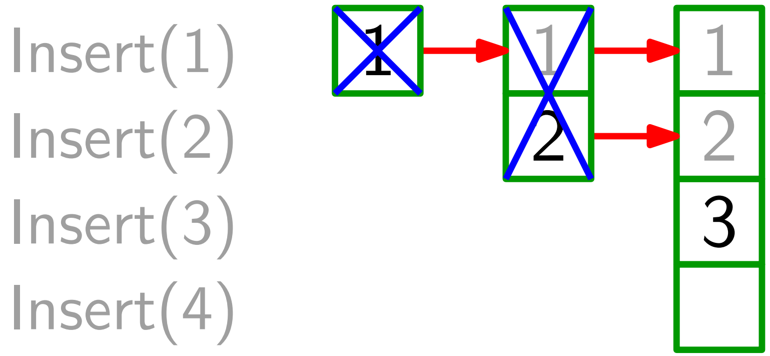
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Dynamische Tabellen

Idee:

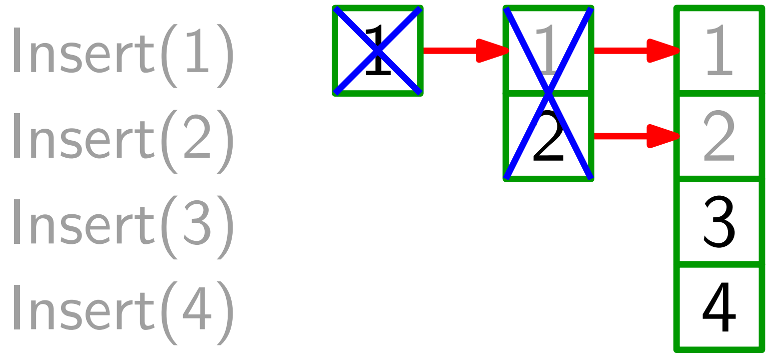
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

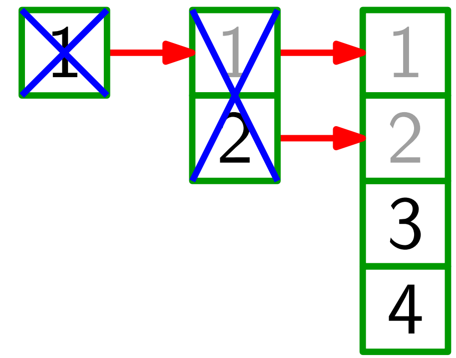


Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)

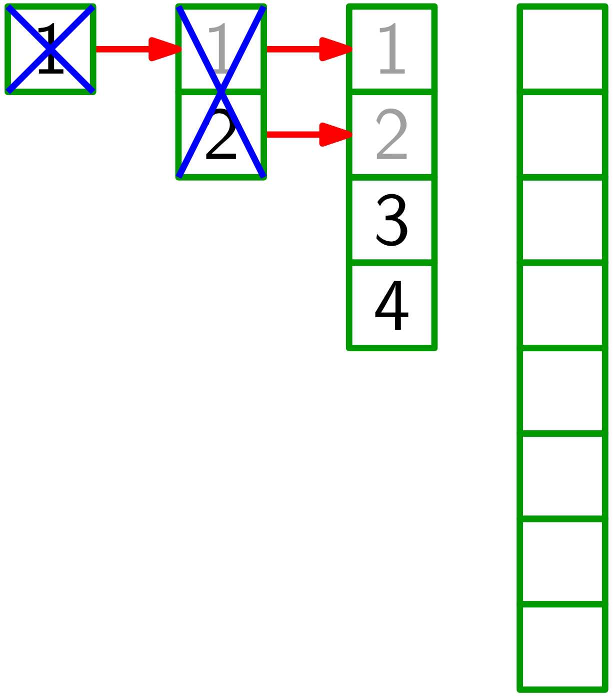


Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

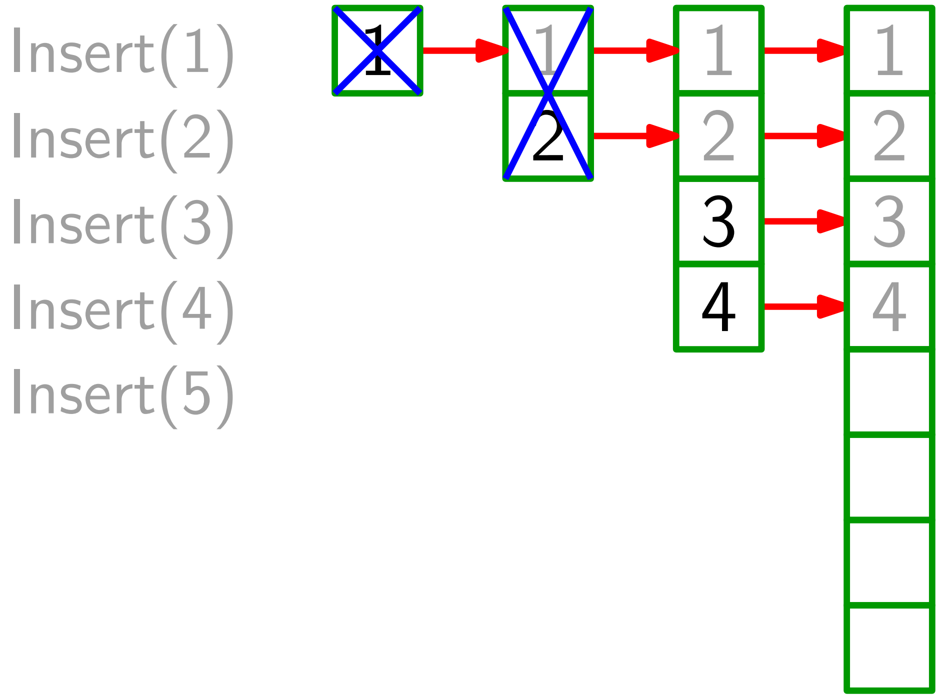
Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)



Dynamische Tabellen

Idee:

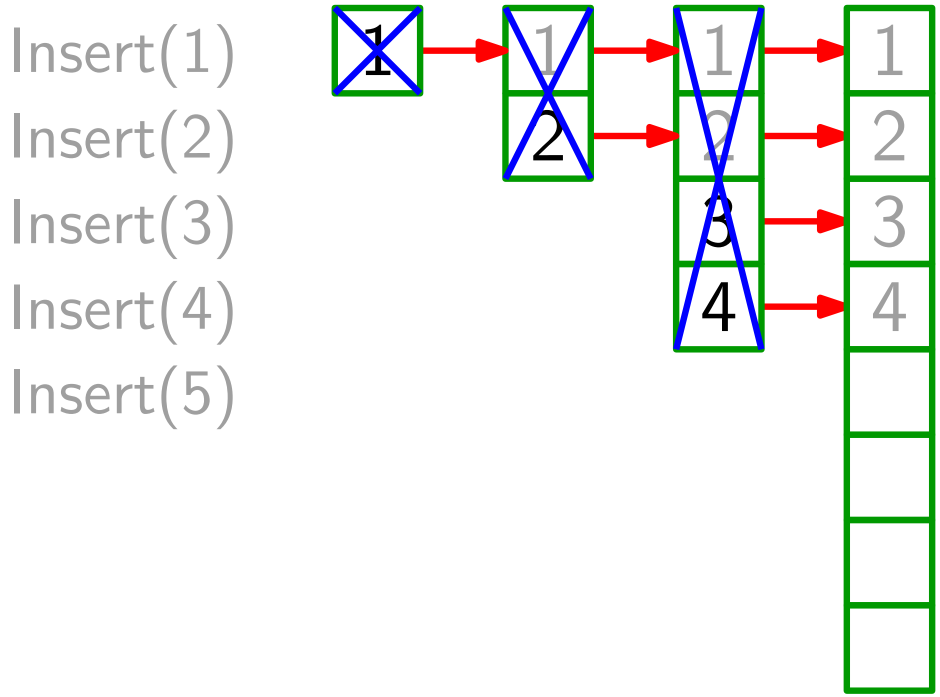
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Dynamische Tabellen

Idee:

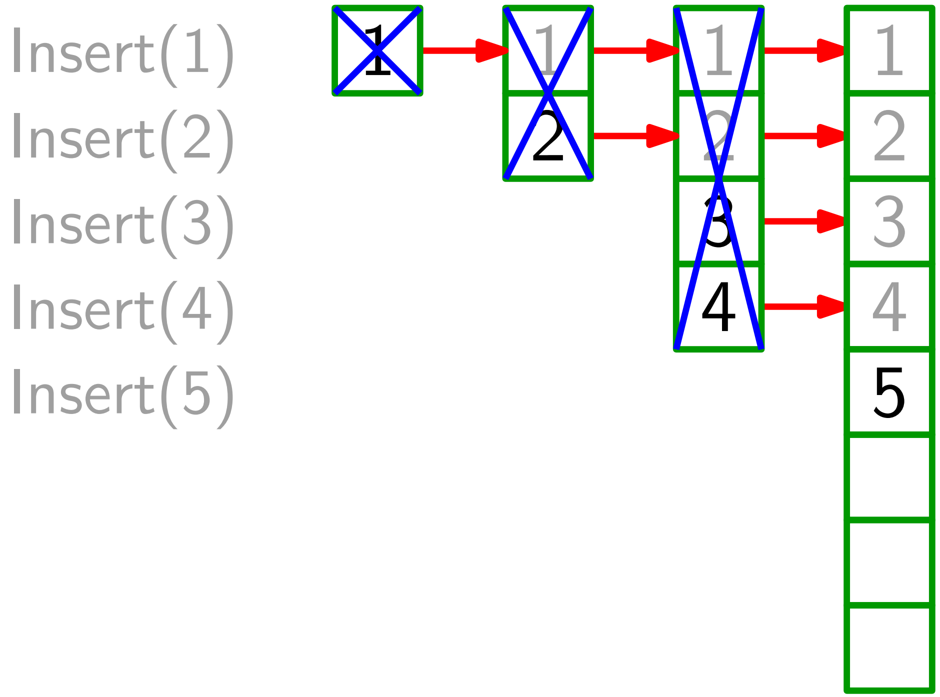
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Dynamische Tabellen

Idee:

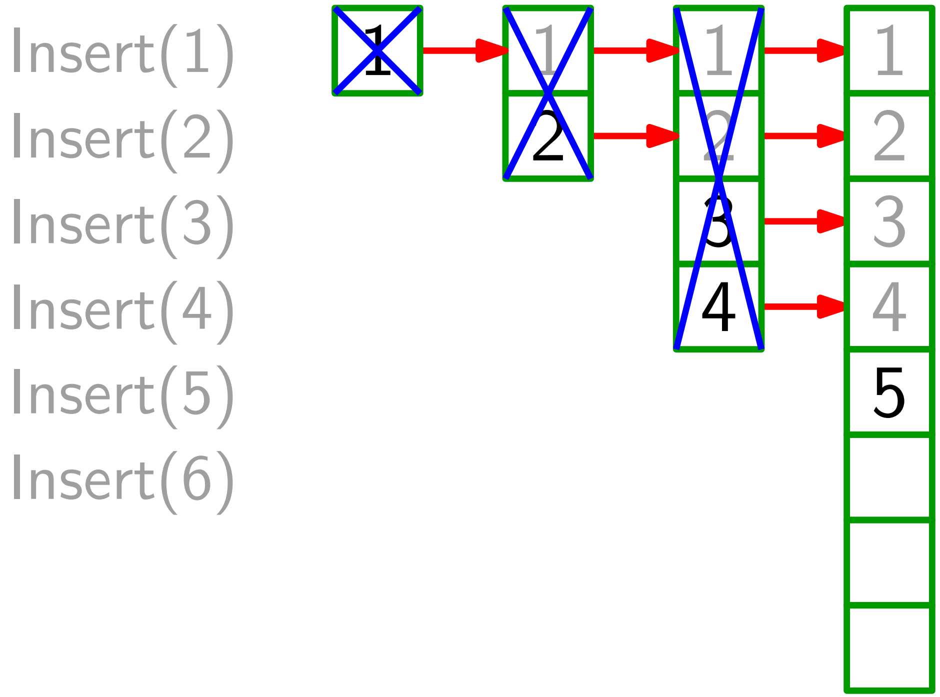
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Dynamische Tabellen

Idee:

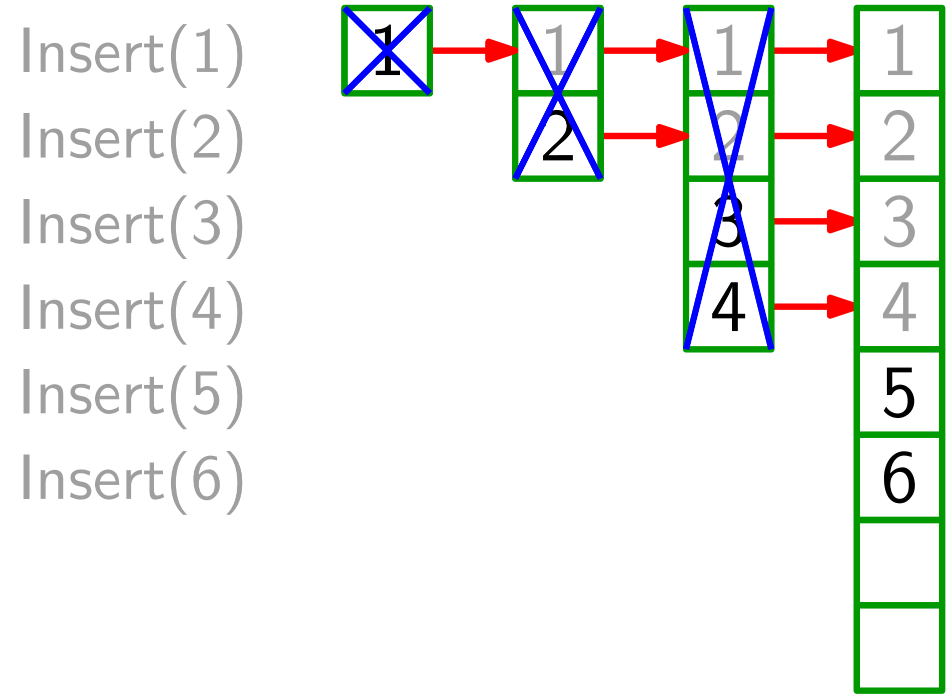
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Dynamische Tabellen

