Karlsruher Institut für Technologie

Algorithmen I
Tutorium 33

Woche 2 | 04. Mai 2018

Daniel Jungkind (daniel.jungkind@student.kit.edu)

INSTITUT FÜR THEORETISCHE INFORMATIK

Inhalt

Laufzeiten

Das Master-Theorem

Quiz

Für alle Funktionen f, g gilt $f(n) < g(n)$ oder $f(n) = g(n)$ oder $f(n) > g(n)$. **?**

Quiz

Für alle Funktionen f, g gilt $f(n) < g(n)$ oder $f(n) = g(n)$ oder $f(n) > g(n)$.

Falsch.

Quiz

Für alle Funktionen f, g gilt $f(n) < g(n)$ oder $f(n) = g(n)$ oder $f(n) > g(n)$. **Falsch.**

$e^n \in \mathcal{O}(2^n)$. **?**

Quiz

Für alle Funktionen f, g gilt $f(n) < g(n)$ oder $f(n) = g(n)$ oder $f(n) > g(n)$. **Falsch.**

$e^n \in \mathcal{O}(2^n)$. **Falsch.**

Quiz

Für alle Funktionen f, g gilt $f(n) < g(n)$ oder $f(n) = g(n)$ oder $f(n) > g(n)$. **Falsch.**

$e^n \in \mathcal{O}(2^n)$. **Falsch.**

$e^n \in \mathcal{O}(n!)$. **?**

Quiz

Für alle Funktionen f, g gilt $f(n) < g(n)$ oder $f(n) = g(n)$ oder $f(n) > g(n)$. **Falsch.**

$e^n \in \mathcal{O}(2^n)$. **Falsch.**

$e^n \in \mathcal{O}(n!)$. **Wahr.**

Quiz

Für alle Funktionen f, g gilt $f(n) < g(n)$ oder $f(n) = g(n)$ oder $f(n) > g(n)$. **Falsch.**

$e^n \in \mathcal{O}(2^n)$. **Falsch.**

$e^n \in \mathcal{O}(n!)$. **Wahr.**

Für Pseudocode gilt die DIN 1946-6. **?**

Quiz

Für alle Funktionen f, g gilt $f(n) < g(n)$ oder $f(n) = g(n)$ oder $f(n) > g(n)$. **Falsch.**

$e^n \in \mathcal{O}(2^n)$. **Falsch.**

$e^n \in \mathcal{O}(n!)$. **Wahr.**

Für Pseudocode gilt die DIN 1946-6. **Falsch.**

Diese Norm regelt die Anzahl und Dauer von Lüftvorgängen in Wohnungen.

Laufzeiten

...denn Zeit ist Geld

Laufzeitabschätzung

Laufzeit?

```
function boing(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        if i > n − 4 then
            for j := 1 to n do
                k++
    return k + ℓ
```

Laufzeitabschätzung

Laufzeit?

```
function boing(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        if i > n − 4 then
            for j := 1 to n do
                k++
    return k + ℓ
```

- Erster Gedanke:
  Äußere Schleife: n Durchläufe,
  Innere Schleife: n Durchläufe
  $\Rightarrow$ Laufzeit in $\Theta(n \cdot n) = \Theta(n^2)$
- Aber: Innere Schleife wird nur max. 4x erreicht (nämlich für $i \in \{n-3, n-2, n-1, n\}$)
  $\Rightarrow$ Laufzeit in $\Theta(n + 4n) = \Theta(n)$

Laufzeitabschätzung

Laufzeit?

```
function boing(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        if i > n − 4 then
            for j := 1 to n do
                k++
    return k + ℓ
```

- Erster Gedanke:
  Äußere Schleife: n Durchläufe,
  Innere Schleife: n Durchläufe
  $\Rightarrow$ Laufzeit in ~~$\Theta(n \cdot n) = \Theta(n^2)$~~
- **Aber**: Innere Schleife wird nur **max. 4x** erreicht
  (nämlich für
  $i \in \{n-3, n-2, n-1, n\}$)

  $\Rightarrow$ Laufzeit in $\Theta(n + 4n) = \Theta(n)$

Laufzeitabschätzung

Laufzeit?

```
function doing(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        for j := i to n do
            k++
    return k + ℓ
```

Laufzeitabschätzung

- Erster Gedanke: Äußere Schleife macht n-mal „irgendwas“
  $\Rightarrow\ n \cdot (???)$
  (Klammer? Wie schreiben wir das auf? Nicht n-mal dasselbe, sondern **von** *i* **abhängig**!)
- Rettung: Anzahl innere Schleifendurchläufe **einzeln** für jedes $i = 1, \ldots, n$ **aufsummieren**!
- Für ein festes i wird innere Schleife $(n - i + 1)$-mal durchlaufen $\Rightarrow$ Gesamtanzahl der inneren Schleifendurchläufe:

$$\sum_{i=1}^{n}(n - i + 1) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 = n^2 - \sum_{i=1}^{n} i + n$$

$$= n^2 - \frac{n \cdot (n+1)}{2} + n = n^2 - \frac{n^2}{2} + n - \frac{n}{2} = \frac{n^2 + n}{2} \in \Theta(n^2).$$

Laufzeitabschätzung

- Erster Gedanke: Äußere Schleife macht n-mal „irgendwas“
  $\Rightarrow\ n \cdot (???)$
  (Klammer? Wie schreiben wir das auf? Nicht n-mal dasselbe, sondern **von** i **abhängig**!)
- Rettung: Anzahl innere Schleifendurchläufe **einzeln** für jedes $i = 1, \ldots, n$ **aufsummieren**!
- Für ein festes i wird innere Schleife $(n - i + 1)$-mal durchlaufen $\Rightarrow$ Gesamtanzahl der inneren Schleifendurchläufe:

$$\sum_{i=1}^{n} (n - i + 1) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 = n^2 - \sum_{i=1}^{n} i + n$$

$$= n^2 - \frac{n \cdot (n+1)}{2} + n = n^2 - \frac{n^2}{2} + n - \frac{n}{2} = \frac{n^2 + n}{2} \in \Theta(n^2).$$

