Analyse von Algorithmen

- Analyse von Algorithmen
  - Laufzeit
  - Platz
- Asymptotische Notation
  - O, Θ, und Ω-Notation
  - Rekursionsgleichungen, Mastertheorem
- Experimentelle Analyse von Algorithmen

Holger Dell
Folien adaptiert von Philip Bille

Analyse von Algorithmen

- Analyse von Algorithmen
  - Laufzeit
  - Platz
- Asymptotische Notation
  - O, Θ, und Ω-Notation
  - Rekursionsgleichungen, Mastertheorem
- Experimentelle Analyse von Algorithmen

Analyse von Algorithmen

- Ziel.
  Ressourcenverbrauch und Korrektheit von Algorithmen feststellen und vorhersagen.

- Beispiel.
  - Funktioniert mein Weg-finden Algorithmus?
  - Wie schnell kann ich eine Weg-finden Anfrage beantworten?
  - Skaliert das auf 10k Anfragen pro Sekunde?
  - Reicht der Speicher aus, wenn die Kartendaten groß sind?
  - Wie viele *cache misses* erzeugt der Algorithmus pro Anfrage? Und wie beeinflusst das die Performanz?

- Primärer Fokus.
  - Korrektheit, Laufzeit, Platzverbrauch.
  - Theoretische und experimentelle Analyse.

Laufzeit

- Laufzeit / Zeitkomplexität.
  - T(n) = Anzahl der Schritte, die der Algorithmus auf Eingabelänge n ausführt.

- Schritte.
  - Lesen/Schreiben im Speicher (`x := y`, `A[i]`, `i = i + 1`, ...)
  - Arithmetische/Boolsche Operationen (`+ - * / % && || & | ^ ~`)
  - Vergleiche (`<` `>` `=<` `=>` `=` ≠)
  - Programmfluss (if-then-else, while, for, goto, function call, ...)

Laufzeit

- im schlimmsten Fall (*worst-case*).
  Maximale Laufzeit über alle Eingaben der Größe n.

- im besten Fall (*best-case*).
  Minimale Laufzeit über alle Eingaben der Größe n.

- im mittleren Fall (*average-case*).
  Durchschnittliche Laufzeit über alle Eingaben der Größe n.

- Terminologie.
  “Zeit” = Laufzeit im schlimmsten Fall (sofern nicht anders deklariert).

- Andere Varianten.
  Amortisiert, randomisiert, deterministisch, nicht-deterministisch, etc.

Platz

- Platz. Anzahl der Speicherzellen, die ein Algorithmus benutzt.

- Speicherzellen.
  - Variablen (auch Zeiger) = 1 Speicherzelle.
  - Feld der Länge k = k Speicherzellen.

- Terminologie. Platzverbrauch, Speicher, Platzkomplexität.

- Varianten. worst-case, best-case, ...