Idee:

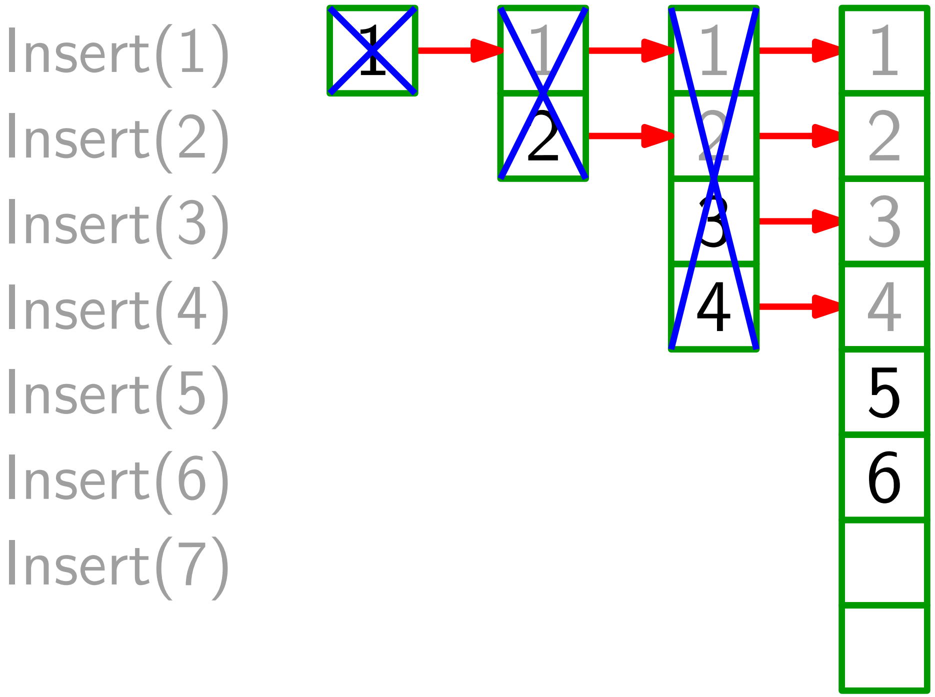
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Dynamische Tabellen

Idee:

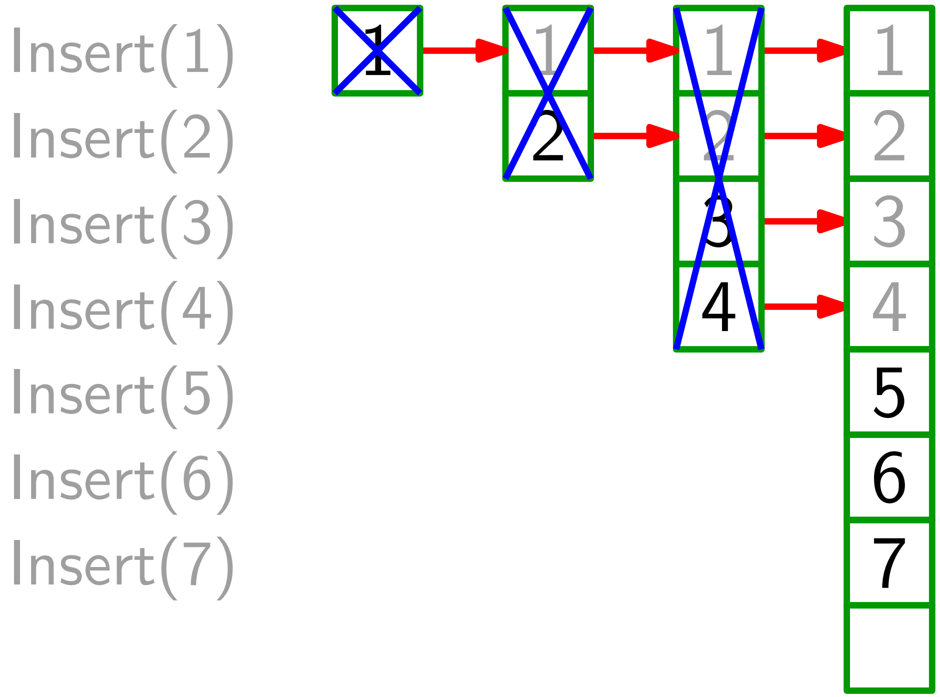
- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

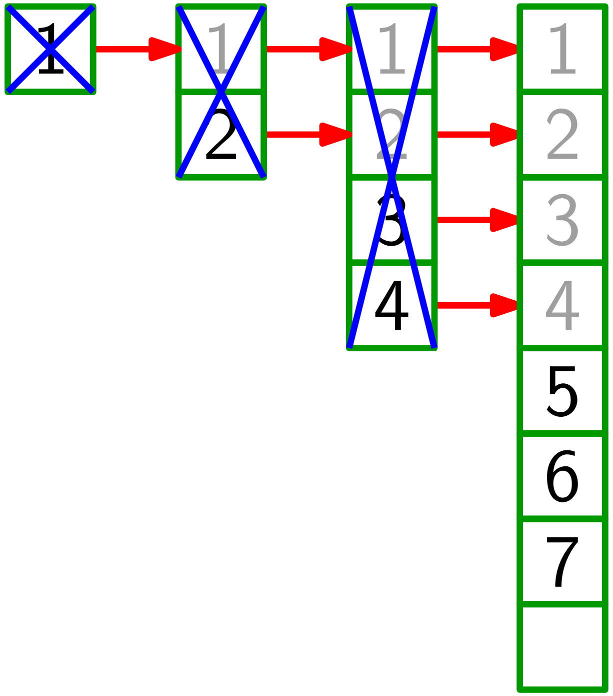


Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

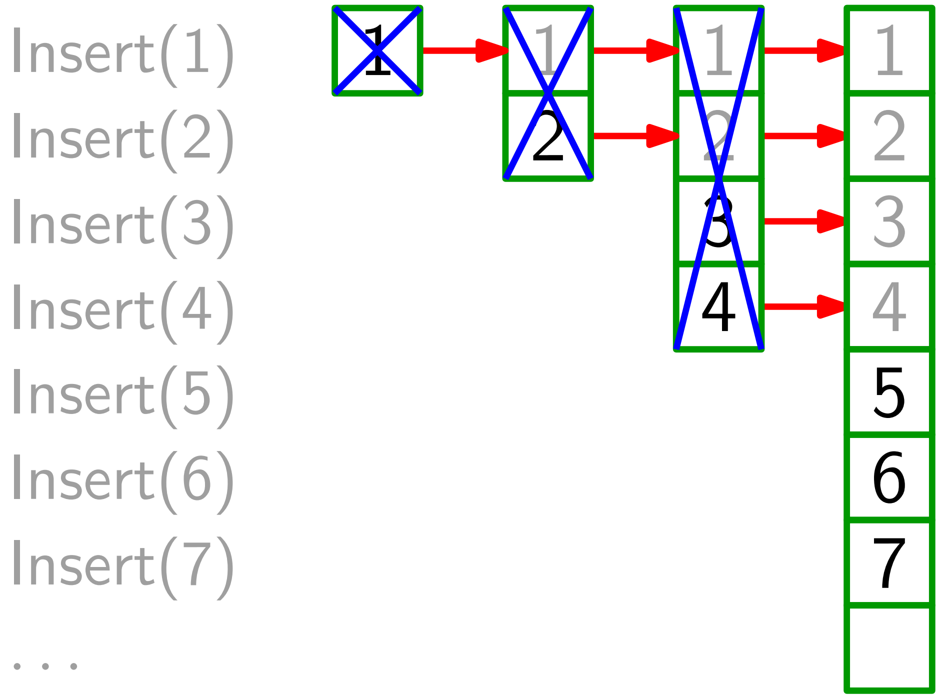
Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
...



Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



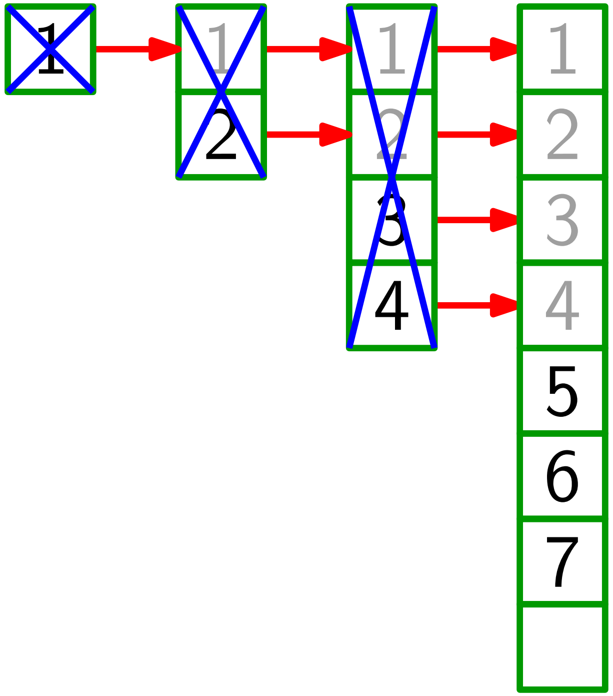
Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
...



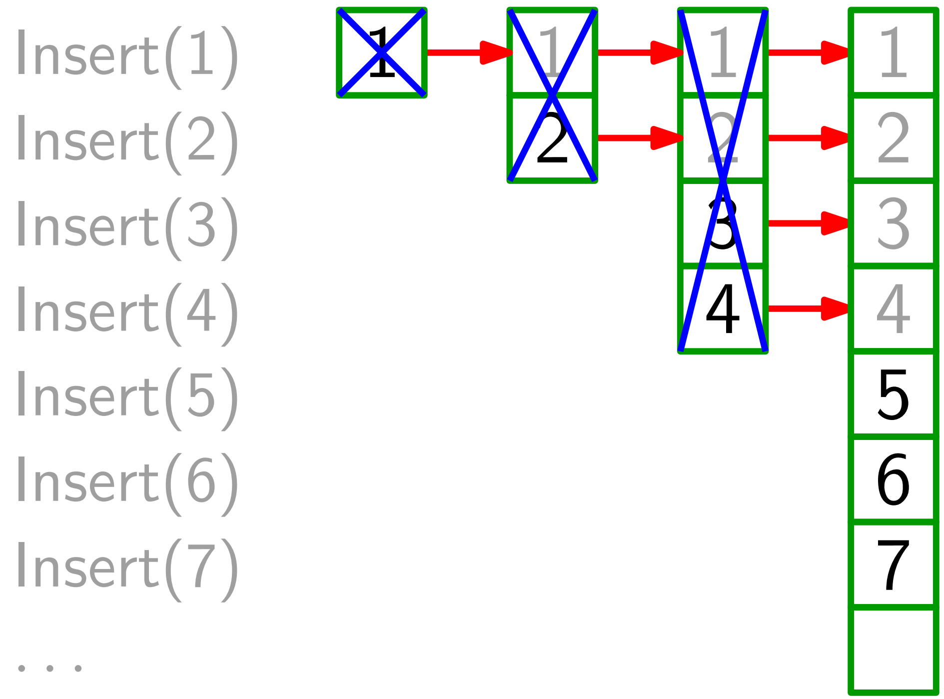
Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Antwort:

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.



Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

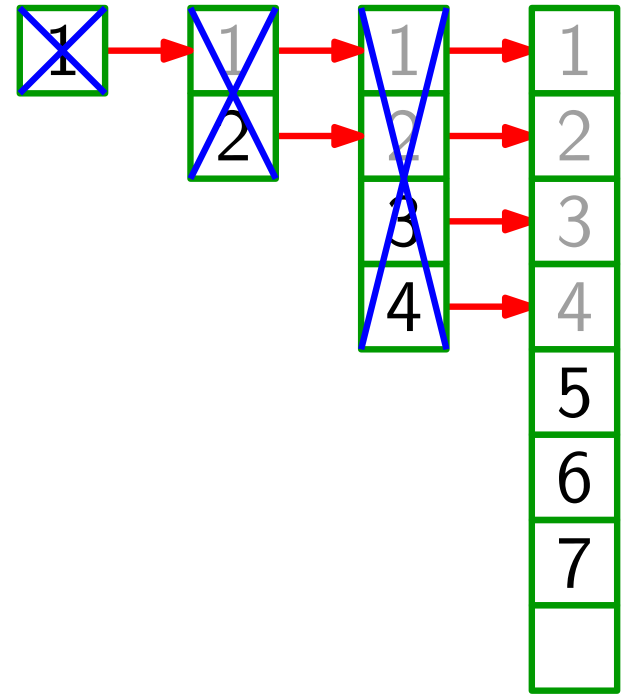
Antwort:
- Tabelle wird genau $(\lceil \log_2 n \rceil)$-mal kopiert.

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
...



Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Antwort:

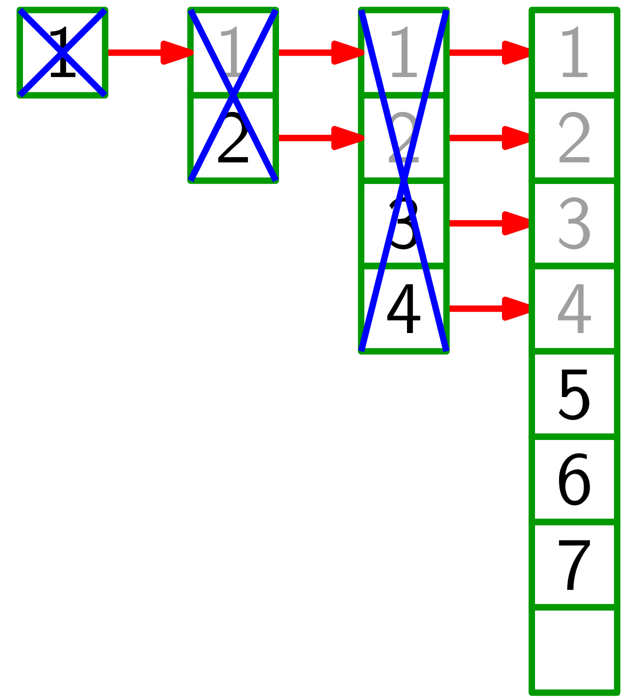
- Tabelle wird genau $(\lceil \log_2 n \rceil)$-mal kopiert.
- Im schlimmsten (letzten!) Fall ist der Aufwand $\Theta(n)$.