Laufzeitabschätzung

- Erster Gedanke: Äußere Schleife macht n-mal „irgendwas“
  $\Rightarrow\ n \cdot (???)$
  (Klammer? Wie schreiben wir das auf? Nicht n-mal dasselbe, sondern **von** i **abhängig**!)
- Rettung: Anzahl innere Schleifendurchläufe **einzeln** für jedes $i = 1, \ldots, n$ **aufsummieren**!
- Für ein festes i wird innere Schleife $(n - i + 1)$-mal durchlaufen $\Rightarrow$

Gesamtanzahl der inneren Schleifendurchläufe:

$$\sum_{i=1}^{n}(n-i+1) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 = n^2 - \sum_{i=1}^{n} i + n$$

$$= n^2 - \frac{n \cdot (n+1)}{2} + n = n^2 - \frac{n^2}{2} + n - \frac{n}{2} = \frac{n^2+n}{2} \in \Theta(n^2).$$

Laufzeitabschätzung

- Erster Gedanke: Äußere Schleife macht n-mal „irgendwas“
  $\Rightarrow\ n \cdot (???)$
  (Klammer? Wie schreiben wir das auf? Nicht n-mal dasselbe, sondern **von** i **abhängig**!)
- Rettung: Anzahl innere Schleifendurchläufe **einzeln** für jedes $i = 1, \ldots, n$ **aufsummieren**!
- Für ein festes i wird innere Schleife $(n - i + 1)$-mal durchlaufen $\Rightarrow$ Gesamtanzahl der inneren Schleifendurchläufe:

$$\sum_{i=1}^{n}(n - i + 1) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 = n^2 - \sum_{i=1}^{n} i + n$$

$$= n^2 - \frac{n \cdot (n+1)}{2} + n = n^2 - \frac{n^2}{2} + n - \frac{n}{2} = \frac{n^2 + n}{2} \in \Theta(n^2).$$

Laufzeitabschätzung

- Erster Gedanke: Äußere Schleife macht n-mal „irgendwas“
  $\Rightarrow\ n \cdot (???)$
  (Klammer? Wie schreiben wir das auf? Nicht n-mal dasselbe, sondern **von** i **abhängig**!)
- Rettung: Anzahl innere Schleifendurchläufe **einzeln** für jedes $i = 1, \ldots, n$ **aufsummieren**!
- Für ein festes i wird innere Schleife $(n - i + 1)$-mal durchlaufen $\Rightarrow$ Gesamtanzahl der inneren Schleifendurchläufe:

$$\sum_{i=1}^{n}(n - i + 1) = \sum_{i=1}^{n} n - \sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 = n^2 - \sum_{i=1}^{n} i + n$$

$$= n^2 - \frac{n \cdot (n+1)}{2} + n = n^2 - \frac{n^2}{2} + n - \frac{n}{2} = \frac{n^2 + n}{2} \in \Theta(n^2).$$

Laufzeitabschätzung

Laufzeit?

```
function going(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        if i > n − 4 then
            for j := i to n do
                k++
    return k + ℓ
```

Die innere Schleife wird wieder **nur max. vier Mal** erreicht (s. *boing*) (nämlich für $i \in \{n-3, n-2, n-1, n\}$).

$\Rightarrow$ Die innere Schleife wird erst vier-, dann drei-, dann zwei- und dann einmal durchlaufen.

$\Rightarrow$ Laufzeit in $\Theta(n+4+3+2+1) = \Theta(n)$

Laufzeitabschätzung

Laufzeit?

```
function going(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        if i > n − 4 then
            for j := i to n do
                k++
    return k + ℓ
```

Die innere Schleife wird wieder **nur max. vier Mal** erreicht (s. *boing*) (nämlich für $i \in \{n-3, n-2, n-1, n\}$).

$\Rightarrow$ Die innere Schleife wird erst vier-, dann drei-, dann zwei- und dann einmal durchlaufen

$\Rightarrow$ Laufzeit in $\Theta(n+4+3+2+1) = \Theta(n)$

Laufzeitabschätzung

Laufzeit?

```
function going(n : ℕ) : ℕ
    k := 0
    ℓ := 0
    for i := 1 to n do
        ℓ++
        if i > n − 4 then
            for j := i to n do
                k++
    return k + ℓ
```

Die innere Schleife wird wieder **nur max. vier Mal** erreicht (s. *boing*) (nämlich für $i \in \{n-3, n-2, n-1, n\}$).

$\Rightarrow$ Die innere Schleife wird erst vier-, dann drei-, dann zwei- und dann einmal durchlaufen

$\Rightarrow$ Laufzeit in $\Theta(n + 4 + 3 + 2 + 1) = \Theta(n)$

Laufzeitabschätzung

Hinweise für Aufgaben

- „Obere Schranke“ gefordert
  $\Rightarrow$ $O(f(n))$ (potenziell „zu große“ Schranke) ausreichend
- „Scharfe asymptotische Schranke“ gefordert
  $\Rightarrow$ $\Theta(f(n))$ benötigt
- Laufzeit eines Algorithmus soll angegeben bzw. bestimmt werden
  $\Rightarrow$ Offiziell $\Theta(f(n))$ erwünscht
  (in VL oder Musterlösungen aber oft auch $O(f(n))$)

Laufzeitabschätzung

Hinweise für Aufgaben

- „Obere Schranke“ gefordert
  $\Rightarrow$ $O(f(n))$ (potenziell „zu große“ Schranke) ausreichend
- „Scharfe asymptotische Schranke“ gefordert
  $\Rightarrow$ $\Theta(f(n))$ benötigt
- Laufzeit eines Algorithmus soll angegeben bzw. bestimmt werden
  $\Rightarrow$ Offiziell $\Theta(f(n))$ erwünscht
  (in VL oder Musterlösungen aber oft auch $O(f(n))$)

