Tutorium Algorithmen 1

01 · Pseudocode und O-Kalkül · 22.4.2024

Peter Bohner   Tutorium 3

Über mich

Peter Bohner

- 6. Semester, Informatik Bachelor
- Bei Fragen
  - Hier im Tut
  - Auf Discord: In #fragen, oder per DM
  - Per Email: peter.bohner@student.kit.edu

Über euch

- Name
- Studiengang
- Semester
- etc.

Organisation des Tutoriums

- Jeden Montag 17:30 - 19:00 hier (50.34(Infobau) Raum -120)
- Inhalte der Vorlesung wiederholen und anwenden
- Beispiele und Aufgaben
- Eure Fragen klären
- Das Tutorium ist kein Ersatz für die Vorlesung, geht da hin, die ist echt gut

Organisation

- Klausurtermin: 30.08.2024

Übungsblätter

- Die Übungsblätter sind **Frewillig** (aber helfen euch natürlich den Stoff zu verstehen)
- Veröffentlichung: Jeweils Mittwochs
- Abgabe: 9 Tage später, am Freitag
  - (maximal) Zu Zweit
  - Im Ilias als PDF (Eingescannt oder Digital)
- Bonuspunkte für die Klausur
  - Ab 50% gibt es einen Notenschritt Bonus auf die Klausur
  - Aber nur auf bestandene Klausuren

Alle wichtigen Infos gibt es auch auf der Homepage:

`https://scale.iti.kit.edu/teaching/2024ss/algo1/start`

Hinweise zu den Übungsblättern

- Sucht euch einen Partner für die Blätter, das hilf euch und auch mir bei der Korrektur
- Abgabe von einer Person aus der Gruppe, diese Person bekommt auch die Rückmeldung

Achtung

- Benutzt nur Pseudocode wenn das in der Aufgabe explizit gefordert ist
- Wenn in der Aufgabe steht „Beschreibt“ dann heißt das **in Worten** und **nicht** in Pseudocode
- Wenn ihr Pseudocode benutzt wenn ihr beschreiben sollt oder umgekehrt gibt es 0 Punkte

Was sind Algorithmen?

- endliche, präzise Folge von Anweisungen
- endliche Eingabe
- generiert endliche Ausgabe
- „hält“ bei jeder (erlaubten) Eingabe
- unabhängig von Implementierung/Sprache/OS/Hardware
- i.d.R. hängen Algorithmen mit bestimmten Datenstrukturen zusammen

Inhalt von Algorithmen 1

- Wir wollen eigene Algorithmen konstruieren
- Wie beschreibt man Algorithmen? (1. Thema heute)
- Wie bewertet man Algorithmen? (2. Thema heute)
  - Funktionalität (beweisen/testen) hier weniger wichtig
  - hier: Laufzeit/Speicherbedarf im O-Kalkül
- VL stellt nur Bausteine/wichtige Konzepte vor

$\Rightarrow$ verstehen & anwenden wichtig!

Peter Bohner - peter.bohner@student.kit.edu

Algorithmen beschreiben

gegeben: Beschreibung einer gültigen Lösung

Lösungsidee als Fließtext

Pseudocode

Programmiersprachen

Assemblersprache

Schaltwerke

abstrakter

wichtig für das Verständnis

- hat explizit keine formalen Anforderungen

Zu konkret für unsere Zwecke

- Ändert sich alle paar Jahre
- unsere Algorithmen sind z.T. deutlich älter und immer noch wichtig

Pseudocode

- Fünf wichtige Einzelteile:
  - Zuweisungen ($\leftarrow$ oder $:=$)
  - Schleifen/Bedingungen
  - **beliebige** mathematische Ausdrücke
  - Variablen/Typen/Datenstrukturen/Funktionen
  - Kommentare!

Beispiel Zuweisungen:

$b \leftarrow 5$		
$c := 3$		
$a \leftarrow b$		
$b := c$	Welchen Wert haben a, b?	$a = 5 \wedge b = 3$
$c, a := a, b + c$	Welchen Wert haben a, c?	$a = 6 \wedge c = 5$
$(a, c) \leftarrow (c, a)$	Welchen Wert haben a, c?	$a = 5 \wedge c = 6$

Schleifen/Bedingungen

- Bedingungen:

  if $a > b$ **then**

  // do something

  else

  // do something else

- Schleifen:

  while Bedingung **do**

  // do something

- for-Schleifen:

  for i ∈ M **do** // $|M| < \infty$

  // do something

- **od** oder **end for** bzw. **end while**
- **end** geht immer
- sinnvoll einrücken geht auch (falls lesbar)