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
...



Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Antwort:
- Tabelle wird genau $(\lceil \log_2 n \rceil)$-mal kopiert.
- Im schlimmsten (letzten!) Fall ist der Aufwand $\Theta(n)$.

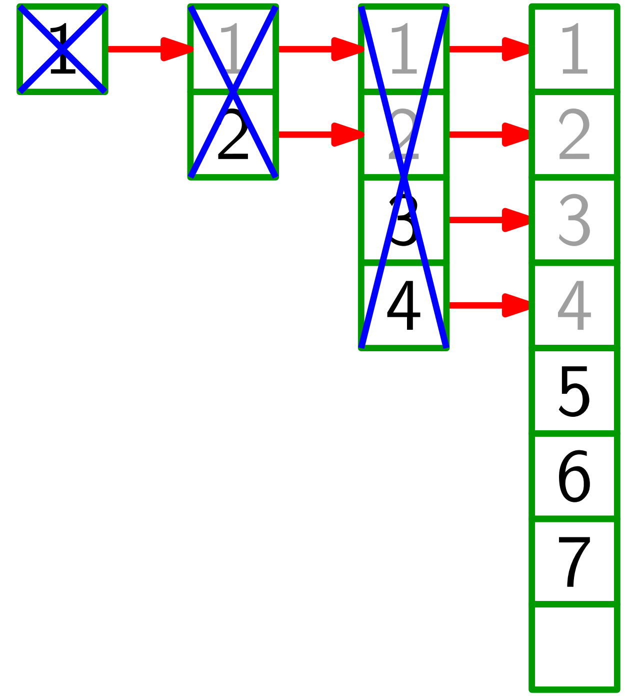
Also ist der Gesamtaufwand $\Theta(n \log n)$.

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
. . .



Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Antwort:
- Tabelle wird genau $(\lceil \log_2 n \rceil)$-mal kopiert.
- Im schlimmsten (letzten!) Fall ist der Aufwand $\Theta(n)$.

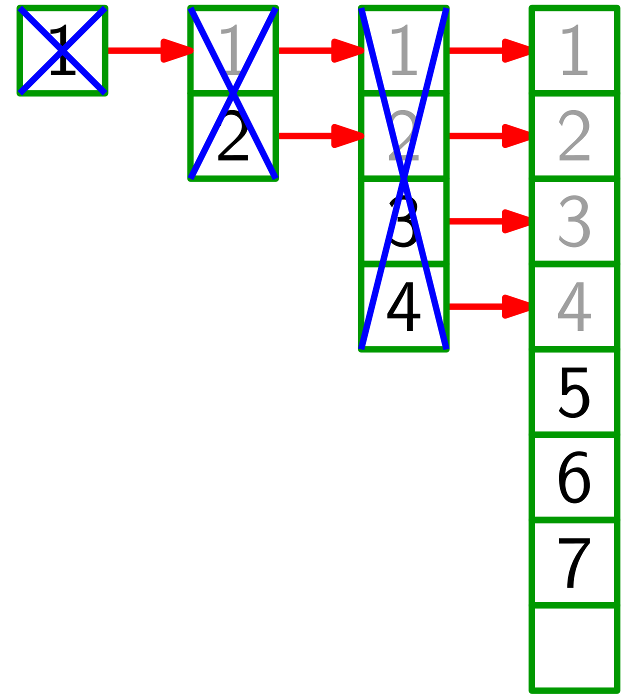
Also ist der Gesamtaufwand $\Theta(n \log n)$.

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
...



Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Antwort:

- Tabelle wird genau $(\lceil \log_2 n \rceil)$-mal kopiert.
- Im schlimmsten (letzten!) Fall ist der Aufwand $\Theta(n)$.

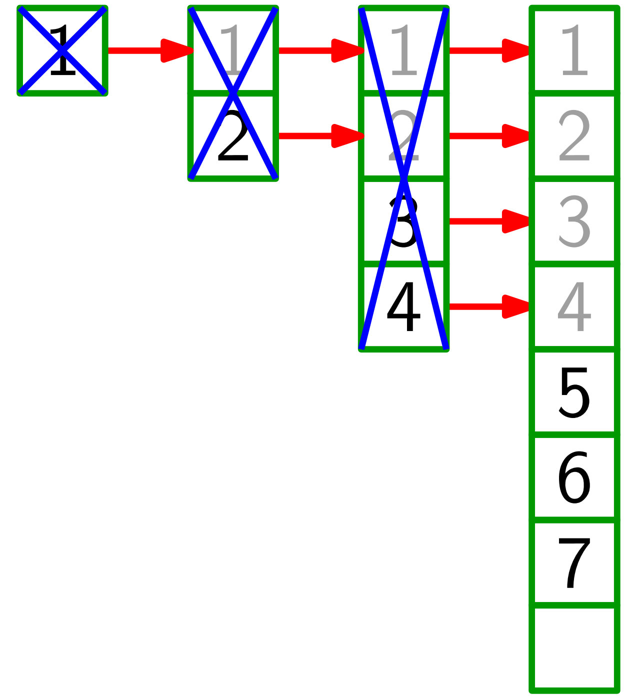
Also ist der Gesamtaufwand $\underset{O}{\cancel{\Theta}}(n \log n)$.

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
...



Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Antwort:

- Tabelle wird genau $(\lceil \log_2 n \rceil)$-mal kopiert.
- Im schlimmsten (letzten!) Fall ist der Aufwand $\Theta(n)$.

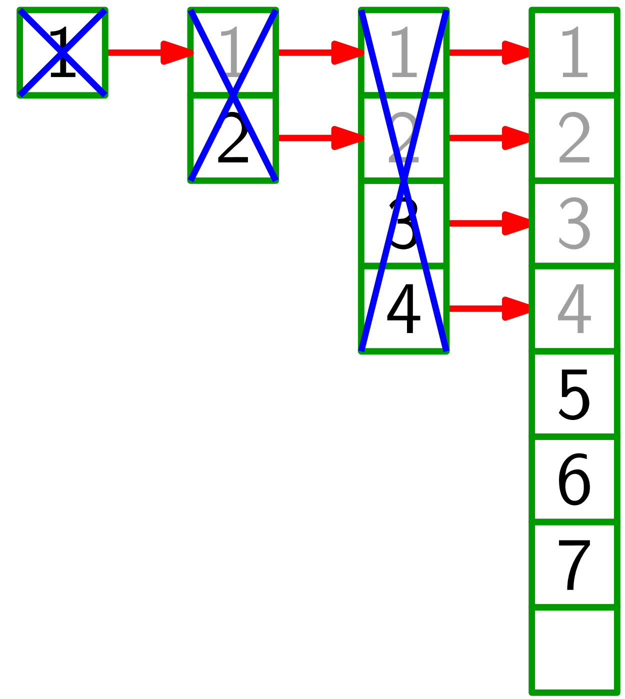
Also ist der Gesamtaufwand $\underset{O}{\cancel{\Theta}}(n \log n)$, genauer

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
...



Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Antwort:
- Tabelle wird genau $(\lceil \log_2 n \rceil)$-mal kopiert.
- Im schlimmsten (letzten!) Fall ist der Aufwand $\Theta(n)$.

Lüge!

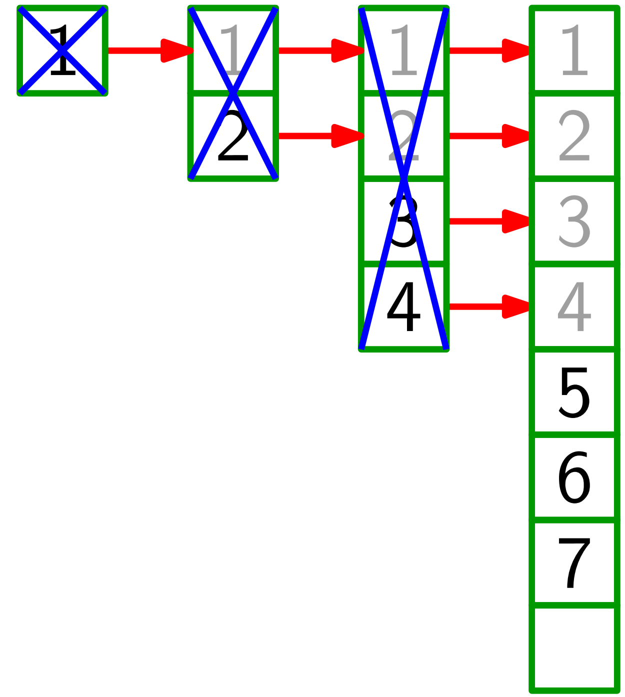
Also ist der Gesamtaufwand $\cancel{\Theta}\,O(n \log n)$, genauer $\Theta(n)$.

Dynamische Tabellen

Idee:

- Wenn Tabelle voll, fordere doppelt so große Tabelle an (mit `new`).
- Kopiere alle Einträge von alter in neue Tabelle.
- Gib Speicher für alte Tabelle frei.

Insert(1)
Insert(2)
Insert(3)
Insert(4)
Insert(5)
Insert(6)
Insert(7)
...



Analyse: Welche Laufzeit benötigen n Einfügeoperationen im schlimmsten Fall?

Antwort:
- Tabelle wird genau $(\lceil \log_2 n \rceil)$-mal kopiert.
- Im schlimmsten (letzten!) Fall ist der Aufwand $\Theta(n)$.

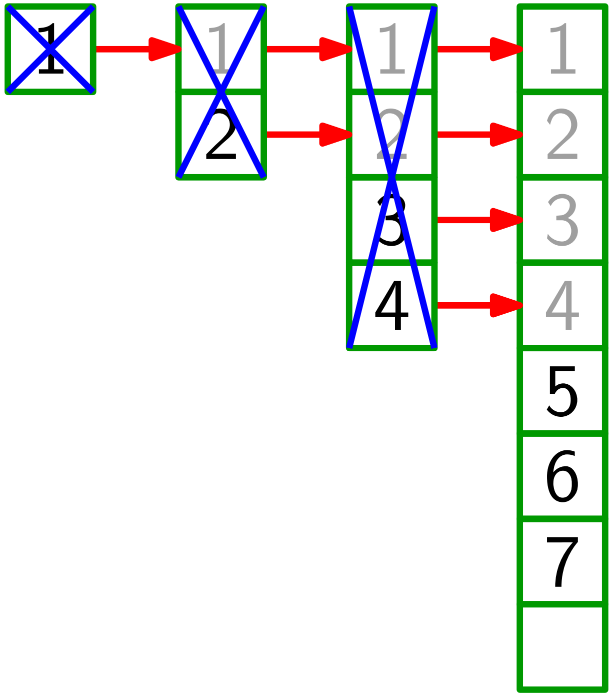
Lüge!

Also ist der Gesamtaufwand $\underset{O}{\cancel{\Theta}}(n \log n)$, genauer $\Theta(n)$.

Let's see why...

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei
$c_i =$ Kosten fürs i-te Einfügen.

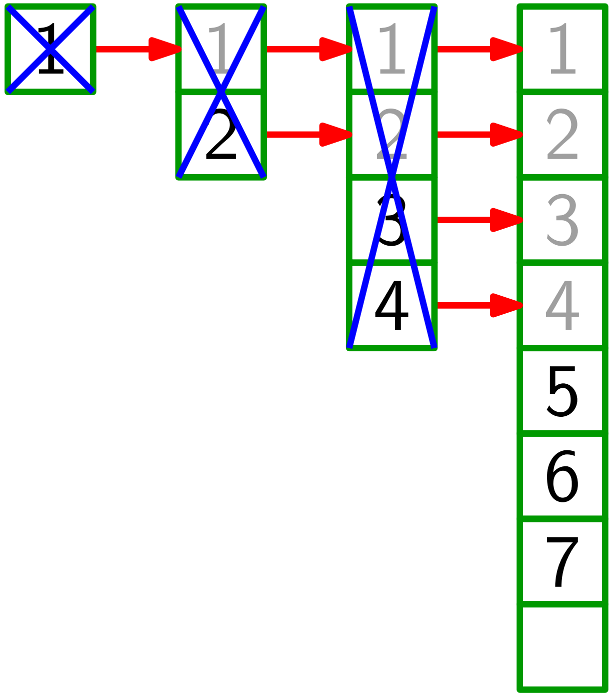


Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei

$c_i =$ Kosten fürs i-te Einfügen.

i	1	2	3	4	5	6	7	8	9
$size_i$	1	2	4	4	8	8	8	8	16

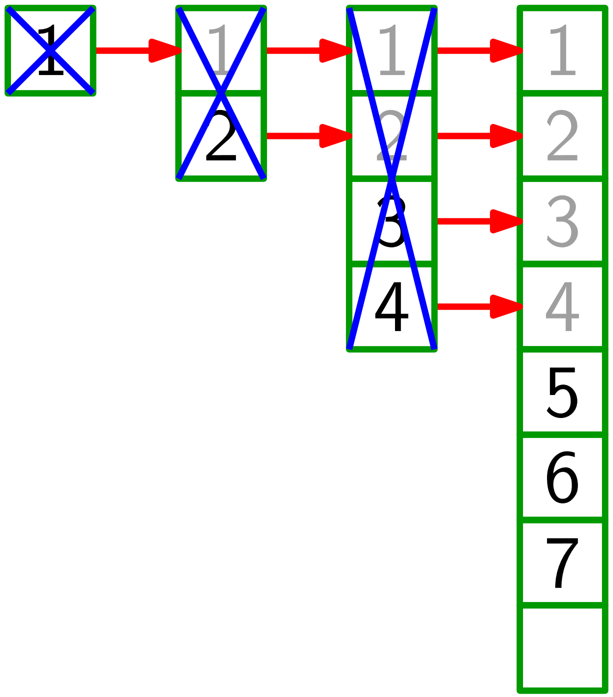


Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei

$c_i =$ Kosten fürs i-te Einfügen.

i	1	2	3	4	5	6	7	8	9
$size_i$	1	2	4	4	8	8	8	8	16
c_i	1	1	1	1	1	1	1	1	1
		1	2		4				8

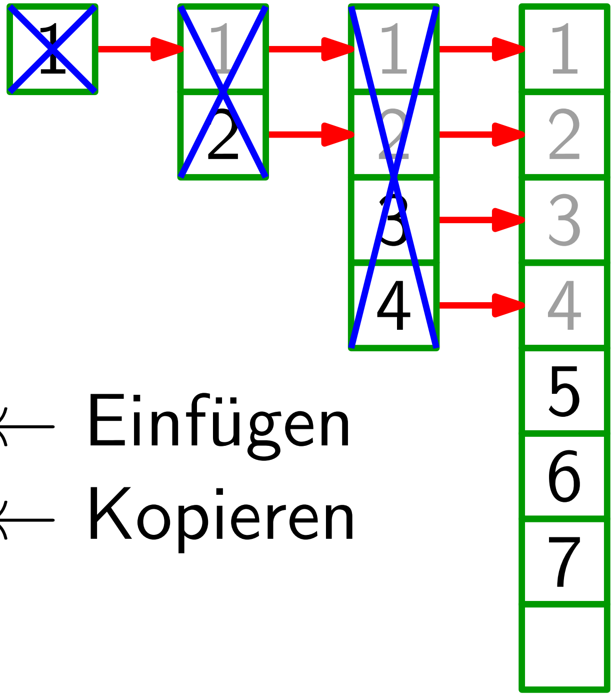


Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei

$c_i =$ Kosten fürs i-te Einfügen.