Laufzeitabschätzung

Hinweise für Aufgaben

- „Obere Schranke“ gefordert
  $\Rightarrow$ $O(f(n))$ (potenziell „zu große“ Schranke) ausreichend
- „**Scharfe** asymptotische Schranke“ gefordert
  $\Rightarrow$ $\Theta(f(n))$ benötigt
- Laufzeit eines Algorithmus soll angegeben bzw. bestimmt werden
  $\Rightarrow$ Offiziell $\Theta(f(n))$ erwünscht
  (in VL oder Musterlösungen aber oft auch $O(f(n))$)

Laufzeitabschätzung

Hinweise für Aufgaben

- „Obere Schranke“ gefordert
  $\Rightarrow$ $O(f(n))$ (potenziell „zu große“ Schranke) ausreichend
- „**Scharfe** asymptotische Schranke“ gefordert
  $\Rightarrow$ $\Theta(f(n))$ benötigt
- Laufzeit eines Algorithmus soll angegeben bzw. bestimmt werden
  $\Rightarrow$ Offiziell $\Theta(f(n))$ erwünscht
  (in VL oder Musterlösungen aber oft auch $O(f(n))$)

Das Master-Theorem

Das Thema des Bachelors

Das Master-Theorem (einfache Form)

Seien a, b, c, d positive Konstanten und für $n \in \mathbb{N}$ sei

$$T(n) = \begin{cases} a, & \text{für } n = 1 \\ d \cdot T(\frac{n}{b}) + c \cdot n, & \text{für } n > 1 \end{cases}$$

gegeben.

Dann gilt:

$$T(n) \in \begin{cases} \Theta(n), & d < b \\ \Theta(n \log n), & d = b \\ \Theta\left(n^{\log_b d}\right), & d > b \end{cases}.$$

Master-Theorem: Beispiele

Wir betrachten verschiedene Sortierverfahren:

Mergesort

method Mergesort(L : List **with** $|L| = n$)
- teile L in der Mitte auf in L_1 und L_2
- sortiere L_1 und L_2 rekursiv mit Mergesort
- füge L_1 und L_2 in $\Theta(n)$ zusammen

Laufzeit: $T(n) = \begin{cases} 1, & n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + 1 \cdot n, & n > 1 \end{cases}$

Nach MT (Fall 2) also: $T(n) \in \Theta(n \log n)$.

Master-Theorem: Beispiele

Wir betrachten verschiedene Sortierverfahren:

Mergesort

method Mergesort(L : List **with** $|L| = n$)
- teile L in der Mitte auf in L_1 und L_2
- sortiere L_1 und L_2 rekursiv mit Mergesort
- füge L_1 und L_2 in $\Theta(n)$ zusammen

$$\text{Laufzeit:} \quad T(n) = \begin{cases} 1, & n = 1 \\ 2 \cdot T(\frac{n}{2}) + 1 \cdot n, & n > 1 \end{cases}$$

Nach MT (Fall 2) also: $T(n) \in \Theta(n \log n)$.

Master-Theorem: Beispiele

Wir betrachten verschiedene Sortierverfahren:

Mergesort

method Mergesort(L : List **with** $|L| = n$)
- teile L in der Mitte auf in L_1 und L_2
- sortiere L_1 und L_2 rekursiv mit Mergesort
- füge L_1 und L_2 in $\Theta(n)$ zusammen

$$\text{Laufzeit:} \quad T(n) = \begin{cases} 1, & n = 1 \\ 2 \cdot T(\frac{n}{2}) + 1 \cdot n, & n > 1 \end{cases}$$

Nach MT (Fall 2) also: $T(n) \in \Theta(n \log n)$.

Master-Theorem: Beispiele

Wir betrachten verschiedene Sortierverfahren:

DoubleMergesort

method DoubleMergesort(L : List **with** $|L| = n$)
- teile L in der Mitte auf in L_1 und L_2
- sortiere L_1 und L_2 **jeweils zwei Mal** rekursiv mit DoubleMergesort
  - *// Ja, das ist natürlich konstruierter Blödsinn!*
- füge L_1 und L_2 in $\Theta(n)$ zusammen

Laufzeit: $T(n) = \begin{cases} 1, & n = 1 \\ 4 \cdot T\left(\frac{n}{2}\right) + 1 \cdot n, & n > 1 \end{cases}$

Nach MT (Fall 3) also: $T(n) \in \Theta\left(n^{\log_2 4}\right) = \Theta\left(n^2\right)$.

Master-Theorem: Beispiele

Wir betrachten verschiedene Sortierverfahren:

DoubleMergesort

method DoubleMergesort(L : List **with** $|L| = n$)
- teile L in der Mitte auf in L_1 und L_2
- sortiere L_1 und L_2 **jeweils zwei Mal** rekursiv mit DoubleMergesort
  - *// Ja, das ist natürlich konstruierter Blödsinn!*
- füge L_1 und L_2 in $\Theta(n)$ zusammen

$$\text{Laufzeit:} \quad T(n) = \begin{cases} 1, & n = 1 \\ 4 \cdot T\left(\frac{n}{2}\right) + 1 \cdot n, & n > 1 \end{cases}$$

Nach MT (Fall 3) also: $T(n) \in \Theta\left(n^{\log_2 4}\right) = \Theta\left(n^2\right)$.

Master-Theorem: Beispiele

Wir betrachten verschiedene Sortierverfahren:

DoubleMergesort

method DoubleMergesort(L : List **with** $|L| = n$)
- teile L in der Mitte auf in L_1 und L_2
- sortiere L_1 und L_2 **jeweils zwei Mal** rekursiv mit DoubleMergesort
  - *// Ja, das ist natürlich konstruierter Blödsinn!*
- füge L_1 und L_2 in $\Theta(n)$ zusammen

$$\text{Laufzeit:} \quad T(n) = \begin{cases} 1, & n = 1 \\ 4 \cdot T\left(\frac{n}{2}\right) + 1 \cdot n, & n > 1 \end{cases}$$

Nach MT (Fall 3) also: $T(n) \in \Theta\left(n^{\log_2 4}\right) = \Theta\left(n^2\right)$.