Elemente in M müssen feste Reihenfolge haben
⇒falls unklar explizit angeben

for i **from** 1 **to** n geht auch

Beispiel

Ziel: Algorithmus der die Summe eines Arrays aus ganzzahlen berechnet

<u>**sum**(A : array[1...n] of $\mathbb{N}_0$):</u>

```
s := 0
for i := 1 to do
    s := s + A[i]
return s
```


Peter Bohner - peter.bohner@student.kit.edu

Aufgabe

Karlsruher Institut für Technologie

Gebt einen Algorithmus in Pseudocode an, der:

- n Studenten in zwei Gruppen einteilt,
- je nachdem ob die Quersumme der Matrikelnummer, gerade oder ungerade ist
- Die zwei Gruppen in einem Tupel ausgibt

Hinweise:

- Die Eingabe ist ein Array der Matrikelnummern
- An ein dynamisches Array kann mit **pushBack** ein neues Element angefügt werden
- ihr könnt ein neues Array mit **new** DynamicArray ertellen

Peter Bohner - peter.bohner@student.kit.edu

Lösung

```
splitEvenOdd(A):
    evenSet = new Dynamic Array
    oddSet = new Dynamic Array
    for a ∈ A do
        s ← sum(a)
        if s % 2 = 0 then
            evenSet.pushBack(a)
        else
            oddSet.pushBack(a)
    return (evenSet, oddSet)
```

Sonstiges

- Funktionsparameter explizit angeben
- Kommentare sind Teil des Algorithmus!
- Pseudocode hat viele Formen
- Folien sind nur Beispiele
- Beachtet auch die Pseudocode-Richtlinien

Landau-Symbole

- grobe Abschätzung von Laufzeit/Speicherbedarf
- Aussagen über große Eingaben
- unabhängig von konkreten Rechner/OS/usw. (wie Pseudocode)
- ignoriert Konstanten/konst. Faktoren

Wiederholung GBI:

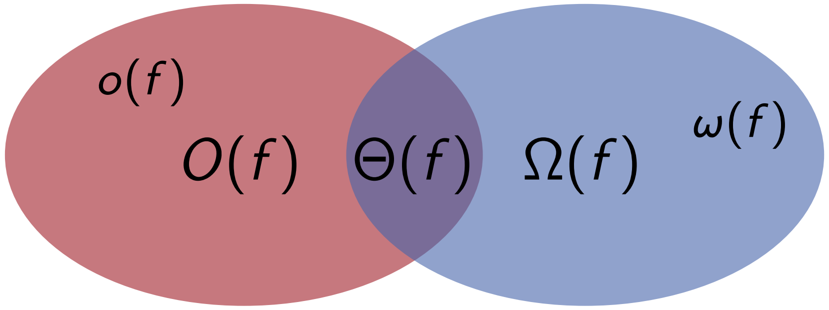
- $O(f) = \{g \mid g$ wächst asymptotisch nicht schneller als $f\}$
- $\Omega(f) = \{g \mid g$ wächst asymptotisch nicht langsamer als $f\}$
- $\Theta(f) = O(f) \cap \Omega(f) = \{g \mid g$ wächst asymptotisch genauso schnell wie $f\}$

Peter Bohner - peter.bohner@student.kit.edu

Landau Symbole

Jetzt neu:

- $o(f) = O(f) \setminus \Theta(f) = \{g \mid g$ wächst asymptotisch echt langsamer als $f\}$
- $\omega(f) = \Omega(f) \setminus \Theta(f) = \{g \mid g$ wächst asymptotisch echt schneller als $f\}$



Peter Bohner - peter.bohner@student.kit.edu

Formale Definition

„Für alle Eingaben größer als n_0..."

- $g \in O(f) \Leftrightarrow \exists c > 0, n_0 \in \mathbb{N}\ \forall n > n_0 : g(n) \leq c \cdot f(n)$

„...wächst f schneller als oder gleich schnell wie g"

- $g \in \Omega(f) \Leftrightarrow \exists c > 0, n_0 \in \mathbb{N}\ \forall n > n_0 : c \cdot f(n) \leq g(n)$

„...wächst f langsamer als oder gleich schnell wie g"

- $g \in o(f) \Leftrightarrow \forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0 : g(n) < c \cdot f(n)$

„...wächst f schneller als g"

- $g \in \omega(f) \Leftrightarrow \forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall n > n_0 : c \cdot f(n) < g(n)$

„...wächst f langsamer als g"