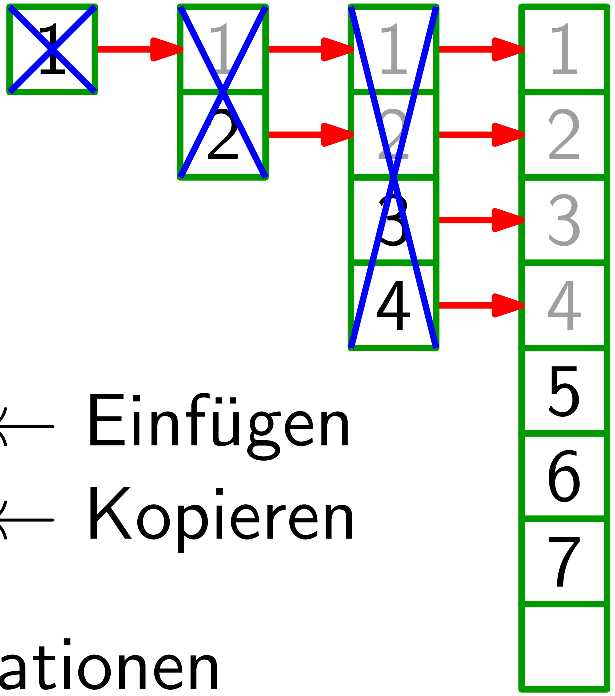
i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	$\leftarrow$ Einfügen
		1	2		4				8	$\leftarrow$ Kopieren



Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei

c_i = Kosten fürs i-te Einfügen.



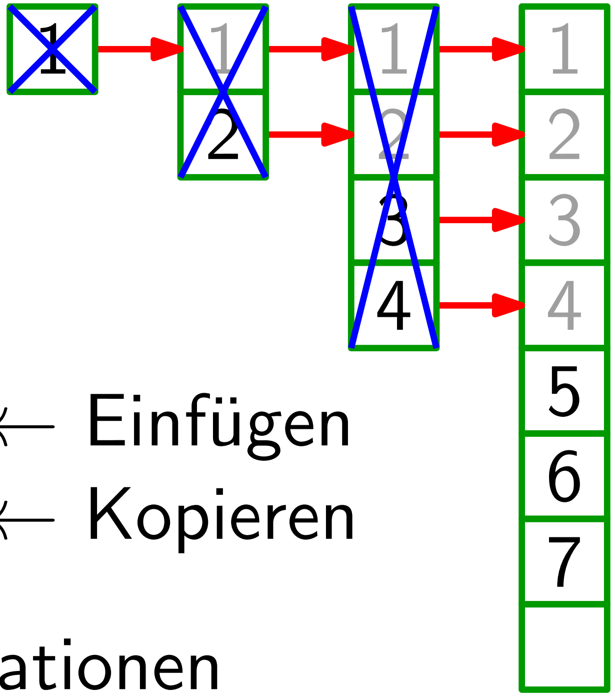
i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	$\leftarrow$ Einfügen
		1	2		4				8	$\leftarrow$ Kopieren

Also betragen die Kosten für n Einfügeoperationen

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei

$c_i =$ Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	$\leftarrow$ Einfügen
		1	2		4				8	$\leftarrow$ Kopieren

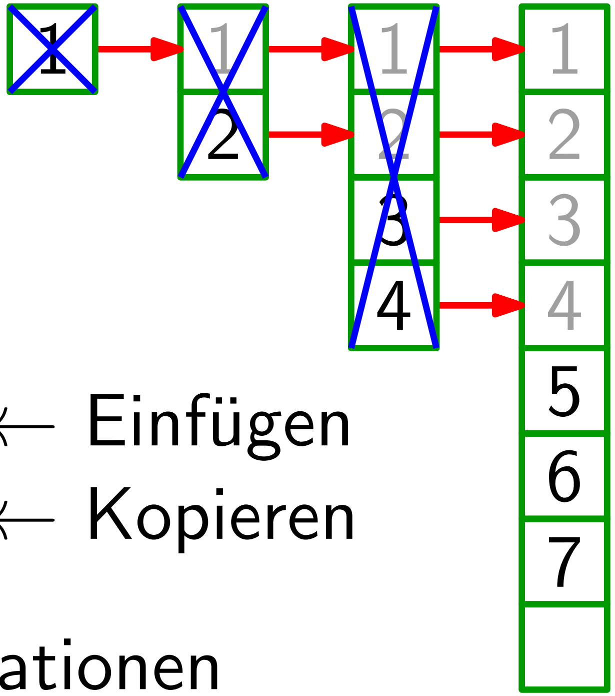
Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} =$$

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei

c_i = Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	$\leftarrow$ Einfügen
		1	2		4				8	$\leftarrow$ Kopieren

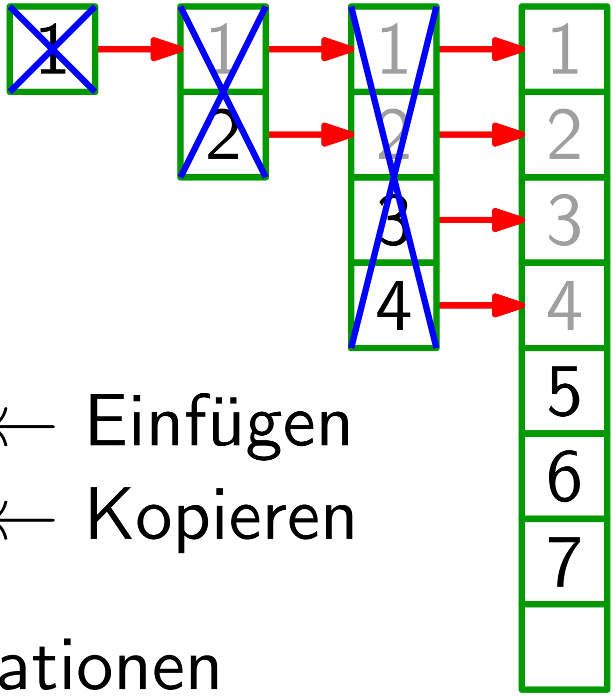
Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = \blacksquare + \sum_{j=0}$$

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei
c_i = Kosten fürs i-te Einfügen.

i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren



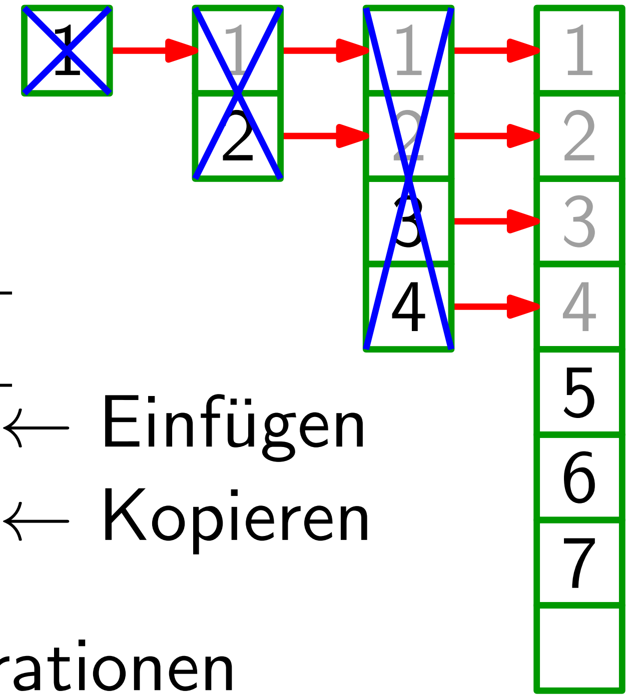
Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = n + \sum_{j=0}$$

Genauere Abschätzung: $\boxed{\text{Aggregationsmethode}}$

Für $i = 1, \ldots, n$ sei

c_i = Kosten fürs i-te Einfügen.



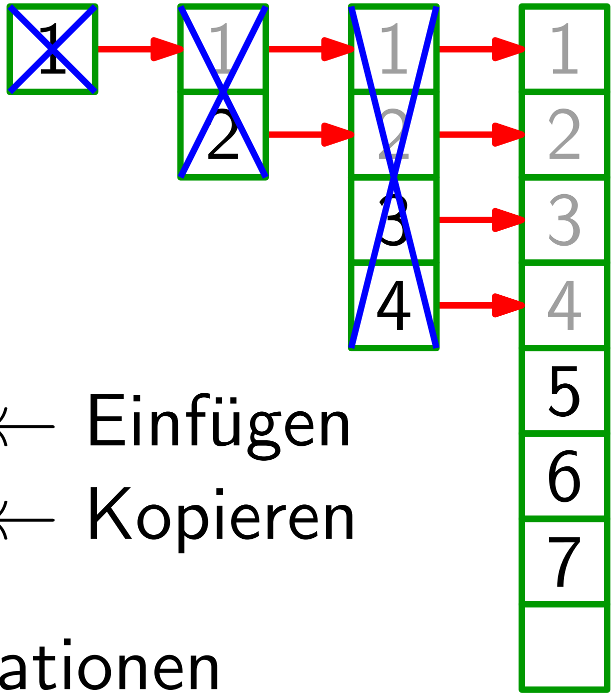
i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = n + \sum_{j=0} 2^j$$

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei
c_i = Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

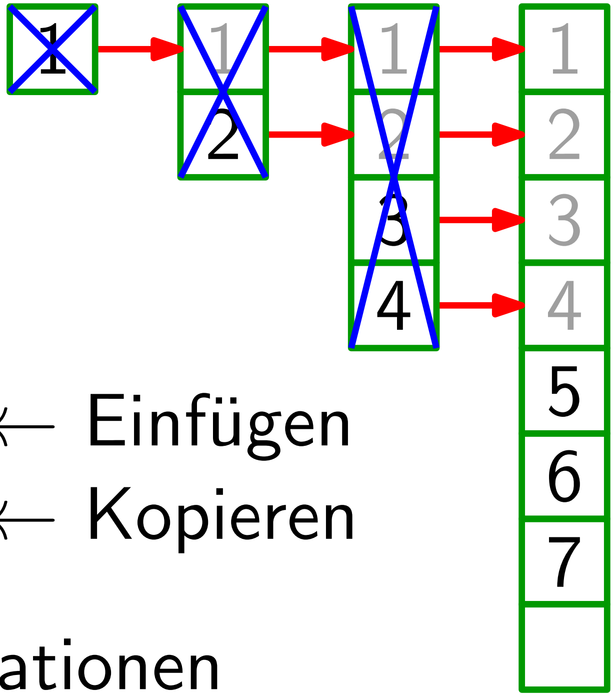
Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = \boxed{n} + \sum_{j=0}^{\lfloor \log_2(n-1) \rfloor} 2^j$$

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei

c_i = Kosten fürs i-te Einfügen.