Master-Theorem: Beispiele

Wir betrachten verschiedene Sortierverfahren:

Magicsort

method Magicsort(L : List **with** $|L| = n$)
- teile L in der Mitte auf in L_1 und L_2
- sortiere L_1 rekursiv mit Magicsort
- sortiere L_2 mithilfe eines Flaschengeistes (in **Nullkommanichts**!)
- füge L_1 und L_2 in $\Theta(n)$ zusammen

Laufzeit: $T(n) = \begin{cases} 1, & n = 1 \\ 1 \cdot T(\frac{n}{2}) + 1 \cdot n, & n > 1 \end{cases}$

Nach MT (Fall 1) also: $T(n) \in \Theta(n)$.

Master-Theorem: Beispiele

Wir betrachten verschiedene Sortierverfahren:

Magicsort

method Magicsort(L : List **with** $|L| = n$)
- teile L in der Mitte auf in L_1 und L_2
- sortiere L_1 rekursiv mit Magicsort
- sortiere L_2 mithilfe eines Flaschengeistes (in **Nullkommanichts**!)
- füge L_1 und L_2 in $\Theta(n)$ zusammen

$$\text{Laufzeit:} \quad T(n) = \begin{cases} 1, & n = 1 \\ 1 \cdot T(\frac{n}{2}) + 1 \cdot n, & n > 1 \end{cases}$$

Nach MT (Fall 1) also: $T(n) \in \Theta(n)$.

Master-Theorem: Beispiele

Wir betrachten verschiedene Sortierverfahren:

Magicsort

method Magicsort(L : List **with** $|L| = n$)
- teile L in der Mitte auf in L_1 und L_2
- sortiere L_1 rekursiv mit Magicsort
- sortiere L_2 mithilfe eines Flaschengeistes (in **Nullkommanichts**!)
- füge L_1 und L_2 in $\Theta(n)$ zusammen

$$\text{Laufzeit:} \quad T(n) = \begin{cases} 1, & n = 1 \\ 1 \cdot T(\frac{n}{2}) + 1 \cdot n, & n > 1 \end{cases}$$

Nach MT (Fall 1) also: $T(n) \in \Theta(n)$.

Aufgaben Master-Theorem

Aufgabe 1: Master-Theorem

Binäre Suche

function BinarySearch(A : **sorted array**$[a..b]$ **of** Elem, e : Elem) : Elem

- **if** $a = b$ **then**
  - **return** $\begin{cases} A[a], & \text{falls } A[a] = e \\ \bot, & \text{falls } A[a] \neq e \end{cases}$
- **else**
  - $m := \left\lfloor \frac{a+b}{2} \right\rfloor$
  - **if** $e \leqslant A[m]$ **then**
    - **return** BinarySearch($A[a \ldots m]$, e) // A wird ***nicht*** kopiert!
  - **else**
    - **return** BinarySearch($A[m+1 \ldots b]$, e) // A wird ***nicht*** kopiert!

Erinnerung Master-Theorem:

$$T(n) = \begin{cases} a, & n = 1 \\ d \cdot T(\frac{n}{b}) + c \cdot n, & n > 1 \end{cases}$$

$$\Rightarrow T(n) \in \begin{cases} \Theta(n), & d < b \\ \Theta(n \log n), & d = b. \\ \Theta\left(n^{\log_b d}\right), & d > b \end{cases}$$

Ermittelt die Laufzeit von BinarySearch.

Aufgaben Master-Theorem

Lösung zu Aufgabe 1

Laufzeit von BinarySearch ist $T(n) = \begin{cases} 1, & n = 1 \\ 1 \cdot T(\frac{n}{2}) + 1, & n > 1 \end{cases}$.

$\Rightarrow$ Master-Theorem sagt :

$\Rightarrow$ **Reingefallen!** :P Master-Theorem will noch „$+ c \cdot n$“. **Gibt's hier nicht!**

Hirn einschalten: BinarySearch **halbiert das Array** in jedem Durchlauf.

Wie oft kann man $n := |A|$ halbieren?

$\Rightarrow (\log_2 n)$-mal.

$\Rightarrow T(n) \in \Theta(\log n)$.

Aufgaben Master-Theorem

Lösung zu Aufgabe 1

Laufzeit von BinarySearch ist $T(n) = \begin{cases} 1, & n = 1 \\ 1 \cdot T(\frac{n}{2}) + 1, & n > 1 \end{cases}$.

$\Rightarrow$ Master-Theorem sagt $T(n) \in \Theta(n)$.

$\Rightarrow$ **Reingefallen!** :P Master-Theorem will noch „$+ c \cdot n$“. **Gibt's hier nicht!**

Hirn einschalten: BinarySearch **halbiert das Array** in jedem Durchlauf.

Wie oft kann man $n := |A|$ halbieren?

$\Rightarrow (\log_2 n)$-mal.

$\Rightarrow T(n) \in \Theta(\log n)$.

Aufgaben Master-Theorem

Lösung zu Aufgabe 1

Laufzeit von BinarySearch ist $T(n) = \begin{cases} 1, & n = 1 \\ 1 \cdot T(\frac{n}{2}) + 1, & n > 1 \end{cases}$.

$\Rightarrow$ Master-Theorem sagt ~~$T(n) \in \Theta(n)$~~.

$\Rightarrow$ **Reingefallen**! :P Master-Theorem will noch „$+ c \cdot n$“. **Gibt's hier nicht**!

Hirn einschalten: BinarySearch **halbiert das Array** in jedem Durchlauf.

Wie oft kann man $n := |A|$ halbieren?

$\Rightarrow (\log_2 n)$-mal.

$\Rightarrow T(n) \in \Theta(\log n)$.

Aufgaben Master-Theorem

Lösung zu Aufgabe 1

Laufzeit von BinarySearch ist $T(n) = \begin{cases} 1, & n = 1 \\ 1 \cdot T(\frac{n}{2}) + 1, & n > 1 \end{cases}$.

$\Rightarrow$ Master-Theorem sagt ~~$T(n) \in \Theta(n)$~~.