Analyse von Algorithmen

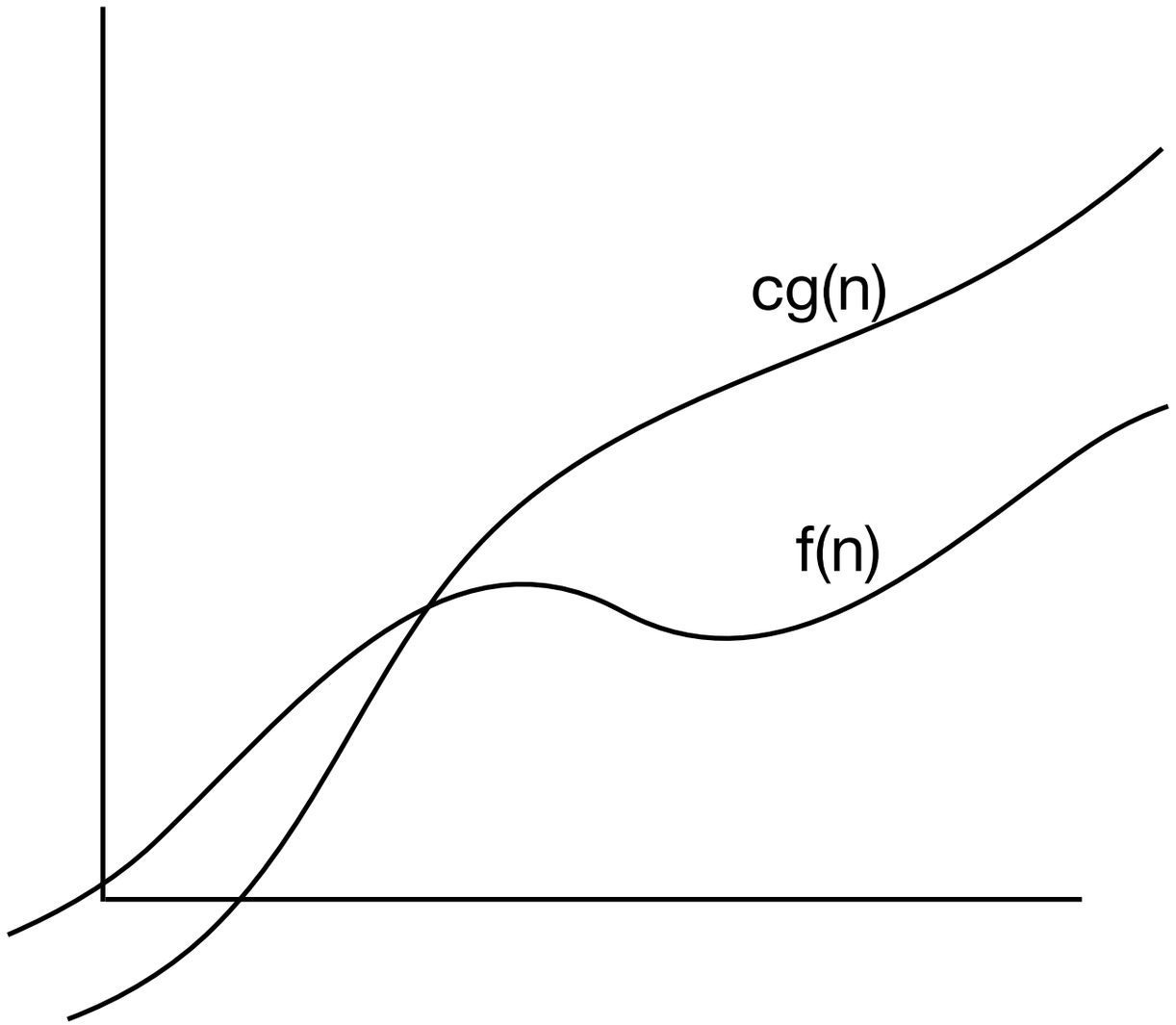
- Analyse von Algorithmen
  - Laufzeit
  - Platz
- Asymptotische Notation
  - O, Θ, und Ω-Notation
  - Rekursionsgleichungen, Mastertheorem
- Experimentelle Analyse von Algorithmen

Asymptotische Notation

- Asymptotische Notation.
  - O, Θ, und Ω-Notation.
  - Notation, um das asymptotische Wachstum von Funktionen zu beschränken.
  - Ganz wichtig, um Laufzeit und Platzverbrauch von Algorithmen zu beschreiben.

O-Notation

- Intuitive Definition. f(n) = O(g(n)), wenn f(n) ≤ cg(n) für große n.