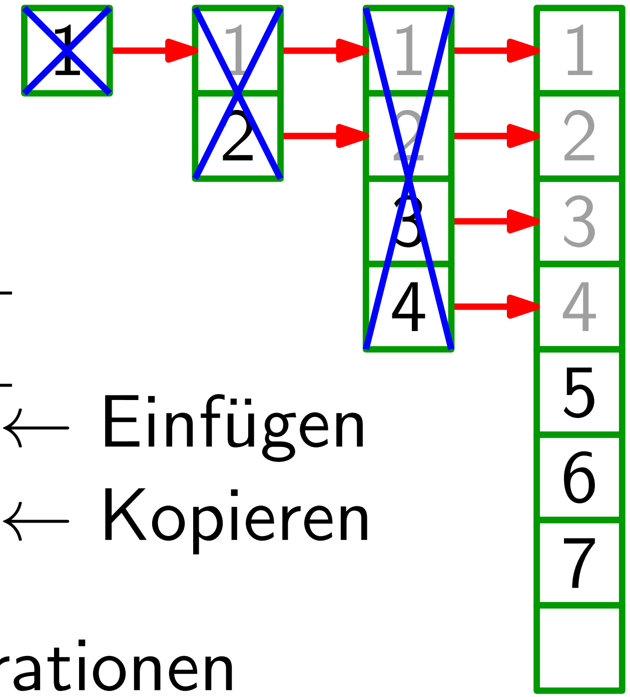
i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = n + \sum_{j=0}^{\lfloor \log_2(n-1) \rfloor} 2^j \leq$$

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei
c_i = Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

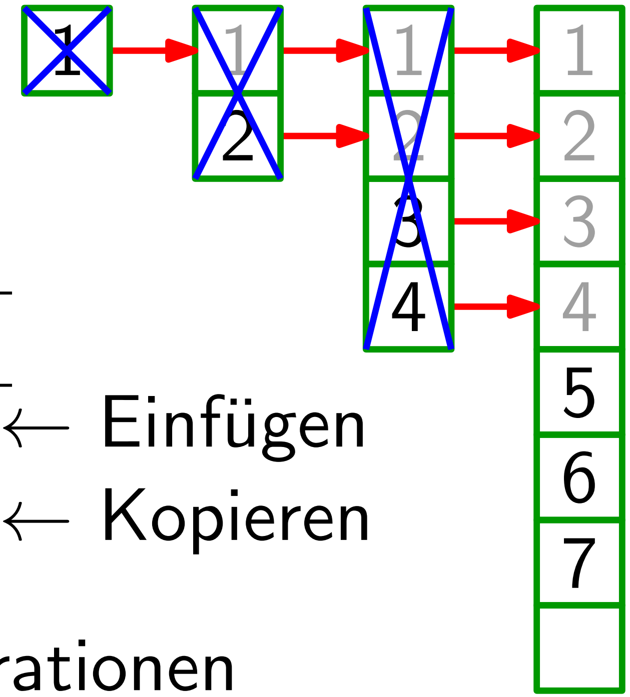
Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = n + \sum_{j=0}^{\lfloor \log_2(n-1) \rfloor} 2^j \le$$

$$\sum_{j=0}^{k} q^j = \begin{bmatrix} \text{endl.} \\ \text{geom.} \\ \text{Reihe} \end{bmatrix}$$

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei
    c_i = Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

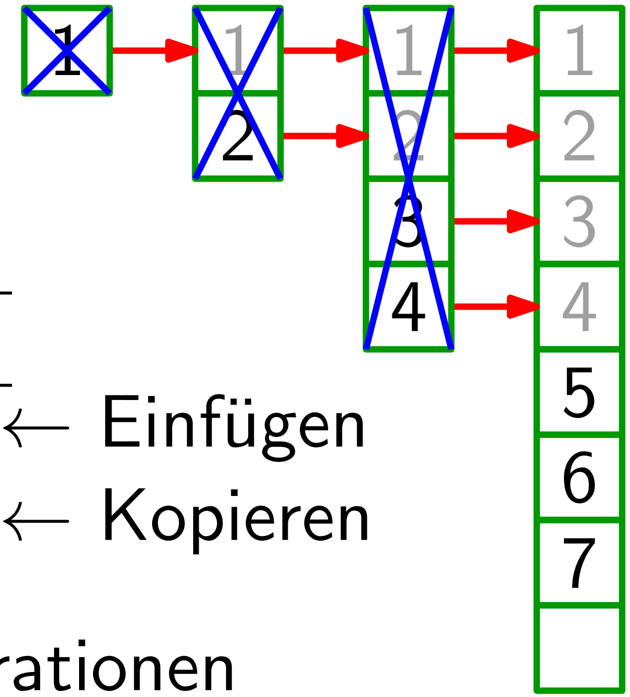
Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = \boxed{n} + \sum_{j=0}^{\lfloor \log_2 (n-1) \rfloor} 2^j \leq$$

$$\sum_{j=0}^{k} q^j = \frac{q^{k+1} - 1}{q - 1} \quad \begin{bmatrix}\text{endl.} \\ \text{geom.} \\ \text{Reihe}\end{bmatrix}$$

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei
c_i = Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

Also betragen die Kosten für n Einfügeoperationen

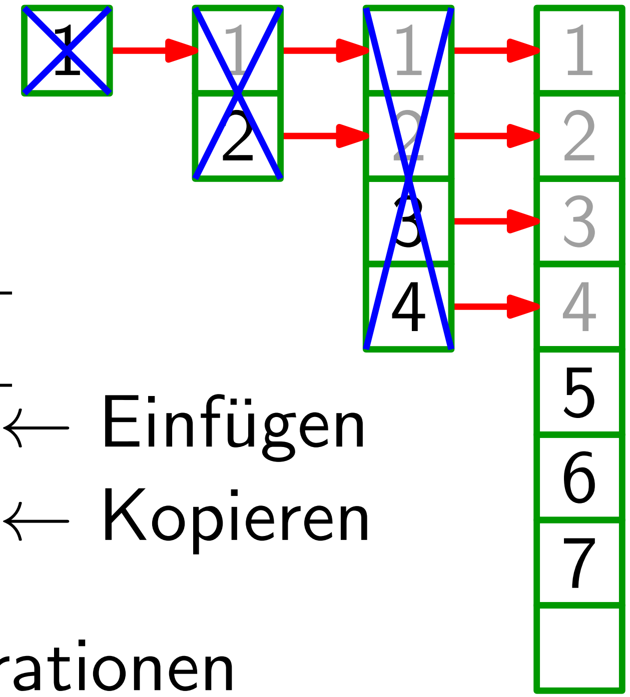
$$\boxed{\sum_{1}^{n} c_i} = n + \sum_{j=0}^{\lfloor \log_2(n-1) \rfloor} 2^j \le n + \frac{2^{\log_2(n-1)+1} - 1}{2 - 1}$$

$$\sum_{j=0}^{k} q^j = \frac{q^{k+1} - 1}{q - 1} \quad \begin{bmatrix}\text{endl.}\\ \text{geom.}\\ \text{Reihe}\end{bmatrix}$$

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei

$c_i =$ Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = n + \sum_{j=0}^{\lfloor \log_2(n-1) \rfloor} 2^j \le n + \frac{2^{\log_2(n-1)+1} - 1}{2-1}$$

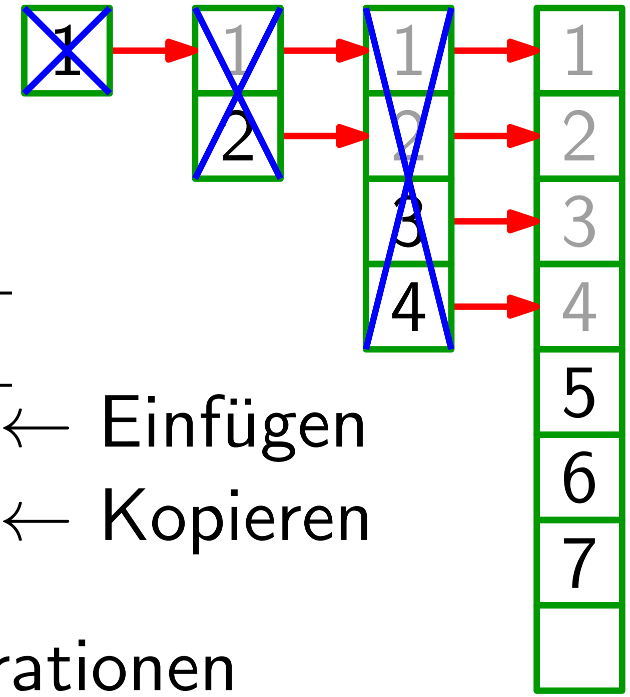
$$= n + 2(n-1) - 1$$

$$\sum_{j=0}^{k} q^j = \frac{q^{k+1} - 1}{q - 1}$$ [endl. geom. Reihe]

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei

c_i = Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_1^n c_i} = n + \sum_{j=0}^{\lfloor \log_2 (n-1) \rfloor} 2^j \leq n + \frac{2^{\log_2(n-1)+1} - 1}{2-1}$$

$$= n + 2(n-1) - 1$$

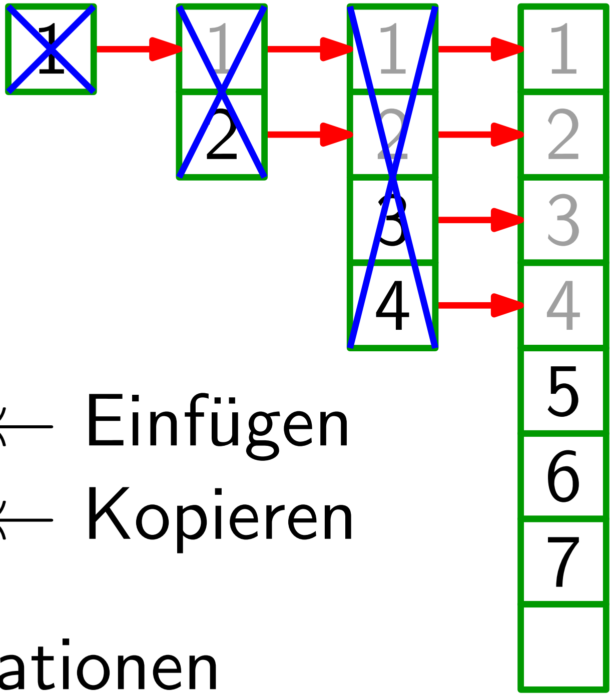
$$< 3n$$

$$\sum_{j=0}^{k} q^j = \frac{q^{k+1}-1}{q-1} \quad \begin{bmatrix}\text{endl.} \\ \text{geom.} \\ \text{Reihe}\end{bmatrix}$$

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \ldots, n$ sei
c_i = Kosten fürs i-te Einfügen.

i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren



Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = n + \sum_{j=0}^{\lfloor \log_2(n-1) \rfloor} 2^j \leq n + \frac{2^{\log_2(n-1)+1} - 1}{2-1}$$

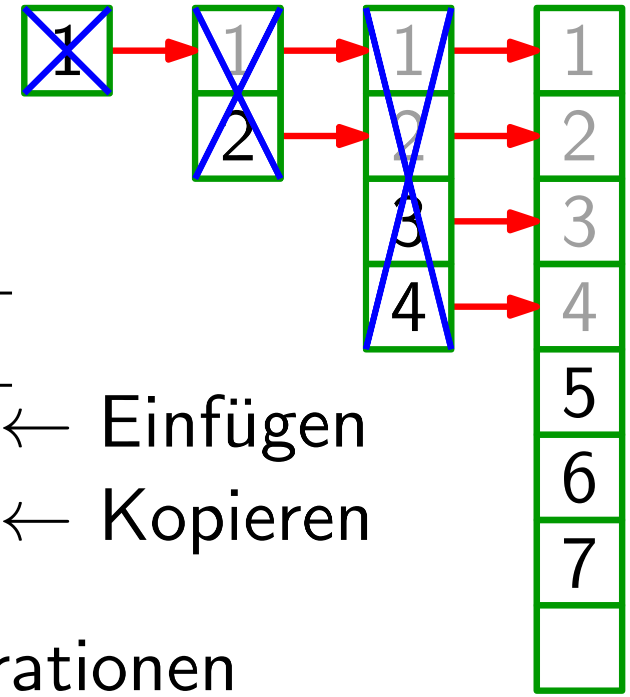
$$= n + 2(n-1) - 1$$

$$< 3n \in \Theta(n)$$

$$\sum_{j=0}^{k} q^j = \frac{q^{k+1}-1}{q-1}$$ [endl. geom. Reihe]

Genauere Abschätzung: Aggregationsmethode

Für $i = 1, \dots, n$ sei
c_i = Kosten fürs i-te Einfügen.



i	1	2	3	4	5	6	7	8	9	
$size_i$	1	2	4	4	8	8	8	8	16	
c_i	1	1	1	1	1	1	1	1	1	← Einfügen
		1	2		4				8	← Kopieren

Also betragen die Kosten für n Einfügeoperationen

$$\boxed{\sum_{1}^{n} c_i} = n + \sum_{j=0}^{\lfloor \log_2(n-1) \rfloor} 2^j \leq n + \frac{2^{\log_2(n-1)+1} - 1}{2 - 1}$$

$$= n + 2(n-1) - 1$$

$$< 3n \in \Theta(n)$$

$$\sum_{j=0}^{k} q^j = \frac{q^{k+1} - 1}{q - 1} \quad \left[\text{endl. geom. Reihe}\right]$$

D.h. die durchschnittlichen („amortisierten“) Kosten sind $\Theta(1)$.

Amortisierte Analyse...

...bedeutet zu zeigen, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben

Amortisierte Analyse...

...bedeutet zu zeigen, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben –
auch wenn einzelne Operationen in der Folge teuer sind!

Amortisierte Analyse...

...bedeutet zu zeigen, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben –
auch wenn einzelne Operationen in der Folge teuer sind!

Auch *randomisierte Analyse* kann man als Durchschnittsbildung (über alle mögl. Ergebnisse, gewichtet nach Wahrscheinlichkeit) betrachten.

Amortisierte Analyse...

...bedeutet zu zeigen, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben –
auch wenn einzelne Operationen in der Folge teuer sind!

Auch *randomisierte Analyse* kann man als Durchschnittsbildung (über alle mögl. Ergebnisse, gewichtet nach Wahrscheinlichkeit) betrachten.

Bei amortisierter Analyse geht es jedoch um die durchschnittliche Laufzeit *im schlechtesten Fall* –
nicht im Erwartungswert!

Amortisierte Analyse...

...bedeutet zu zeigen, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben – *auch wenn einzelne Operationen in der Folge teuer sind!*

Auch *randomisierte Analyse* kann man als Durchschnittsbildung (über alle mögl. Ergebnisse, gewichtet nach Wahrscheinlichkeit) betrachten.

Bei amortisierter Analyse geht es jedoch um die durchschnittliche Laufzeit *im schlechtesten Fall* – nicht im Erwartungswert!

Wir betrachten 3 verschiedene Typen von amortisierter Analyse:

Amortisierte Analyse...

...bedeutet zu zeigen, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben –
auch wenn einzelne Operationen in der Folge teuer sind!

Auch *randomisierte Analyse* kann man als Durchschnittsbildung (über alle mögl. Ergebnisse, gewichtet nach Wahrscheinlichkeit) betrachten.

Bei amortisierter Analyse geht es jedoch um die durchschnittliche Laufzeit *im schlechtesten Fall* –
nicht im Erwartungswert!

Wir betrachten 3 verschiedene Typen von amortisierter Analyse:

- Aggregationsmethode
- Buchhaltermethode
- Potentialmethode

Amortisierte Analyse...

...bedeutet zu zeigen, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben –
auch wenn einzelne Operationen in der Folge teuer sind!

Auch *randomisierte Analyse* kann man als Durchschnittsbildung (über alle mögl. Ergebnisse, gewichtet nach Wahrscheinlichkeit) betrachten.

Bei amortisierter Analyse geht es jedoch um die durchschnittliche Laufzeit *im schlechtesten Fall* –
nicht im Erwartungswert!

