

# Tutorium 2

## Algorithmen I SS 14

Institut für Theoretische Informatik



# Übungsblatt 1.

# Übungsblatt 2?

$$T(n) = \begin{cases} 1, & \text{falls } n = 1 \\ 2T(\lfloor n/2 \rfloor) + n, & \text{falls } n \geq 2 \end{cases}$$

$$n \in 2^m \text{ für } m \in \mathbb{N}_0$$

## Vorgehen

- 1 Lösung raten...
- 2 Beweisen, dass die Lösung stimmt

Umformung:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \text{ mit } T(1) = 1$$

$$T(2^m) = 2 * T(2^{m-1}) + 2^m \text{ mit } T(1) = 1$$

$$S(m) = 2 * S(m-1) + 2^m \text{ mit } S(0) = 1$$

Ausrechnen der ersten Werte:

$$S(0) = T(2^0) = T(1) = 1$$

$$S(1) = T(2^1) = T(2) = 2 * 1 + 2 = 4$$

$$S(2) = T(2^2) = T(4) = 2 * (2 * 1 + 2) + 4 = 12$$

$$S(3) = T(2^3) = T(8) = 2 * (2 * (2 * 1 + 2) + 4) + 8 = 32$$

$$S(4) = T(2^4) = T(16) = 2 * (2 * (2 * (2 * 1 + 2) + 4) + 8) + 16 = 70$$

$$S(m) = 2 * S(m-1) + 2^m \text{ mit } S(0) = 1$$

$$S(0) = 1 = 2^0$$

$$S(1) = 2 * 1 + 2 = 2^1 + 2^1 = 2^2$$

$$S(2) = 2 * (2 * 1 + 2) + 4 = 2^3 + 2^2$$

$$S(3) = 2 * (2 * (2 * 1 + 2) + 4) + 8 = 2^4 + 2 * 2^3$$

$$S(4) = 2 * (2 * (2 * (2 * 1 + 2) + 4) + 8) + 16 = 2^5 + 3 * 2^4$$

Durch scharfes Hinsehen erkennen wir eine mögliche Lösung:

$$S(m) = 2^{m+1} + (m-1) * 2^m$$

Beweis der Lösung durch vollständige Induktion.

**IA:**  $S(0) = T(2^0) = T(1) = 1$

**IV:**  $S(m) = 2^{m+1} + (m-1) * 2^m$

**IS:**  $m \rightarrow m+1$

$$S(m+1) = 2 * S(m) + 2^{m+1} \quad (1)$$

$$= 2 * (2^{m+1} + (m-1) * 2^m) + 2^{m+1} \quad (2)$$

$$= 2^{m+2} + (m-1) * 2^{m+1} + 2^{m+1} \quad (3)$$

$$= 2^{m+2} + m * 2^{m+1} \quad (4)$$

Zurück nach  $T(n)$  rechnen für Finale Lösung:

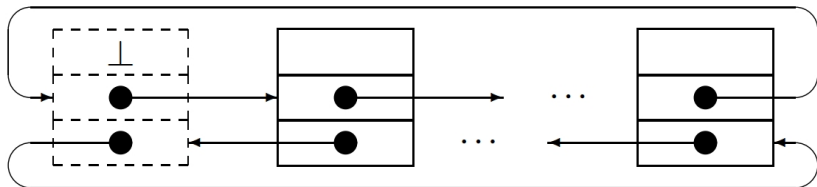
$$T(n) = S(\log_2 n) = 2^{\log_2 n + 1} * (\log_2 n - 1) * 2^{\log_2 n}$$

Class Item of Element:

```
e : Element  
next : Pointer to Item  
prev : Pointer to Item
```

Class List of Element:

```
h = Item( $\perp$ ,  
h,  
h)
```





- Einfügen am Anfang
- Einfügen am Ende
- Element finden
- Element entfernen (mit Verweis auf Element)
- Zusammenfügen zweier Listen
- Zugriff auf das  $k$ -te Element
- Aufteilen einer Liste in zwei (mit Verweis auf Stelle)
- Länge der Liste bestimmen