- $g \in \Theta(f) \Leftrightarrow \exists c_1, c_2 > 0, n_0 \in \mathbb{N}\ \forall n > n_0 : c_1 \cdot f(n) \geq g(n) \geq c_2 \cdot f(n)$

„...wächst f gleich schnell wie g"

Rechenregeln

- $O(f+k) = O(f), k \in \mathbb{R}$
- $O(c \cdot f) = O(f), c \in \mathbb{R}_+$
- $O(f), O(g) \subseteq O(f+g)$
- $g \in O(f) \Leftrightarrow O(g) \subseteq O(f)$
- $g \in \Omega(f) \Leftrightarrow \Omega(g) \subseteq \Omega(f)$
- $g \in \Theta(f) \Leftrightarrow \Theta(g) = \Theta(f)$
- $g(O(f)) = \{g(h) \mid h \in O(f)\}$

z.B. $n^{n^2} \in n^{\Omega(n)}$

- analog für Ω, Θ, ω und o
- $p \in \Theta(n^{deg(p)})$, p Polynom
- $a < b \Leftrightarrow n^a \in o(n^b)$

Aufgabe (2 min)

Zu welcher Menge gehören folgende Funktionen:

$\log(n)$ $\quad 5n+7 \quad (4n+3)^2 \quad n^2 - n \cdot \ln(n) \quad 2n \cdot \log(3n) \quad \log(n)^2 \quad 2n + \log(3n) \quad n! \quad \log(3^n) \quad 2^n$

$o(n)$	$\Theta(n)$	$\omega(n) \cap o(n^2)$	$\Theta(n^2)$	$\omega(n^2)$
$\log(n)$	$2n + \log(3n)$	$n \cdot \log(n)$	$(4n+3)^2$	2^n
$\log(n)^2$	$5n+7$		$n^2 - n \cdot \ln(n)$	$n!$
	$\log(3^n)$			

Grenzwerte

Karlsruher Institut für Technologie

Betrachte Grenzwert $\ell = \lim\limits_{n\to\infty} \frac{f(n)}{g(n)}$

	$\ell = 0$	$\ell < \infty$	$0 < \ell < \infty$	$0 < \ell$	$\ell = \infty$
$f(n) \in$	$o(g(n))$	$O(g(n))$	$\Theta(g(n))$	$\Omega(g(n))$	$\omega(g(n))$

- Wer sich mit HM auskennt findet diese Definition wahrscheinlich angenehm
- Hier können auch Regeln wie l'Hospital angewandt werden

Ein paar elementare Funktionen

1	$\log^* n$	$\log n$	$\log^2 n$	$\sqrt[3]{n}$	$\sqrt{n}$	n	n^2	n^3	$n^{\log n}$	$2^{\sqrt{n}}$	2^n	3^n	4^n	$n!$	2^{n^2}

Aufgabe

Aufgabe (5 min)

Zeigt oder wiederlegt:

$\omega(f)$	$\Theta(f)$	$o(f)$	$l = \lim\limits_{n\to\infty} \frac{g(n)}{f(n)}$
$l = \infty$	$l \in \mathbb{R}_+$	$l = 0$	

$n + 1 \in \Theta(n)$

$(n + 1)! \in \Theta(n!)$

$n^5 + 7 \cdot n^4 \in \omega(n^5)$

$1{,}5^n \in O(2^n)$

$ln(3) \in \Theta(\sin(n) + 3)$

$log(n)^2 \in o(\frac{n}{log(n)})$

$$\lim_{n\to\infty} \frac{n+1}{n} = 1 + \lim_{n\to\infty} \frac{1}{n} = 1$$

$$\lim_{n\to\infty} \frac{(n+1)!}{n!} = \lim_{n\to\infty} n + 1 = \infty$$

$$\lim_{n\to\infty} \frac{n^5 + 7 \cdot n^4}{n^5} = 1 + \lim_{n\to\infty} \frac{7 \cdot n^4}{n^5} = 1$$

$$\lim_{n\to\infty} \frac{1{,}5^n}{2^n} = \lim_{n\to\infty} (\frac{3}{4})^n = 0$$

Aufgabe

Aufgabe (5 min)

Zeigt oder wiederlegt:

$n + 1 \in \Theta(n)$
$(n+1)! \in \Theta(n!)$
$n^5 + 7 \cdot n^4 \in \omega(n^5)$
$1{,}5^n \in O(2^n)$
$ln(3) \in \Theta(\sin(n) + 3)$
$log(n)^2 \in o(\frac{n}{log(n)})$