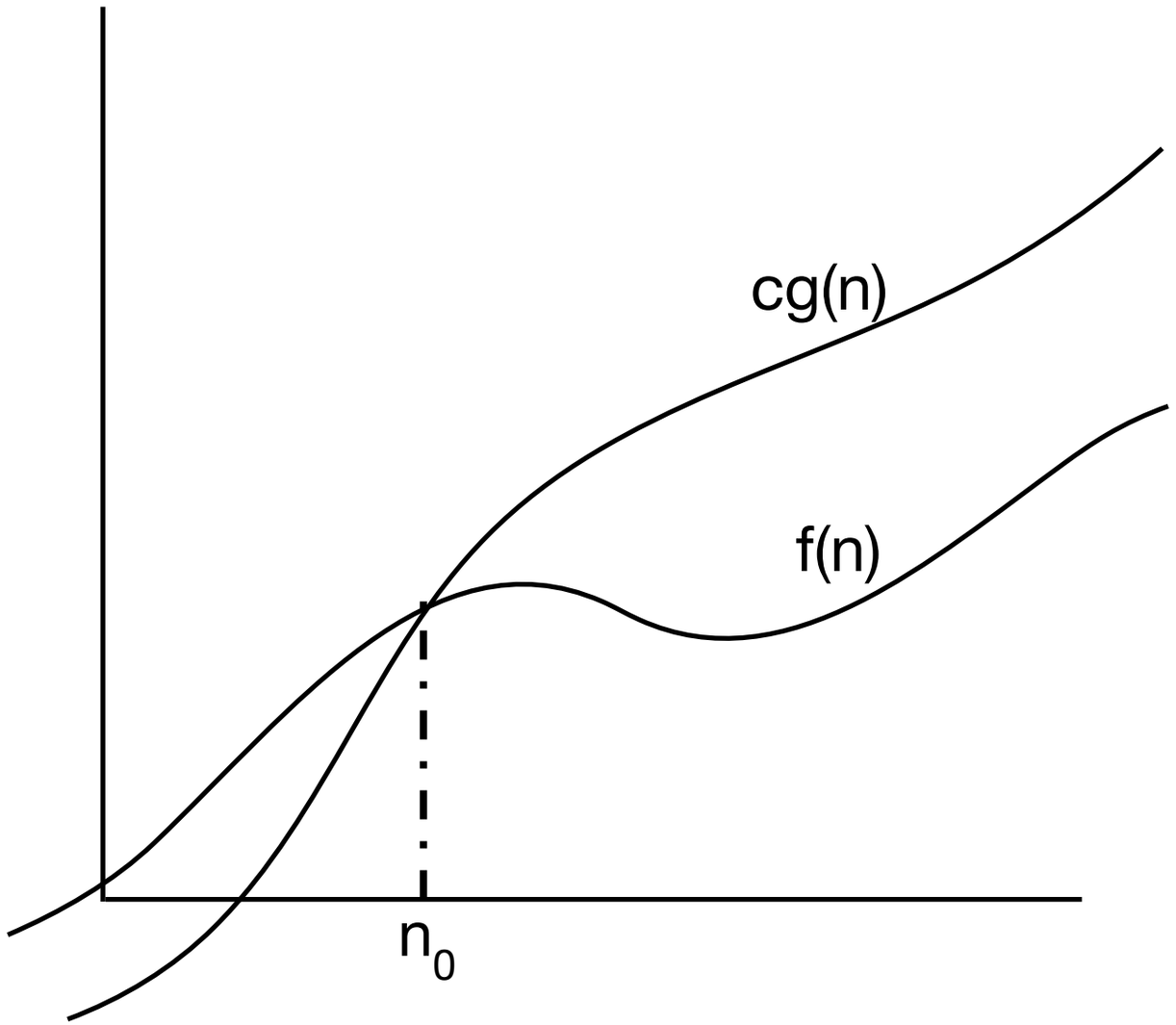
O-Notation

- Beispiel. $f(n) = O(n^2)$ gilt, wenn $f(n) \leq cn^2$ für große n.

- $5n^2 = O(n^2)$?
  - $5n^2 \leq 5n^2$ für große n.
- $5n^2 + 3 = O(n^2)$?
  - $5n^2 + 3 \leq 6n^2$ für große n.
- $5n^2 + 3n = O(n^2)$?
  - $5n^2 + 3n \leq 6n^2$ für große n.
- $5n^2 + 3n^2 = O(n^2)$?
  - $5n^2 + 3n^2 = 8n^2 \leq 8n^2$ für große n.
- $5n^3 = O(n^2)$?
  - für alle Konstanten c gilt $5n^3 \geq cn^2$ sobald n groß genug ist.

O-Notation

- Intuitive Definition. f(n) = O(g(n)), wenn f(n) ≤ cg(n) für große n.
- Definition. f(n) = O(g(n)), wenn es Konstanten c, $n_0 > 0$ gibt,
  so dass für alle $n \geq n_0$ gilt: f(n) ≤ cg(n).