- Einfügen am Anfang  $\mathcal{O}(1)$
- Einfügen am Ende
- Element finden
- Element entfernen (mit Verweis auf Element)
- Zusammenfügen zweier Listen
- Zugriff auf das  $k$ -te Element
- Aufteilen einer Liste in zwei (mit Verweis auf Stelle)
- Länge der Liste bestimmen

- Einfügen am Anfang  $\mathcal{O}(1)$
- Einfügen am Ende  $\mathcal{O}(1)$
- Element finden
- Element entfernen (mit Verweis auf Element)
- Zusammenfügen zweier Listen
- Zugriff auf das  $k$ -te Element
- Aufteilen einer Liste in zwei (mit Verweis auf Stelle)
- Länge der Liste bestimmen

- Einfügen am Anfang  $\mathcal{O}(1)$
- Einfügen am Ende  $\mathcal{O}(1)$
- Element finden  $\mathcal{O}(n)$
- Element entfernen (mit Verweis auf Element)
- Zusammenfügen zweier Listen
- Zugriff auf das  $k$ -te Element
- Aufteilen einer Liste in zwei (mit Verweis auf Stelle)
- Länge der Liste bestimmen

- Einfügen am Anfang  $\mathcal{O}(1)$
- Einfügen am Ende  $\mathcal{O}(1)$
- Element finden  $\mathcal{O}(n)$
- Element entfernen (mit Verweis auf Element)  $\mathcal{O}(1)$
- Zusammenfügen zweier Listen
- Zugriff auf das  $k$ -te Element
- Aufteilen einer Liste in zwei (mit Verweis auf Stelle)
- Länge der Liste bestimmen

- Einfügen am Anfang  $\mathcal{O}(1)$
- Einfügen am Ende  $\mathcal{O}(1)$
- Element finden  $\mathcal{O}(n)$
- Element entfernen (mit Verweis auf Element)  $\mathcal{O}(1)$
- Zusammenfügen zweier Listen  $\mathcal{O}(1)$
- Zugriff auf das  $k$ -te Element
- Aufteilen einer Liste in zwei (mit Verweis auf Stelle)
- Länge der Liste bestimmen

- Einfügen am Anfang  $\mathcal{O}(1)$
- Einfügen am Ende  $\mathcal{O}(1)$
- Element finden  $\mathcal{O}(n)$
- Element entfernen (mit Verweis auf Element)  $\mathcal{O}(1)$
- Zusammenfügen zweier Listen  $\mathcal{O}(1)$
- Zugriff auf das  $k$ -te Element  $\mathcal{O}(n)$
- Aufteilen einer Liste in zwei (mit Verweis auf Stelle)
- Länge der Liste bestimmen

- Einfügen am Anfang  $\mathcal{O}(1)$
- Einfügen am Ende  $\mathcal{O}(1)$
- Element finden  $\mathcal{O}(n)$
- Element entfernen (mit Verweis auf Element)  $\mathcal{O}(1)$
- Zusammenfügen zweier Listen  $\mathcal{O}(1)$
- Zugriff auf das  $k$ -te Element  $\mathcal{O}(n)$
- Aufteilen einer Liste in zwei (mit Verweis auf Stelle)  $\mathcal{O}(1)$
- Länge der Liste bestimmen



- Einfügen am Anfang  $\mathcal{O}(1)$
- Einfügen am Ende  $\mathcal{O}(1)$
- Element finden  $\mathcal{O}(n)$
- Element entfernen (mit Verweis auf Element)  $\mathcal{O}(1)$
- Zusammenfügen zweier Listen  $\mathcal{O}(1)$
- Zugriff auf das  $k$ -te Element  $\mathcal{O}(n)$
- Aufteilen einer Liste in zwei (mit Verweis auf Stelle)  $\mathcal{O}(1)$
- Länge der Liste bestimmen  $\mathcal{O}(n)$

Implementiert folgende Operationen für eine doppelt verkettete, *sortierte* Liste in Pseudocode:

- 1 *insert*(*e*) (Einfügen des Elements *e* an die richtige Stelle)
- 2 *findFirst*(*e*) (Erstes Vorkommen von *e* in der Liste)
- 3 *remove*(*e*) (Entfernen des Elements *e* aus der Liste)