Wir betrachten 3 verschiedene Typen von amortisierter Analyse:
- Aggregationsmethode ✓
- Buchhaltermethode
- Potentialmethode

Amortisierte Analyse...

...bedeutet zu zeigen, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben – *auch wenn einzelne Operationen in der Folge teuer sind!*

Auch *randomisierte Analyse* kann man als Durchschnittsbildung (über alle mögl. Ergebnisse, gewichtet nach Wahrscheinlichkeit) betrachten.

Bei amortisierter Analyse geht es jedoch um die durchschnittliche Laufzeit *im schlechtesten Fall* – nicht im Erwartungswert!

Wir betrachten 3 verschiedene Typen von amortisierter Analyse:

– Aggregationsmethode ✓
– Buchhaltermethode
– Potentialmethode

Buchhaltermethode

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$,

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

  $\hat{c}_i > c_i \ \Rightarrow$

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

  $\hat{c}_i > c_i \;\Rightarrow\;$ Wir legen etwas beiseite.

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

  $\hat{c}_i > c_i \;\Rightarrow$ Wir legen etwas beiseite.

  $c_i > \hat{c}_i \;\Rightarrow$

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

  $\hat{c}_i > c_i \;\Rightarrow\;$ Wir legen etwas beiseite.

  $c_i > \hat{c}_i \;\Rightarrow\;$ Wir bezahlen teure Operationen mit vorher Beiseitegelegtem.

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

  $\hat{c}_i > c_i \;\Rightarrow$ Wir legen etwas beiseite.

  $c_i > \hat{c}_i \;\Rightarrow$ Wir bezahlen teure Operationen mit vorher Beiseitegelegtem.

- Damit's klappt: wir dürfen nie in die Miesen kommen –

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

  $\hat{c}_i > c_i \;\Rightarrow\;$ Wir legen etwas beiseite.

  $c_i > \hat{c}_i \;\Rightarrow\;$ Wir bezahlen teure Operationen mit vorher Beiseitegelegtem.

- Damit's klappt: wir dürfen nie in die Miesen kommen –

  Guthaben $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$ darf nicht negativ werden!

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

  $\hat{c}_i > c_i \;\Rightarrow\;$ Wir legen etwas beiseite.

  $c_i > \hat{c}_i \;\Rightarrow\;$ Wir bezahlen teure Operationen mit vorher Beiseitegelegtem.

- Damit's klappt: wir dürfen nie in die Miesen kommen –

  Guthaben $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$ darf nicht negativ werden!

  Dann gilt $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$.

Buchhaltermethode

- Verbindet mit jeder Operation op_i *amortisierte* Kosten $\hat{c}_i$, die oft nicht mit den tatsächlichen Kosten c_i übereinstimmen.

  $\hat{c}_i > c_i \;\Rightarrow\;$ Wir legen etwas beiseite.

  $c_i > \hat{c}_i \;\Rightarrow\;$ Wir bezahlen teure Operationen mit vorher Beiseitegelegtem.

- Damit's klappt: wir dürfen nie in die Miesen kommen –

  Guthaben $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$ darf nicht negativ werden!

  Dann gilt $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$.

D.h. amortisierte Kosten sind obere Schranke für tatsächliche Kosten!

Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i =$

Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung

Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

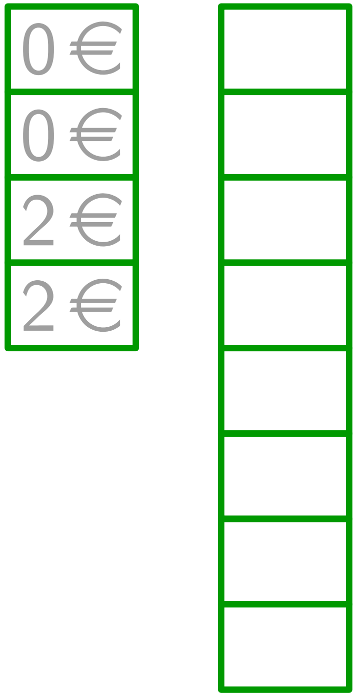
- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung

0 €
0 €
2 €
2 €

Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

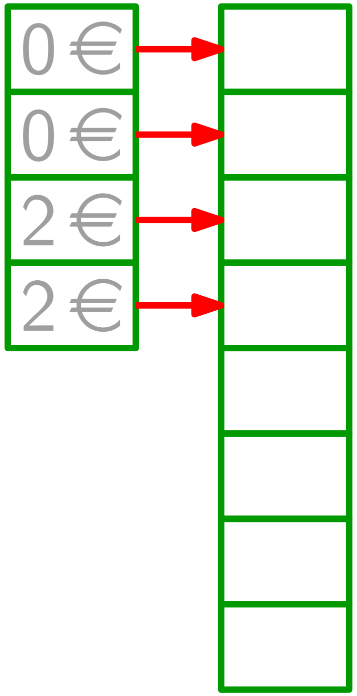
- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung



Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

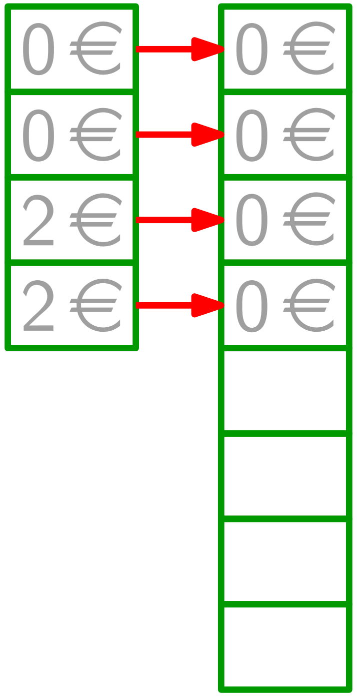
- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung



Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

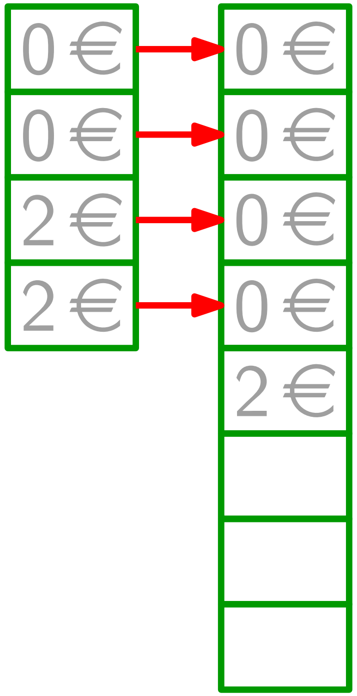
- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung



Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung

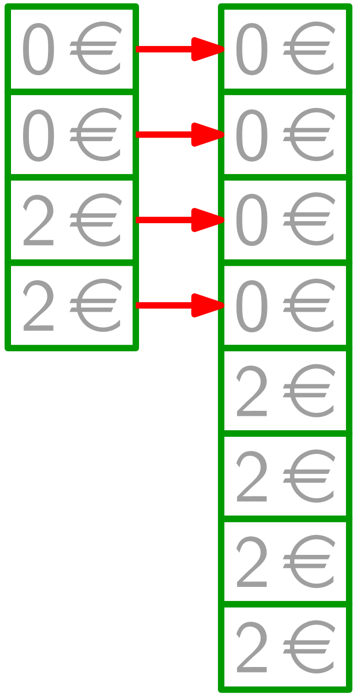


Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

– 1 € fürs tatsächliche Einfügen

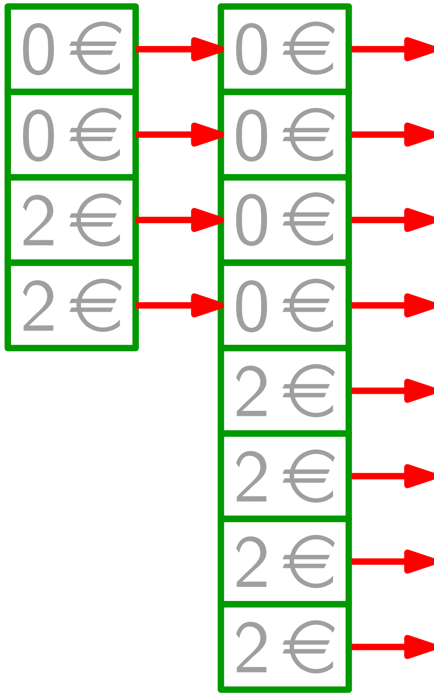
– 2 € für die nächste Tabellenvergrößerung



Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung

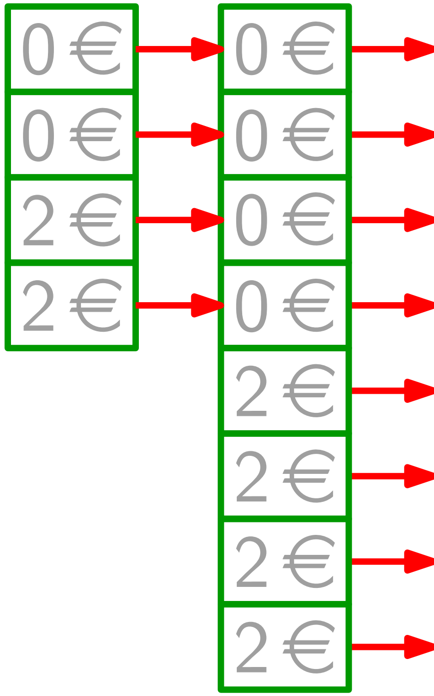


Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung

Wir verknüpfen die Teilguthaben mit konkreten Objekten der Datenstruktur.



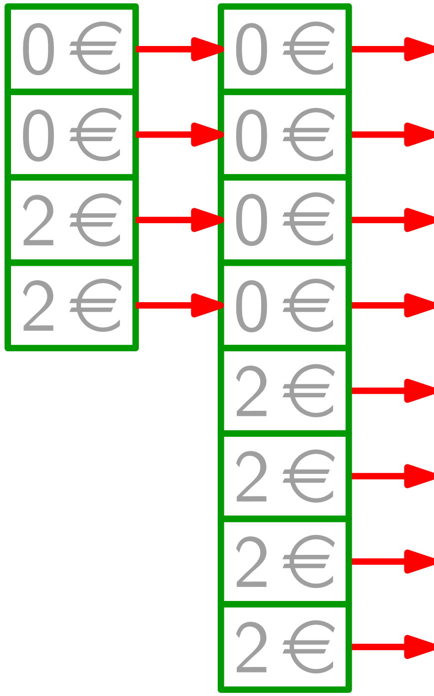
Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung

Wir verknüpfen die Teilguthaben mit konkreten Objekten der Datenstruktur.

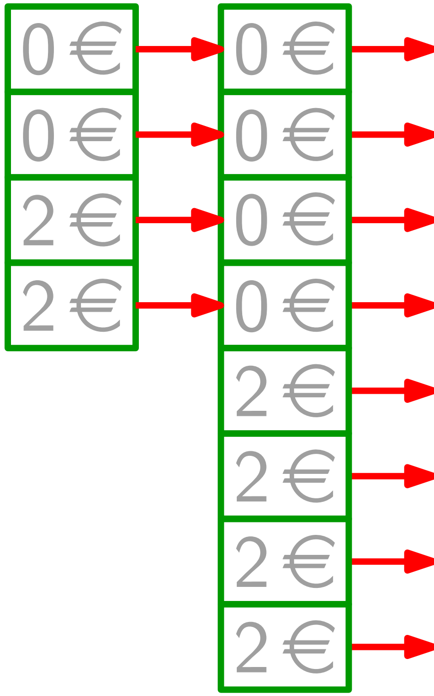
Damit wird deutlich, dass die DS nie Miese macht.



Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung



Wir verknüpfen die Teilguthaben mit konkreten Objekten der Datenstruktur.

Damit wird deutlich, dass die DS nie Miese macht.

$$\Downarrow$$

Also sind amortisierte Kosten obere Schranke für tatsächliche Kosten!

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

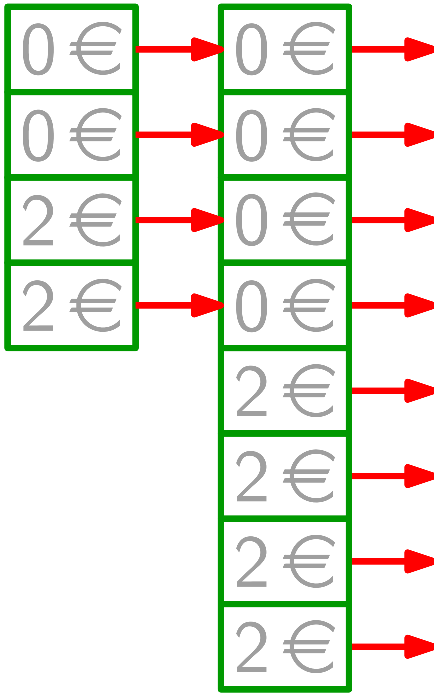
Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

– 1 € fürs tatsächliche Einfügen
– 2 € für die nächste Tabellenvergrößerung

Wir verknüpfen die Teilguthaben mit konkreten Objekten der Datenstruktur.

Damit wird deutlich, dass die DS nie Miese macht.



$$\Downarrow$$

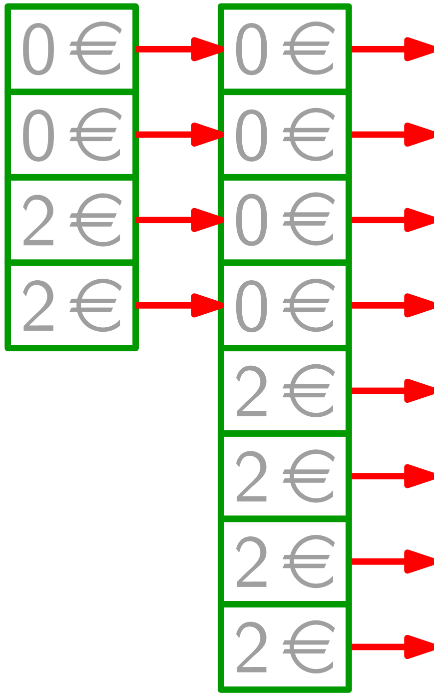
Also sind amortisierte Kosten obere Schranke für tatsächliche Kosten!