$\Rightarrow$ **Reingefallen**! :P Master-Theorem will noch „$+ c \cdot n$“. **Gibt's hier nicht**!

Hirn einschalten: BinarySearch **halbiert das Array** in jedem Durchlauf.
Wie oft kann man $n := |A|$ halbieren?

$\Rightarrow (\log_2 n)$-mal.

$\Rightarrow T(n) \in \Theta(\log n)$.

Aufgaben Master-Theorem

Lösung zu Aufgabe 1

Laufzeit von BinarySearch ist $T(n) = \begin{cases} 1, & n = 1 \\ 1 \cdot T(\frac{n}{2}) + 1, & n > 1 \end{cases}$.

$\Rightarrow$ Master-Theorem sagt ~~$T(n) \in \Theta(n)$~~.

$\Rightarrow$ **Reingefallen**! :P Master-Theorem will noch „$+ c \cdot n$“. **Gibt's hier nicht**!

Hirn einschalten: BinarySearch **halbiert das Array** in jedem Durchlauf.
Wie oft kann man $n := |A|$ halbieren?

$\Rightarrow$ $(\log_2 n)$-mal.

$\Rightarrow$ $T(n) \in \Theta(\log n)$.

Binäre Suche: Exkurs

Binäre Suche: Generelles Prinzip

- Grundmenge muss **sortiert** sein!
- Generelles Suchprinzip:
  Findet nicht nur Elemente in Arrays, sondern auch
  $\Rightarrow$ Nullstellen/Inverse monotoner Funktionen (falls vorhanden)
  $\Rightarrow$ Größte Arbeitslast eines Mehrkernsystems, die noch sinnvoll tragbar ist
  $\Rightarrow$ ...

Binäre Suche: Exkurs

Binäre Suche: Generelles Prinzip

- Grundmenge muss **sortiert** sein!
- Generelles Suchprinzip:
  Findet nicht nur Elemente in Arrays, sondern auch

$\Rightarrow$ Nullstellen/Inverse monotoner Funktionen (falls vorhanden)

$\Rightarrow$ Größte Arbeitslast eines Mehrkernsystems, die noch sinnvoll tragbar ist

$\Rightarrow$ ...

Aufgaben Master-Theorem

Aufgabe 2: Master-Sandwich

Die Laufzeit eines Algorithmus A wird beschrieben durch

$$T(n) = \begin{cases} 17, & n = 1 \\ 3 \cdot T\left(\left\lceil \frac{n}{5} \right\rceil\right) + 2n + 1, & n > 1 \end{cases}.$$

Berechnet die Laufzeit von A mit dem Master-Theorem.

Erinnerung Master-Theorem:

$$T(n) = \begin{cases} a, & n = 1 \\ d \cdot T\left(\frac{n}{b}\right) + c \cdot n, & n > 1 \end{cases}$$

$$\Rightarrow T(n) \in \begin{cases} \Theta(n), & d < b \\ \Theta(n \log n), & d = b \\ \Theta\left(n^{\log_b d}\right), & d > b \end{cases}.$$

Master-Sandwich

Lösung zu Aufgabe 2

Definiere

$$T_-(n) := \begin{cases} 17, & n = 1 \\ 3 \cdot T_-(\lceil \frac{n}{5} \rceil) + 2n, & n > 1 \end{cases},$$

$$T_+(n) := \begin{cases} 17, & n = 1 \\ 3 \cdot T_+(\lceil \frac{n}{5} \rceil) + 3n, & n > 1 \end{cases}.$$

Dann gilt $\forall n \geqslant 1 : \quad T_-(n) \leqslant T(n) \leqslant T_+(n)$.
Für T_- und T_+ gilt jeweils nach MT (Fall 1): $\quad T_+(n), T_-(n) \in \Theta(n)$.
Damit liegt auch $T(n) \in \Theta(n)$. $\square$

Aufgaben Master-Theorem

Aufgabe 3: Master-Theorem

Die Laufzeit eines Algorithmus A wird beschrieben durch

$$U(n) = \begin{cases} 1, & n = 1 \\ 7 \cdot U(\lceil \frac{n}{2} \rceil) + n, & n > 1 \end{cases}.$$

Ein weiterer Algorithmus B hat die Laufzeit

$$V(n) = \begin{cases} 1, & n = 1 \\ a \cdot V(\lceil \frac{n}{4} \rceil) + 5n, & n > 1 \end{cases}.$$

Was ist der größte Wert $a \in \mathbb{N}$, so dass B asymptotisch schneller als A ist?

Erinnerung Master-Theorem:

$$T(n) = \begin{cases} a, & n = 1 \\ d \cdot T(\frac{n}{b}) + c \cdot n, & n > 1 \end{cases}$$

$$\Rightarrow T(n) \in \begin{cases} \Theta(n), & d < b \\ \Theta(n \log n), & d = b. \\ \Theta\left(n^{\log_b d}\right), & d > b \end{cases}$$

Aufgaben Master-Theorem

Lösung zu Aufgabe 3

- Master-Theorem: Algorithmus A hat Laufzeit $\Theta(n^{\log_2 7})$, wächst also stärker als n^2
- Fall $a \leqslant 4$ also uninteressant $\Rightarrow a > 4$, d. h. Algorithmus B läuft in $\Theta(n^{\log_4 a})$
- Also:

$$\log_4 a < \log_2 7 \Leftrightarrow \underbrace{4^{\log_4 a}}_{=a} < 4^{\log_2 7} = 7^{\log_2 4} = 7^2 = 49$$

$$\Longrightarrow a = 48.$$

Aufgaben Master-Theorem

Lösung zu Aufgabe 3

- Master-Theorem: Algorithmus A hat Laufzeit $\Theta(n^{\log_2 7})$, wächst also stärker als n^2
- Fall $a \leqslant 4$ also uninteressant $\Rightarrow$ $a > 4$, d. h. Algorithmus B läuft in $\Theta(n^{\log_4 a})$
- Also:

$$\log_4 a < \log_2 7 \Leftrightarrow \underbrace{4^{\log_4 a}}_{=a} < 4^{\log_2 7} = 7^{\log_2 4} = 7^2 = 49$$

$$\Longrightarrow \quad a = 48.$$

Aufgaben Master-Theorem

Lösung zu Aufgabe 3

- Master-Theorem: Algorithmus A hat Laufzeit $\Theta(n^{\log_2 7})$, wächst also stärker als n^2
- Fall $a \leqslant 4$ also uninteressant $\Rightarrow$ $a > 4$, d. h. Algorithmus B läuft in $\Theta(n^{\log_4 a})$
- Also:

$$\log_4 a < \log_2 7 \Leftrightarrow \underbrace{4^{\log_4 a}}_{=a} < 4^{\log_2 7} = 7^{\log_2 4} = 7^2 = 49$$

$$\Longrightarrow a = 48.$$

Aufgaben Master-Theorem

Lösung zu Aufgabe 3

- Master-Theorem: Algorithmus A hat Laufzeit $\Theta(n^{\log_2 7})$, wächst also stärker als n^2
- Fall $a \leqslant 4$ also uninteressant $\Rightarrow$ $a > 4$, d. h. Algorithmus B läuft in $\Theta(n^{\log_4 a})$
- Also:

$$\log_4 a < \log_2 7 \Leftrightarrow \underbrace{4^{\log_4 a}}_{=a} < 4^{\log_2 7} = 7^{\log_2 4} = 7^2 = 49$$

$$\Longrightarrow \quad a = 48.$$

Rekurrenzen lösen

Aufgabe 4: Rekurrenzen lösen

Gegeben sei folgende Rekurrenz für $n = 4^k, k \in \mathbb{N}_0$

$$T(n) = \begin{cases} 2, & \text{für } n = 1 \\ 2 \cdot T(\frac{n}{4}), & \text{für } n > 1 \end{cases}$$

Findet eine Funktion $f : \mathbb{N} \longrightarrow \mathbb{R}^+$ und Konstanten c_1, c_2, so dass $c_1 \cdot f(n) \leqslant T(n) \leqslant c_2 \cdot f(n)$ und beweist euren Fund.

Tipp: Vorstellung eines „Berechnungsbaums“: Auf jeder Ebene doppelte Anzahl der Probleme mit geviertelter Problemgröße.

$\Rightarrow$ Anzahl Blätter: $2^{\log_4 n} = n^{\log_4 2} = n^{1/2} = \sqrt{n} = f(n)$ (nicht $T(n)$!)

Rekurrenzen lösen

Karlsruher Institut für Technologie

Aufgabe 4: Rekurrenzen lösen

Gegeben sei folgende Rekurrenz für $n = 4^k, k \in \mathbb{N}_0$

$$T(n) = \begin{cases} 2, & \text{für } n = 1 \\ 2 \cdot T(\frac{n}{4}), & \text{für } n > 1 \end{cases}$$

Findet eine Funktion $f : \mathbb{N} \longrightarrow \mathbb{R}^+$ und Konstanten c_1, c_2, so dass $c_1 \cdot f(n) \leqslant T(n) \leqslant c_2 \cdot f(n)$ und beweist euren Fund.

Tipp: Vorstellung eines „Berechnungsbaums“: Auf jeder Ebene doppelte Anzahl der Probleme mit geviertelter Problemgröße.

$\Rightarrow$ Anzahl Blätter: $2^{\log_4 n} = n^{\log_4 2} = n^{1/2} = \sqrt{n} = f(n)$ (nicht $T(n)$!)

Rekurrenzen lösen

Aufgabe 4: Rekurrenzen lösen

Gegeben sei folgende Rekurrenz für $n = 4^k, k \in \mathbb{N}_0$

$$T(n) = \begin{cases} 2, & \text{für } n = 1 \\ 2 \cdot T(\frac{n}{4}), & \text{für } n > 1 \end{cases}$$

Findet eine Funktion $f : \mathbb{N} \longrightarrow \mathbb{R}^+$ und Konstanten c_1, c_2, so dass $c_1 \cdot f(n) \leqslant T(n) \leqslant c_2 \cdot f(n)$ und beweist euren Fund.

Tipp: Vorstellung eines „Berechnungsbaums“: Auf jeder Ebene doppelte Anzahl der Probleme mit geviertelter Problemgröße.
$\Rightarrow$ Anzahl Blätter: $2^{\log_4 n} = n^{\log_4 2} = n^{1/2} = \sqrt{n} = f(n)$ (nicht $T(n)$!)

Rekurrenzen lösen

Lösung zu Aufgabe 4

Behauptung (Magie! ☺): $c_1 := 1$, $c_2 := 3$ und $f(n) := \sqrt{n}$
erfüllen die Bedingung $\forall n = 4^k, k \in \mathbb{N}_0$

Beweis durch vollständige Induktion über k:

IA. $(k = 0 \Rightarrow n = 4^0 = 1)$:

$$1 \cdot \sqrt{1} = 1 \leqslant T(1) = 2 \leqslant 3 \cdot \sqrt{1} = 3$$

IV.: Für ein beliebiges, aber festes $k \in \mathbb{N}_0$ $(n = 4^k)$ gelte

$$1 \cdot \sqrt{4^k} \leqslant T(n) \leqslant 3 \cdot \sqrt{4^k} \quad \left(\Leftrightarrow 1 \cdot \sqrt{n} \leqslant T(n) \leqslant 3 \cdot \sqrt{n}\right)$$

IS. $(k \rightsquigarrow k+1)$: Es gilt:

$$T\left(4^{k+1}\right) \overset{\text{Def.}}{=} 2 \cdot T\left(\frac{4^{k+1}}{4}\right) = 2 \cdot T(n)$$

$$2 \cdot T(n) \overset{\text{IV}}{\geqslant} 2 \cdot 1 \cdot \sqrt{n} = \sqrt{4} \cdot \sqrt{n} = \sqrt{4n} = \sqrt{4^{k+1}}$$

$$2 \cdot T(n) \overset{\text{IV}}{\leqslant} 2 \cdot 3 \cdot \sqrt{n} = 3 \cdot \sqrt{4} \cdot \sqrt{n} = 3 \cdot \sqrt{4n} = 3 \cdot \sqrt{4^{k+1}}; \quad \square$$