- gemittelter Aufwand für eine Sequenz von Operationen im Worst-case
- Kosten einer Operation gering, auch wenn eine einzelne Operation kostspielig ist
- keine Einbeziehung von Wahrscheinlichkeiten
- Beispiel: unbounded Arrays

für jede Operation:

- $\hat{c}_i$  **amortisierte Kosten**
- $c_i$  **tatsächliche Kosten**
- $\hat{c}_i \geq c_i \Rightarrow$  Guthaben einzahlen
- $\hat{c}_i \leq c_i \Rightarrow$  Guthaben abheben
- Konto darf **nie** negativ werden
  - $\forall n \in \mathbb{N} : c + \sum_{i=1}^n \hat{c}_i \geq \sum_{i=1}^n c_i$

- falls ein Feld mit  $n$  Elementen durch **reallocate** verkleinert oder vergrößert wird, müssen  $n$  Elemente kopiert werden  $\Rightarrow n$  Tokens müssen vorhanden sein
- amortisierte Kosten:
  - **push**: 3 Tokens
    - 1 Token: Einfügen des Elements
    - 2 Tokens: Konto
  - **pop**: 2 Tokens
    - 1 Token: Löschen des Elements
    - 1 Token: Konto

- Anfangszustand nach **reallocate**:
  - $n$  Elemente, kein Guthaben vorhanden
- Sequenz von **push**-Operationen:
  - **reallocate** wird nach  $n$  **push**-Operationen aufgerufen
  - Guthaben:  $2n$  Tokens
  - **reallocate**-Aufruf:  $2n$  Elemente werden kopiert  $\rightarrow$  genug Tokens
- Sequenz von **pop**-Operationen:
  - **reallocate** wird nach  $\frac{n}{2}$  **pop**-Operationen aufgerufen
  - Guthaben:  $\frac{n}{2}$  Tokens
  - **reallocate**-Aufruf:  $\frac{n}{2}$  Elemente werden kopiert  $\rightarrow$  genug Tokens

Entwickelt eine Datenstruktur mit folgenden Eigenschaften:

- *pushBack* und *popBack* in  $\mathcal{O}(1)$
- Zugriff auf das  $k$ -te Element in  $\mathcal{O}(\log n)$

Jeweils für Worst-Case und nicht amortisiert. Speicherallokation ist dabei immer in  $\mathcal{O}(1)$ .

Jetzt umgekehrt:

- *pushBack* und *popBack* in  $\mathcal{O}(\log n)$
- Zugriff auf das  $k$ -te Element in  $\mathcal{O}(1)$

Jeweils für Worst-Case und nicht amortisiert. Speicherallokation ist dabei immer in  $\mathcal{O}(1)$ .



# Fragen?

Folien <https://github.com/vincent23/algo1-tut-ss14>

Mail [v.schuessler@gmail.com](mailto:v.schuessler@gmail.com)

# Is it worth the time?

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?  
(ACROSS FIVE YEARS)

		HOW OFTEN YOU DO THE TASK					
		50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
HOW MUCH TIME YOU SHAVE OFF	1 SECOND	<div><div>1</div></div> DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
	5 SECONDS	<div><div>5</div></div> DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
	30 SECONDS	<div><div></div><div></div><div></div><div></div><div></div><div></div></div> 4 WEEKS	<div><div>3</div></div> DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
	1 MINUTE	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div> 8 WEEKS	<div><div>6</div></div> DAYS	<div><div>1</div></div> DAY	4 HOURS	1 HOUR	5 MINUTES
	5 MINUTES	9 MONTHS	<div><div></div><div></div><div></div><div></div><div></div><div></div></div> 4 WEEKS	<div><div>6</div></div> DAYS	21 HOURS	5 HOURS	25 MINUTES
	30 MINUTES		6 MONTHS	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div> 5 WEEKS	<div><div>5</div></div> DAYS	<div><div>1</div></div> DAY	2 HOURS
	1 HOUR		10 MONTHS	2 MONTHS	<div><div>10</div></div> DAYS	<div><div>2</div></div> DAYS	5 HOURS
	6 HOURS				2 MONTHS	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div> 2 WEEKS	<div><div>1</div></div> DAY
	<div><div>1</div></div> DAY					<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div> 8 WEEKS	<div><div>5</div></div> DAYS