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = 3n$$

Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:

- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung



Wir verknüpfen die Teilguthaben mit konkreten Objekten der Datenstruktur.

Damit wird deutlich, dass die DS nie Miese macht.

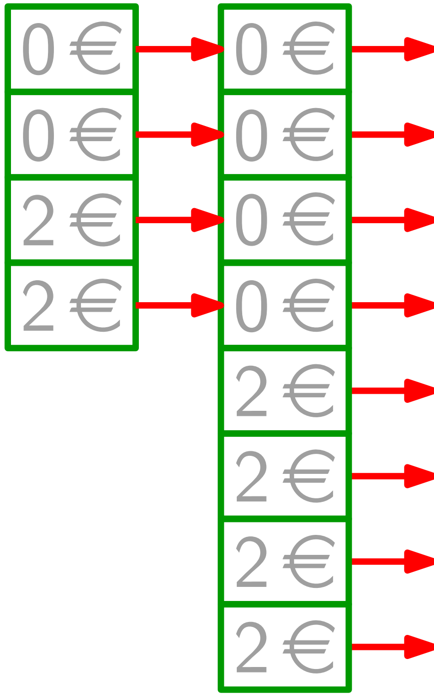
$\Downarrow$

Also sind amortisierte Kosten obere Schranke für tatsächliche Kosten!

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = 3n = \Theta(n)$$

Buchhaltermethode für dynamische Tabellen

Jede Einfügeoperation op_i bezahlt $\hat{c}_i = 3$ €:
- 1 € fürs tatsächliche Einfügen
- 2 € für die nächste Tabellenvergrößerung



Wir verknüpfen die Teilguthaben mit konkreten Objekten der Datenstruktur.

Damit wird deutlich, dass die DS nie Miese macht.

$$\Downarrow$$

Also sind amortisierte Kosten obere Schranke für tatsächliche Kosten!

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i = 3n = \Theta(n)$$

D.h. die (tats.) Kosten für n Einfügeoperationen betragen $\Theta(n)$.

Buchhaltermethode: noch'n Beispiel

Stapel

verwaltet sich ändernde Menge nach *LIFO-Prinzip*

Abs. Datentyp	**Implementierung**
boolean Empty() Push(key k) key Pop() key Top()	

Buchhaltermethode: noch'n Beispiel

Stapel

verwaltet sich ändernde Menge nach *LIFO-Prinzip*

Abs. Datentyp	**Implementierung**
boolean Empty() Push(key k) key Pop() key Top()	
Multipop(int k) *neu!*	

Buchhaltermethode: noch'n Beispiel

Stapel

verwaltet sich ändernde Menge nach *LIFO-Prinzip*

Abs. Datentyp	**Implementierung**
boolean Empty() Push(key k) key Pop() key Top()	

Multipop(int k)
neu!
	while not Empty() **and** $k > 0$ **do**
	    Pop()
	    $k = k - 1$

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push		
Pop		
Multipop(k_i)		

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	
Pop		
Multipop(k_i)		

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	
Pop	1	
Multipop(k_i)		

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	
Pop	1	
Multipop(k_i)	$\min\{k_i, size_i\}$	

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	2
Pop	1	0
Multipop(k_i)	$\min\{k_i, size_i\}$	0

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	2
Pop	1	0
Multipop(k_i)	$\min\{k_i, size_i\}$	0

Geht das gut???

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	2
Pop	1	0
Multipop(k_i)	$\min\{k_i, size_i\}$	0

Geht das gut???

Zeige: Amortisierte Kosten „bezahlen“ immer für die echten!

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	2
Pop	1	0
Multipop(k_i)	$\min\{k_i, size_i\}$	0

Geht das gut???

Zeige: Amortisierte Kosten „bezahlen" immer für die echten!

- Jede Push-Operation legt einen Teller auf den Stapel.

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	2
Pop	1	0
Multipop(k_i)	$\min\{k_i, \mathit{size}_i\}$	0

Geht das gut???

Zeige: Amortisierte Kosten „bezahlen" immer für die echten!

- Jede Push-Operation legt einen Teller auf den Stapel. Dafür bezahlt sie 1 € und legt noch 1 € auf den Teller.

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	2
Pop	1	0
Multipop(k_i)	$\min\{k_i, size_i\}$	0

Geht das gut???

Zeige: Amortisierte Kosten „bezahlen“ immer für die echten!

- Jede Push-Operation legt einen Teller auf den Stapel. Dafür bezahlt sie 1 € und legt noch 1 € auf den Teller.
- Jede (Multi-)Pop-Operation wird mit den Euros auf den Tellern, die sie wegnimmt, komplett bezahlt.

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation$_i$	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	2
Pop	1	0
Multipop(k_i)	$\min\{k_i, \mathit{size}_i\}$	0

Geht das gut??? – *Ja!*

Zeige: Amortisierte Kosten „bezahlen“ immer für die echten!

- Jede Push-Operation legt einen Teller auf den Stapel. Dafür bezahlt sie 1 € und legt noch 1 € auf den Teller.
- Jede (Multi-)Pop-Operation wird mit den Euros auf den Tellern, die sie wegnimmt, komplett bezahlt.

Buchhaltermethode: Stapel mit Multipop

Betrachte Folge von Push-, Pop- und Multipop-Operationen.

Operation_i	tatsächliche Kosten c_i	amortisierte Kosten $\hat{c}_i$
Push	1	2
Pop	1	0
Multipop(k_i)	$\min\{k_i, size_i\}$	0

Geht das gut??? – *Ja!* D.h. Folge von n Op. dauert $\Theta(n)$ Zeit.

Zeige: Amortisierte Kosten „bezahlen“ immer für die echten!

- Jede Push-Operation legt einen Teller auf den Stapel. Dafür bezahlt sie 1 € und legt noch 1 € auf den Teller.
- Jede (Multi-)Pop-Operation wird mit den Euros auf den Tellern, die sie wegnimmt, komplett bezahlt.

Amortisierte Analyse

– Aggregationsmethode
– Buchhaltermethode
– Potentialmethode

Amortisierte Analyse

- Aggregationsmethode ✓
- Buchhaltermethode ✓
- Potentialmethode

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \longrightarrow D_1 \longrightarrow \cdots \longrightarrow D_n$

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$.

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \ldots, n$.

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \ldots, n$.

Def. $\hat{c}_i = c_i + \Delta\Phi(D_i)$

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \dots, n$.

amortisierte Kosten *echte Kosten*

Def. $\hat{c}_i = c_i + \Delta\Phi(D_i)$

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \dots, n$.

amortisierte Kosten *echte Kosten* *Potentialdifferenz*

Def. $\hat{c}_i = c_i + \Delta\Phi(D_i)$, wobei $\Delta\Phi(D_i) = \Phi(D_i) - \Phi(D_{i-1})$

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \ldots, n$.

amortisierte Kosten *echte Kosten* *Potentialdifferenz*

Def. $\hat{c}_i = c_i + \Delta\Phi(D_i)$, wobei $\Delta\Phi(D_i) = \Phi(D_i) - \Phi(D_{i-1})$

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

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \dots, n$.

amortisierte Kosten *echte Kosten* *Potentialdifferenz*

Def. $\hat{c}_i = c_i + \Delta\Phi(D_i)$, wobei $\Delta\Phi(D_i) = \Phi(D_i) - \Phi(D_{i-1})$

Folge: $\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right)$ *teleskopierende Summe*

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi\colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \ldots, n$.

amortisierte Kosten *echte Kosten* *Potentialdifferenz*

Def. $\hat{c}_i = c_i + \Delta\Phi(D_i)$, wobei $\Delta\Phi(D_i) = \Phi(D_i) - \Phi(D_{i-1})$

Folge: $\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right)$ *teleskopierende Summe*

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

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi \colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \ldots, n$.

amortisierte Kosten *echte Kosten* *Potentialdifferenz*

Def. $\hat{c}_i = c_i + \Delta\Phi(D_i)$, wobei $\Delta\Phi(D_i) = \Phi(D_i) - \Phi(D_{i-1})$

Folge: $\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right)$ *teleskopierende Summe*

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

Potentialmethode

Idee: Betrachte Bankguthaben (siehe Buchhaltermethode)
als physikalische Größe,
die den augenblicklichen Zustand der DS beschreibt.

Datenstruktur $D_0 \xrightarrow{op_1} D_1 \xrightarrow{op_2} \cdots \xrightarrow{op_n} D_n$

Wähle Potential $\Phi \colon D_i \to \mathbb{R}$. O.B.d.A. $\Phi(D_0) = 0$

Ziel: Bank macht keine Miesen.
Also fordern wir $\Phi(D_i) \geq 0$ für $i = 1, \ldots, n$.

amortisierte Kosten *echte Kosten* *Potentialdifferenz*

Def. $\hat{c}_i = c_i + \Delta\Phi(D_i)$, wobei $\Delta\Phi(D_i) = \Phi(D_i) - \Phi(D_{i-1})$

Folge: $\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right)$ *teleskopierende Summe*

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

D.h. amortisierte Kosten „bezahlen“ für echte Kosten.

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee:

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \ldots, \Phi(D_n) \geq 0$.

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe:

Was sind die amort. Kosten?

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:

Was sind die amort. Kosten?

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = \mathit{size}_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$

Was sind die amort. Kosten?

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = \mathit{size}_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i$

Was sind die amort. Kosten?

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = \mathit{size}_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, size_i\}$

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

$$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, size_i\}$$
$$c_i = +\min\{k_i, size_i\}$$

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = \mathit{size}_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

$$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, \mathit{size}_i\}$$
$$c_i = +\min\{k_i, \mathit{size}_i\}$$

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

$$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, size_i\}$$

$$c_i = +\min\{k_i, size_i\}$$

$$\hat{c}_i = c_i + \Delta\Phi D_i$$

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = \mathit{size}_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

$$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, \mathit{size}_i\}$$
$$c_i = +\min\{k_i, \mathit{size}_i\}$$
$$\hat{c}_i = c_i + \Delta\Phi D_i = 0$$

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

$$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, size_i\}$$
$$c_i = +\min\{k_i, size_i\}$$

$\hat{c}_i = c_i + \Delta\Phi D_i = 0$, dito mit Pop $(k_i = 1)$.

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = \mathit{size}_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

$$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, \mathit{size}_i\}$$

$$c_i = +\min\{k_i, \mathit{size}_i\}$$

$$\hat{c}_i = c_i + \Delta\Phi D_i = 0, \text{ dito mit Pop } (k_i = 1).$$

Also:

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = \mathit{size}_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

$$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, \mathit{size}_i\}$$
$$c_i = +\min\{k_i, \mathit{size}_i\}$$

$\hat{c}_i = c_i + \Delta\Phi D_i = 0$, dito mit Pop $(k_i = 1)$.

Also: Amortisierte Kosten pro Operation $O(1)$.

Potentialmethode: Stapel mit Multipop

To do: Definiere Potentialfunktion –
in Abhängigkeit vom aktuellen Zustand des Stapels!

Idee: Nimm $\Phi(D_i) = size_i$, also aktuelle Stapelgröße.
$\Rightarrow \Phi(D_0) = 0$ und $\Phi(D_1), \dots, \Phi(D_n) \geq 0$. ✓

Prüfe: Falls die i-te Operation eine Push-Operation ist:
$\Rightarrow \Delta\Phi(D_i) = +1$ und $\hat{c}_i = c_i + \Delta\Phi D_i = 1 + 1 = 2$

Was sind die amort. Kosten?

Falls die i-te Operation eine Multipop-Operation ist:

$$\Rightarrow \Delta\Phi(D_i) = -\min\{k_i, size_i\}$$
$$c_i = +\min\{k_i, size_i\}$$

$\hat{c}_i = c_i + \Delta\Phi D_i = 0$, dito mit Pop $(k_i = 1)$.

Also: Amortisierte Kosten pro Operation $O(1)$.
$\Rightarrow$ Echte Kosten für n Oper. im worst case $O(n)$.

Zusammenfassung

Zeige mit amortisierter Analyse, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben – *auch wenn einzelne Operationen in der Folge teuer sind!*

Zusammenfassung

Zeige mit amortisierter Analyse, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben – *auch wenn einzelne Operationen in der Folge teuer sind!*

Drei Typen von amortisierter Analyse:

– Aggregationsmethode

– Buchhaltermethode

– Potentialmethode

Zusammenfassung

Zeige mit amortisierter Analyse, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben – *auch wenn einzelne Operationen in der Folge teuer sind!*

Drei Typen von amortisierter Analyse:

- Aggregationsmethode ✓
  Summiere tatsächliche Kosten (oder obere Schranken dafür) auf.
- Buchhaltermethode
- Potentialmethode

Zusammenfassung

Zeige mit amortisierter Analyse, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben – *auch wenn einzelne Operationen in der Folge teuer sind!*

Drei Typen von amortisierter Analyse:

– Aggregationsmethode ✓
  Summiere tatsächliche Kosten (oder obere Schranken dafür) auf.

– Buchhaltermethode ✓
  Verbinde Extrakosten mit konkreten Objekten der DS und bezahle damit teure Operationen.

– Potentialmethode

Zusammenfassung

Zeige mit amortisierter Analyse, dass die Operationen einer gegebenen Folge kleine durchschnittliche Kosten haben – *auch wenn einzelne Operationen in der Folge teuer sind!*

Drei Typen von amortisierter Analyse:

- Aggregationsmethode ✓
  Summiere tatsächliche Kosten (oder obere Schranken dafür) auf.
- Buchhaltermethode ✓
  Verbinde Extrakosten mit konkreten Objekten der DS und bezahle damit teure Operationen.
- Potentialmethode ✓
  Definiere Potential der gesamten DS, so dass mit der Potentialdifferenz teure Operationen bezahlt werden können.