Rekurrenzen lösen

Lösung zu Aufgabe 4

Behauptung (Magie! ☺): $c_1 := 1$, $c_2 := 3$ und $f(n) := \sqrt{n}$ erfüllen die Bedingung $\forall n = 4^k, k \in \mathbb{N}_0$

Beweis durch vollständige Induktion über k:

IA. $(k = 0 \Rightarrow n = 4^0 = 1)$:

$$1 \cdot \sqrt{1} = 1 \leqslant T(1) = 2 \leqslant 3 \cdot \sqrt{1} = 3$$

IV.: Für ein beliebiges, aber festes $k \in \mathbb{N}_0$ $(n = 4^k)$ gelte

$$1 \cdot \sqrt{4^k} \leqslant T(n) \leqslant 3 \cdot \sqrt{4^k} \quad \left(\Leftrightarrow 1 \cdot \sqrt{n} \leqslant T(n) \leqslant 3 \cdot \sqrt{n}\right)$$

IS. $(k \rightsquigarrow k+1)$: Es gilt:

$$T\left(4^{k+1}\right) \overset{\text{Def.}}{=} 2 \cdot T\left(\frac{4^{k+1}}{4}\right) = 2 \cdot T(n)$$

$$2 \cdot T(n) \overset{\text{IV}}{\geqslant} 2 \cdot 1 \cdot \sqrt{n} = \sqrt{4} \cdot \sqrt{n} = \sqrt{4n} = \sqrt{4^{k+1}}$$

$$2 \cdot T(n) \overset{\text{IV}}{\leqslant} 2 \cdot 3 \cdot \sqrt{n} = 3 \cdot \sqrt{4} \cdot \sqrt{n} = 3 \cdot \sqrt{4n} = 3 \cdot \sqrt{4^{k+1}}; \quad \square$$

Rekurrenzen lösen

Lösung zu Aufgabe 4

Behauptung (Magie! ☺): $c_1 := 1$, $c_2 := 3$ und $f(n) := \sqrt{n}$ erfüllen die Bedingung $\forall n = 4^k, k \in \mathbb{N}_0$

Beweis durch vollständige Induktion über k:

IA. $(k = 0 \Rightarrow n = 4^0 = 1)$:

$$1 \cdot \sqrt{1} = 1 \leqslant T(1) = 2 \leqslant 3 \cdot \sqrt{1} = 3$$

IV.: Für ein beliebiges, aber festes $k \in \mathbb{N}_0$ $(n = 4^k)$ gelte

$$1 \cdot \sqrt{4^k} \leqslant T(n) \leqslant 3 \cdot \sqrt{4^k} \quad \left(\Leftrightarrow 1 \cdot \sqrt{n} \leqslant T(n) \leqslant 3 \cdot \sqrt{n} \right)$$

IS. $(k \rightsquigarrow k+1)$: Es gilt:

$$T\left(4^{k+1}\right) \overset{\text{Def.}}{=} 2 \cdot T\left(\frac{4^{k+1}}{4}\right) = 2 \cdot T(n)$$

$$2 \cdot T(n) \overset{\text{IV}}{\geqslant} 2 \cdot 1 \cdot \sqrt{n} = \sqrt{4} \cdot \sqrt{n} = \sqrt{4n} = \sqrt{4^{k+1}}$$

$$2 \cdot T(n) \overset{\text{IV}}{\leqslant} 2 \cdot 3 \cdot \sqrt{n} = 3 \cdot \sqrt{4} \cdot \sqrt{n} = 3 \cdot \sqrt{4n} = 3 \cdot \sqrt{4^{k+1}}. \quad \square$$

Rekurrenzen lösen

Lösung zu Aufgabe 4

Behauptung (Magie! ☺): $c_1 := 1$, $c_2 := 3$ und $f(n) := \sqrt{n}$ erfüllen die Bedingung $\forall n = 4^k, k \in \mathbb{N}_0$

Beweis durch vollständige Induktion über k:

IA. $(k = 0 \Rightarrow n = 4^0 = 1)$:

$$1 \cdot \sqrt{1} = 1 \leqslant T(1) = 2 \leqslant 3 \cdot \sqrt{1} = 3$$

IV.: Für ein beliebiges, aber festes $k \in \mathbb{N}_0$ $(n = 4^k)$ gelte

$$1 \cdot \sqrt{4^k} \leqslant T(n) \leqslant 3 \cdot \sqrt{4^k} \quad \left(\Leftrightarrow 1 \cdot \sqrt{n} \leqslant T(n) \leqslant 3 \cdot \sqrt{n} \right)$$

IS. $(k \rightsquigarrow k+1)$: Es gilt:

$$T\left(4^{k+1}\right) \overset{Def.}{=} 2 \cdot T\left(\frac{4^{k+1}}{4}\right) = 2 \cdot T(n)$$

$$2 \cdot T(n) \overset{IV}{\geqslant} 2 \cdot 1 \cdot \sqrt{n} = \sqrt{4} \cdot \sqrt{n} = \sqrt{4n} = \sqrt{4^{k+1}}$$

$$2 \cdot T(n) \overset{IV}{\leqslant} 2 \cdot 3 \cdot \sqrt{n} = 3 \cdot \sqrt{4} \cdot \sqrt{n} = 3 \cdot \sqrt{4n} = 3 \cdot \sqrt{4^{k+1}}. \quad \square$$

Rekurrenzen lösen

Lösung zu Aufgabe 4

Behauptung (Magie! ☺): $c_1 := 1$, $c_2 := 3$ und $f(n) := \sqrt{n}$ erfüllen die Bedingung $\forall n = 4^k, k \in \mathbb{N}_0$

Beweis durch vollständige Induktion über k:

IA. $(k = 0 \Rightarrow n = 4^0 = 1)$:

$$1 \cdot \sqrt{1} = 1 \leqslant T(1) = 2 \leqslant 3 \cdot \sqrt{1} = 3$$

IV.: Für ein beliebiges, aber festes $k \in \mathbb{N}_0$ $(n = 4^k)$ gelte

$$1 \cdot \sqrt{4^k} \leqslant T(n) \leqslant 3 \cdot \sqrt{4^k} \quad \left(\Leftrightarrow 1 \cdot \sqrt{n} \leqslant T(n) \leqslant 3 \cdot \sqrt{n} \right)$$

IS. $(k \rightsquigarrow k+1)$: Es gilt:

$$T\left(4^{k+1}\right) \overset{Def.}{=} 2 \cdot T\left(\frac{4^{k+1}}{4}\right) = 2 \cdot T(n)$$

$$2 \cdot T(n) \overset{IV}{\geqslant} 2 \cdot 1 \cdot \sqrt{n} = \sqrt{4} \cdot \sqrt{n} = \sqrt{4n} = \sqrt{4^{k+1}}$$

$$2 \cdot T(n) \overset{IV}{\leqslant} 2 \cdot 3 \cdot \sqrt{n} = 3 \cdot \sqrt{4} \cdot \sqrt{n} = 3 \cdot \sqrt{4n} = 3 \cdot \sqrt{4^{k+1}}. \quad \square$$

Rekurrenzen lösen

Lösung zu Aufgabe 4

Behauptung (Magie! ☺): $c_1 := 1$, $c_2 := 3$ und $f(n) := \sqrt{n}$ erfüllen die Bedingung $\forall n = 4^k, k \in \mathbb{N}_0$

Beweis durch vollständige Induktion über k:

IA. $(k = 0 \Rightarrow n = 4^0 = 1)$:

$$1 \cdot \sqrt{1} = 1 \leqslant T(1) = 2 \leqslant 3 \cdot \sqrt{1} = 3$$

IV.: Für ein beliebiges, aber festes $k \in \mathbb{N}_0$ $(n = 4^k)$ gelte

$$1 \cdot \sqrt{4^k} \leqslant T(n) \leqslant 3 \cdot \sqrt{4^k} \quad \left(\Leftrightarrow 1 \cdot \sqrt{n} \leqslant T(n) \leqslant 3 \cdot \sqrt{n} \right)$$

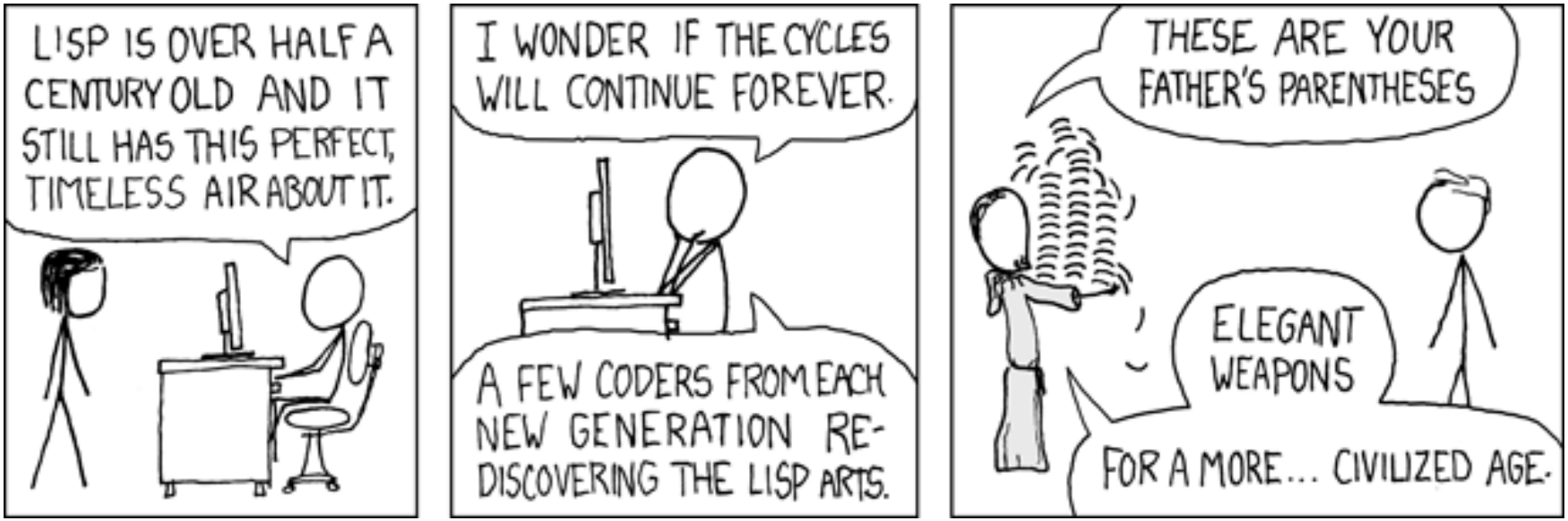
IS. $(k \rightsquigarrow k+1)$: Es gilt:

$$T\left(4^{k+1}\right) \overset{Def.}{=} 2 \cdot T\left(\frac{4^{k+1}}{4}\right) = 2 \cdot T(n)$$

$$2 \cdot T(n) \overset{IV}{\geqslant} 2 \cdot 1 \cdot \sqrt{n} = \sqrt{4} \cdot \sqrt{n} = \sqrt{4n} = \sqrt{4^{k+1}}$$

$$2 \cdot T(n) \overset{IV}{\leqslant} 2 \cdot 3 \cdot \sqrt{n} = 3 \cdot \sqrt{4} \cdot \sqrt{n} = 3 \cdot \sqrt{4n} = 3 \cdot \sqrt{4^{k+1}}. \quad \square$$

May the Fourth Be With You! ☺



http://xkcd.com/297

Credits

Vorgänger dieses Foliensatzes wurden erstellt von:

Christopher Hommel (urspr. Verfasser)
Daniel Jungkind