$\omega(f)$	$\Theta(f)$	$o(f)$	$l = \lim\limits_{n\to\infty} \frac{g(n)}{f(n)}$
$l = \infty$	$l \in \mathbb{R}_+$	$l = 0$	

$\lim\limits_{n\to\infty} \frac{\ln(3)}{\sin(n) + 3}$ divergiert

$g \in \Theta(f) \Leftrightarrow \exists c_1, c_2 > 0, n_0 \in \mathbb{N}\ \forall n > n_0 : c_1 \cdot f(n) \geq g(n) \geq c_2 \cdot f(n)$

Wähle $c_1 = 1$, $c_2 = \frac{1}{4}$, $n_0 = 1$

$c_1 \cdot (\sin(n) + 3) \geq c_1 \cdot 2 \geq ln(3) \geq c_2 \cdot 4 \geq c_2 \cdot (\sin(n) + 3)$

$c_1 \cdot (\sin(n) + 3) \geq c_1 \cdot 2 \geq 2 \geq ln(3) \geq 1 \geq c_2 \cdot 4 \geq c_2 \cdot (\sin(n) + 3)$

$c_1 \cdot (\sin(n) + 3) \geq 1 \cdot 2 = 2 \geq ln(3) \geq 1 = \frac{1}{4} \cdot 4 \geq c_2 \cdot (\sin(n) + 3)$

$\Rightarrow ln(3) \in \Theta(\sin(n) + 3)$

$$\lim_{n\to\infty} \frac{\log(n)^3}{n} = \lim_{n\to\infty} \frac{3 \cdot \log(n)^2}{n} = \lim_{n\to\infty} \frac{6 \cdot \log(n)}{n} = \lim_{n\to\infty} \frac{6}{n} = 0$$

Analyse von Algorithmen

- Jeder Algorithmus induziert drei Abbildungen:
  - Eingabe zu Ausgabe (hier unwichtig)
  - Eingabe zu Speicherbedarf
  - Eingabe zu Arbeitsschritten (quasi Zeit)

- uns interessiert Ober/Untergrenze von Zeit/Speicher
- Laufzeit/Speicherbedarf unabhängig von Ausgabe
- oft Abwägung zwischen Speicher/Laufzeit
- worst case besonders interessant
- Speichernutzung eher abstrakt
- keine Aussage über kleine Eingaben

$\Rightarrow$ wichtiger Randfall bleibt unbeachtet

- z.T. weitere Abwägungen z.B. Erwartungswert

Peter Bohner - peter.bohner@student.kit.edu

Analyse von Pseudocode

- Laufzeit im Allgemeinen nicht entscheidbar (siehe Halteproblem)
- Addition/Multiplikation/usw. *eigentlich* nicht in O(1)
  - in der Realität meist 16/32/64bit Zahlen $\Rightarrow$ O(1)
  - in Algo 1 in O(1) (außer es steht dabei)
  - Algo aus der VL für Langzahlmultiplikation

    siehe BigInteger/BigDecimal in java
- Verzweigungen $\Rightarrow$ Laufzeiten der Zweige aufsummieren
- Hintereinander ausführen $\Rightarrow$ Laufzeiten aufsummieren
- Schleifen in $O(\text{Anzahl Durchläufe} \cdot \text{Laufzeit Schleifenrumpf})$

$\Rightarrow$es gibt z.T. schärfere obere Schranken

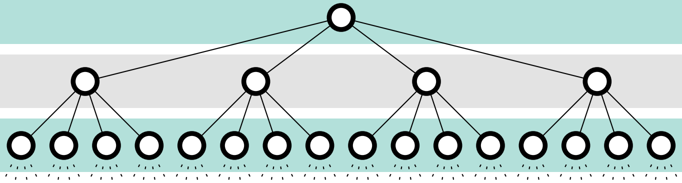
- Summen/Produkte haben gleiche Laufzeit wie äquivalente Schleifen
- Mastertheorem für einige rekursive Funktionen

Peter Bohner - peter.bohner@student.kit.edu

Exponentielle Summen

- $\sum_i c^i$ wird asymptotisch vom größten Summanden dominiert
- also: $\sum_{i=a}^{b} c^i \in \Theta\left(c^a + c^b\right)$

$$T(n) = \begin{cases} 1 & | n = 1 \\ 4 \cdot T(\frac{n}{2}) + n & | sonst \end{cases}$$



Lage i	Anzahl Knoten
0	4^0
1	4^1
2	4^2
$\vdots$	$\vdots$
$k-1$	4^{k-1}
$k = \log_2(n)$	4^k

Anzahl Lagen:

- $\log_2(n)$

Anzahl Knoten auf Lage i:

- 4^i

Arbeit pro Knoten auf Lage i:

- $\frac{n}{2^i}$

Arbeit pro Lage:

$$4^i \cdot \frac{n}{2^i} = n \cdot (\tfrac{4}{2})^i$$

Arbeit Gesamt:

$$\sum_{i=1}^{\log_2(n)} (n \cdot (\tfrac{4}{2})^i)$$

$$= n \cdot \sum_{i=1}^{\log_2(n)} (\tfrac{4}{2})^i \in \Theta\left(n \cdot \left(\tfrac{4}{2}\right)^{\log_2(n)}\right)$$

$$= \Theta\left(n^{\log_2(4)}\right) = \Theta\left(n^2\right)$$

Peter Bohner - peter.bohner@student.kit.edu

Mastertheorem

Karlsruher Institut für Technologie

Theorem

Sei $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ mit $f(n) \in \Theta(n^c)$ und $T(1) \in \Theta(1)$. Dann gilt

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

- T ist Gesamtlaufzeit des Algorithmus
- f ist Laufzeit pro Verzweigung

$\Rightarrow$braucht man gelegentlich (insb. für $b = 2$, $a = 1$, $c = 0$)

- $T(\frac{n}{b})$ und **nicht** $T(n - b)$

Peter Bohner - peter.bohner@student.kit.edu

Beispiel

Karlsruher Institut für Technologie

$$T(n) = \begin{cases} 1 & |n = 1 \\ 4 \cdot T(\frac{n}{2}) + n & |sonst \end{cases}$$

- $a = 4$
- $b = 2$
- $f(n) = n \quad (c = 1)$

$\Rightarrow T(n) \in \Theta\left(n^{\log_2(4)}\right) = \Theta\left(n^2\right)$

Zusammenfassung

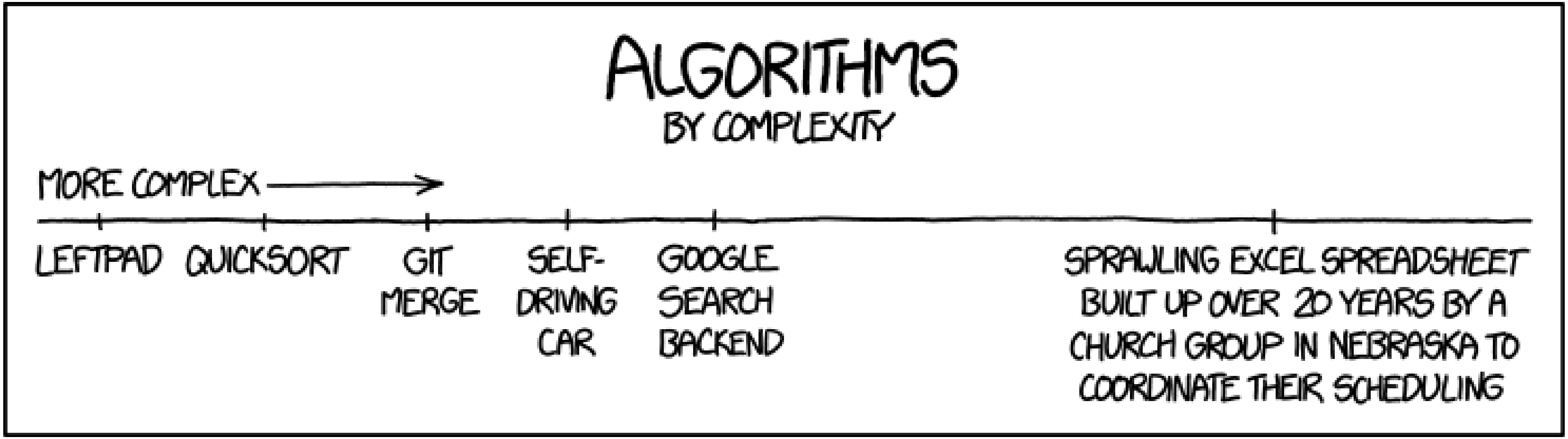
- Was haben wir heute gemacht?
  - Algorithmen mit Pseudocode beschreiben
  - Algorithmen mit Landau-Symbolen bewerten
  - Definition von O, Θ, Ω, o und ω
  - Alternatives Kriterium durch Grenzwerte
  - Mastertheorem

- Was machen wir nächste Woche?
  - amortisierte Analyse
  - Arrays und Listen

Fragen?

Fragen!

Ende



https://xkcd.com/1667/