Übungsaufgaben zur amortisierten Analyse (I)

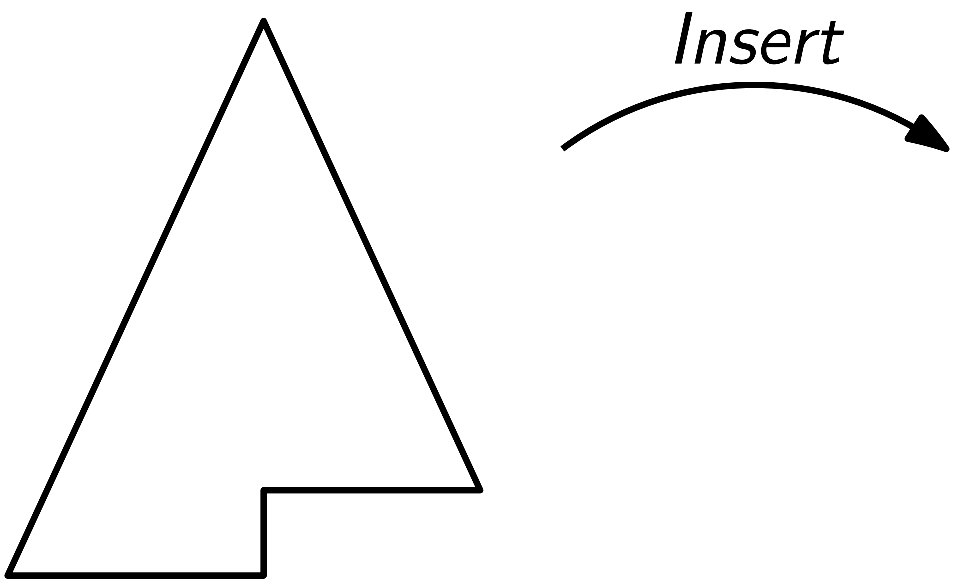
Gegeben sei ein gewöhnlicher MinHeap, dessen Methoden *Insert* und *ExtractMin* im schlechtesten Fall $O(\log n)$ Zeit brauchen.

Zeigen Sie mit der Potentialmethode, dass *Insert* amortisiert $O(\log n)$ Zeit und *ExtractMin* amortisiert $O(1)$ Zeit benötigt.

Übungsaufgaben zur amortisierten Analyse (I)

Gegeben sei ein gewöhnlicher MinHeap, dessen Methoden *Insert* und *ExtractMin* im schlechtesten Fall $O(\log n)$ Zeit brauchen.

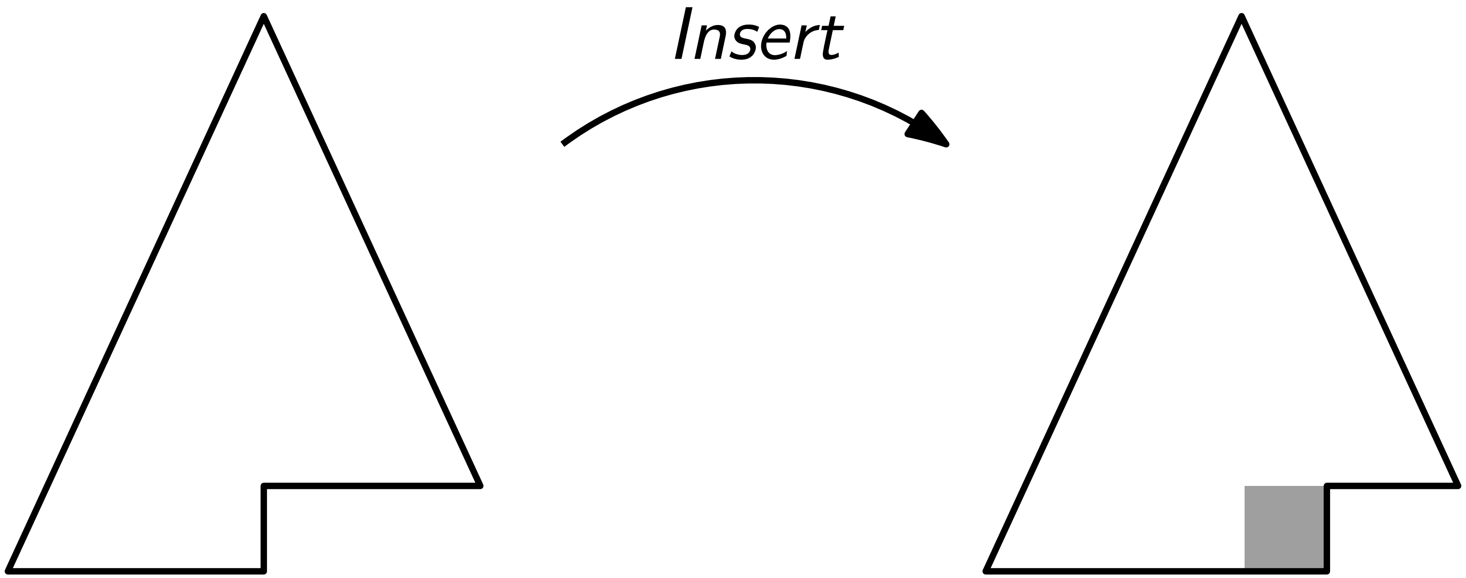
Zeigen Sie mit der Potentialmethode, dass *Insert* amortisiert $O(\log n)$ Zeit und *ExtractMin* amortisiert $O(1)$ Zeit benötigt.



Übungsaufgaben zur amortisierten Analyse (I)

Gegeben sei ein gewöhnlicher MinHeap, dessen Methoden *Insert* und *ExtractMin* im schlechtesten Fall $O(\log n)$ Zeit brauchen.

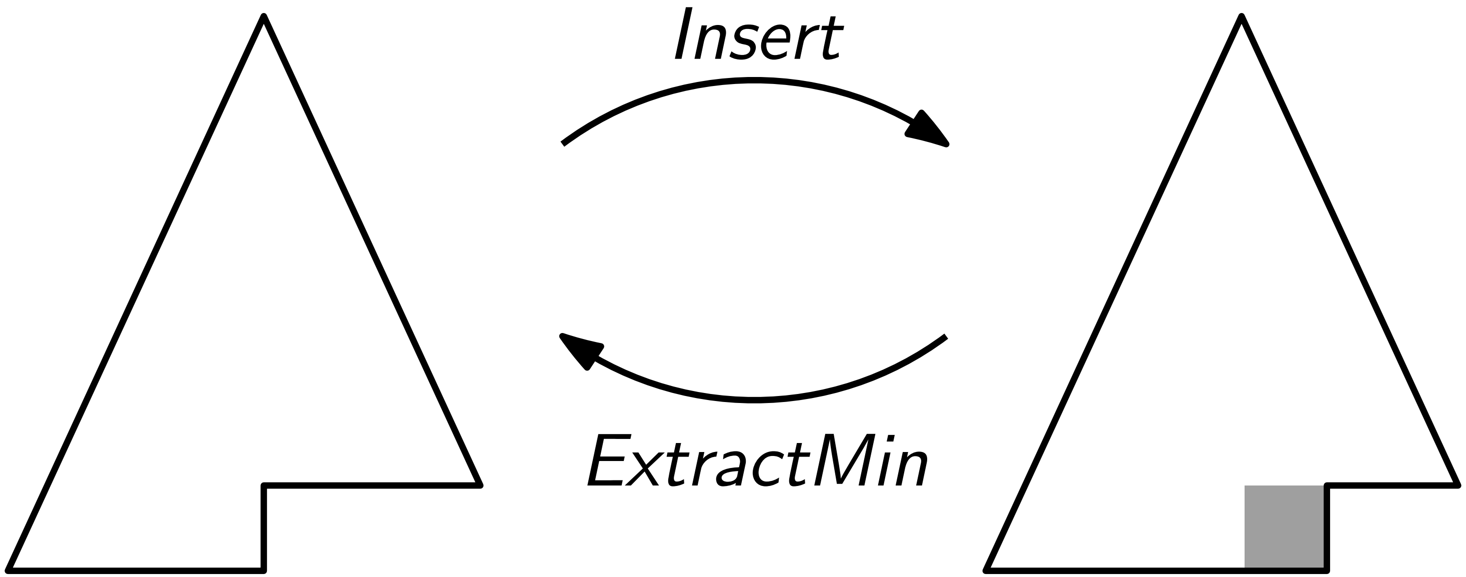
Zeigen Sie mit der Potentialmethode, dass *Insert* amortisiert $O(\log n)$ Zeit und *ExtractMin* amortisiert $O(1)$ Zeit benötigt.



Übungsaufgaben zur amortisierten Analyse (I)

Gegeben sei ein gewöhnlicher MinHeap, dessen Methoden *Insert* und *ExtractMin* im schlechtesten Fall $O(\log n)$ Zeit brauchen.

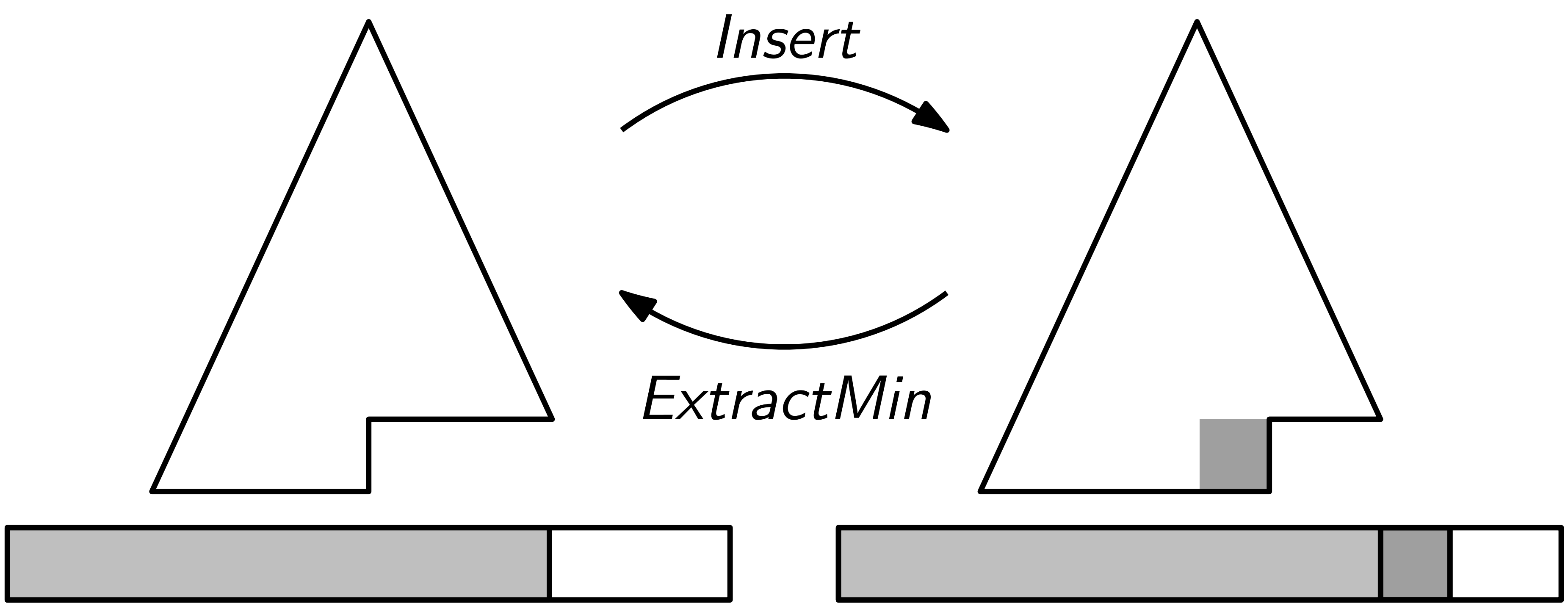
Zeigen Sie mit der Potentialmethode, dass *Insert* amortisiert $O(\log n)$ Zeit und *ExtractMin* amortisiert $O(1)$ Zeit benötigt.



Übungsaufgaben zur amortisierten Analyse (I)

Gegeben sei ein gewöhnlicher MinHeap, dessen Methoden *Insert* und *ExtractMin* im schlechtesten Fall $O(\log n)$ Zeit brauchen.

Zeigen Sie mit der Potentialmethode, dass *Insert* amortisiert $O(\log n)$ Zeit und *ExtractMin* amortisiert $O(1)$ Zeit benötigt.



Übungsaufgaben zur amortisierten Analyse (II)

Entwerfen Sie eine Datenstruktur zum Verwalten einer dynamischen Menge von Zahlen. Die DS soll 2 Methoden haben:

- *Insert* zum Einfügen einer Zahl und
- *DeleteLargerHalf* zum Löschen aller Zahlen aus der Datenstruktur, die größer oder gleich dem aktuellen Median der Zahlenmenge sind.

Übungsaufgaben zur amortisierten Analyse (II)

Entwerfen Sie eine Datenstruktur zum Verwalten einer dynamischen Menge von Zahlen. Die DS soll 2 Methoden haben:

- *Insert* zum Einfügen einer Zahl und
- *DeleteLargerHalf* zum Löschen aller Zahlen aus der Datenstruktur, die größer oder gleich dem aktuellen Median der Zahlenmenge sind.

Beide Methoden sollen amortisiert $O(1)$ Zeit benötigen.

Übungsaufgaben zur amortisierten Analyse (II)

Entwerfen Sie eine Datenstruktur zum Verwalten einer dynamischen Menge von Zahlen. Die DS soll 2 Methoden haben:

- *Insert* zum Einfügen einer Zahl und
- *DeleteLargerHalf* zum Löschen aller Zahlen aus der Datenstruktur, die größer oder gleich dem aktuellen Median der Zahlenmenge sind.

Beide Methoden sollen amortisiert $O(1)$ Zeit benötigen.

Tipp: Verwenden Sie eine Liste!

Übungsaufgaben zur amortisierten Analyse (II)

Entwerfen Sie eine Datenstruktur zum Verwalten einer dynamischen Menge von Zahlen. Die DS soll 2 Methoden haben:

- *Insert* zum Einfügen einer Zahl und
- *DeleteLargerHalf* zum Löschen aller Zahlen aus der Datenstruktur, die größer oder gleich dem aktuellen Median der Zahlenmenge sind.

Beide Methoden sollen amortisiert $O(1)$ Zeit benötigen.
Tipp: Verwenden Sie eine Liste!

1. Beschreiben Sie Ihren Entwurf der Datenstruktur einschließlich der beiden Methoden in Worten.

2. Analysieren Sie mithilfe der Buchhaltermethode.
   Geben Sie die amortisierten Kosten, die Sie mit *Insert* und *DeleteLargerHalf* verbinden, exakt an.