O-Notation

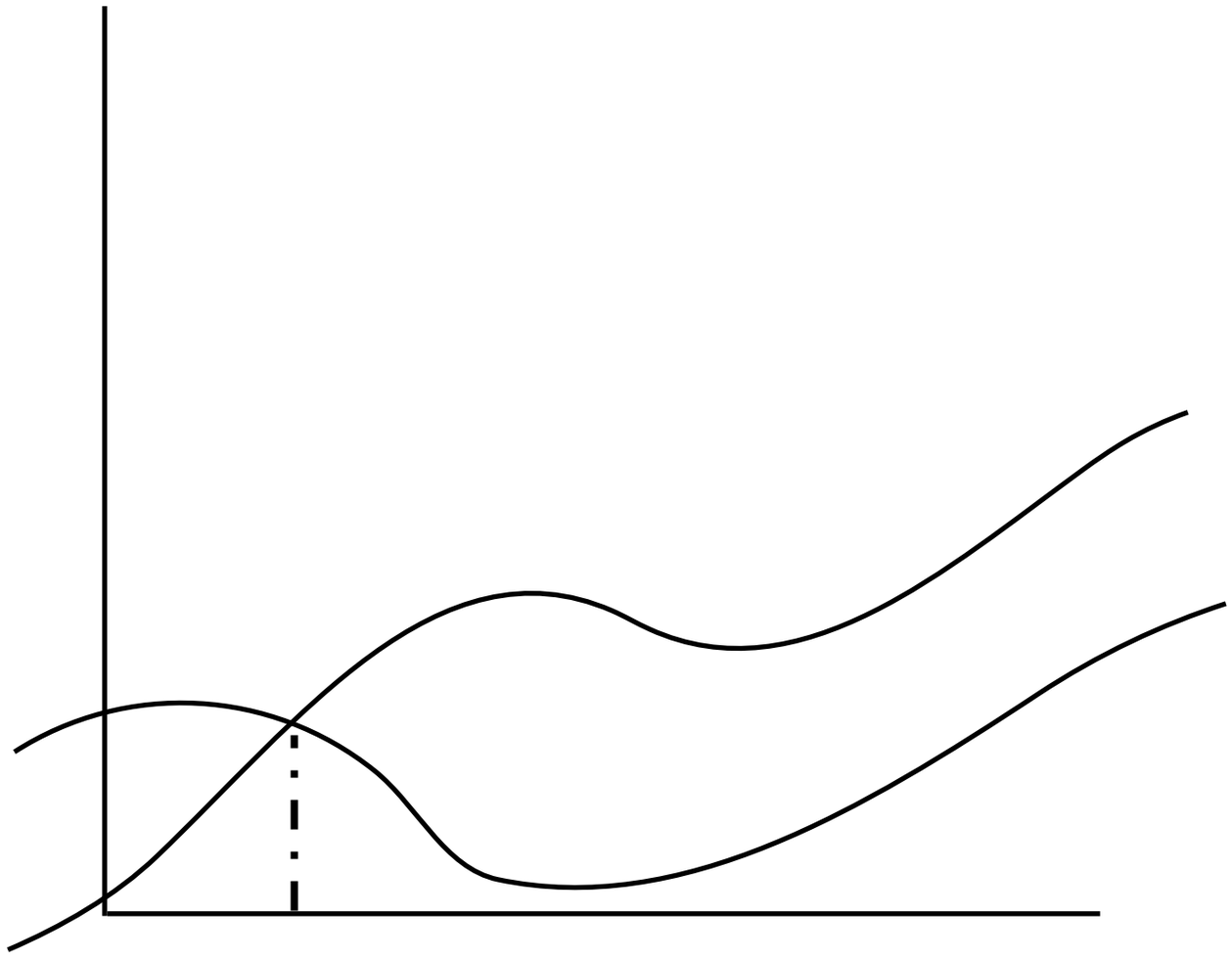
- Notation.
  - Formal ist O(g(n)) ist eine Menge von Funktionen.
  - = meint also eigentlich $\in$ oder $\subseteq$.
  - f(n) = $O(n^2)$ ist ok. $O(n^2)$ = f(n) ist nicht ok!
- Schreiben auch manchmal f(n) $\leq$ O(g(n)) oder f(n) $\in$ O(g(n))

O-Notation

- $f(n) = O(g(n))$, wenn $f(n) \leq cg(n)$ für große n.
- Übung.
  - Seien $f(n) = 3n + 2n^3 - n^2$ und $h(n) = 4n^2 + \log n$
  - Welche Aussagen sind wahr?
    - $f(n) = O(n)$
    - $f(n) = O(n^3)$
    - $f(n) = O(n^4)$
    - $h(n) = O(n^2 \log n)$
    - $h(n) = O(n^2)$
    - $h(n) = O(f(n))$
    - $f(n) = O(h(n))$

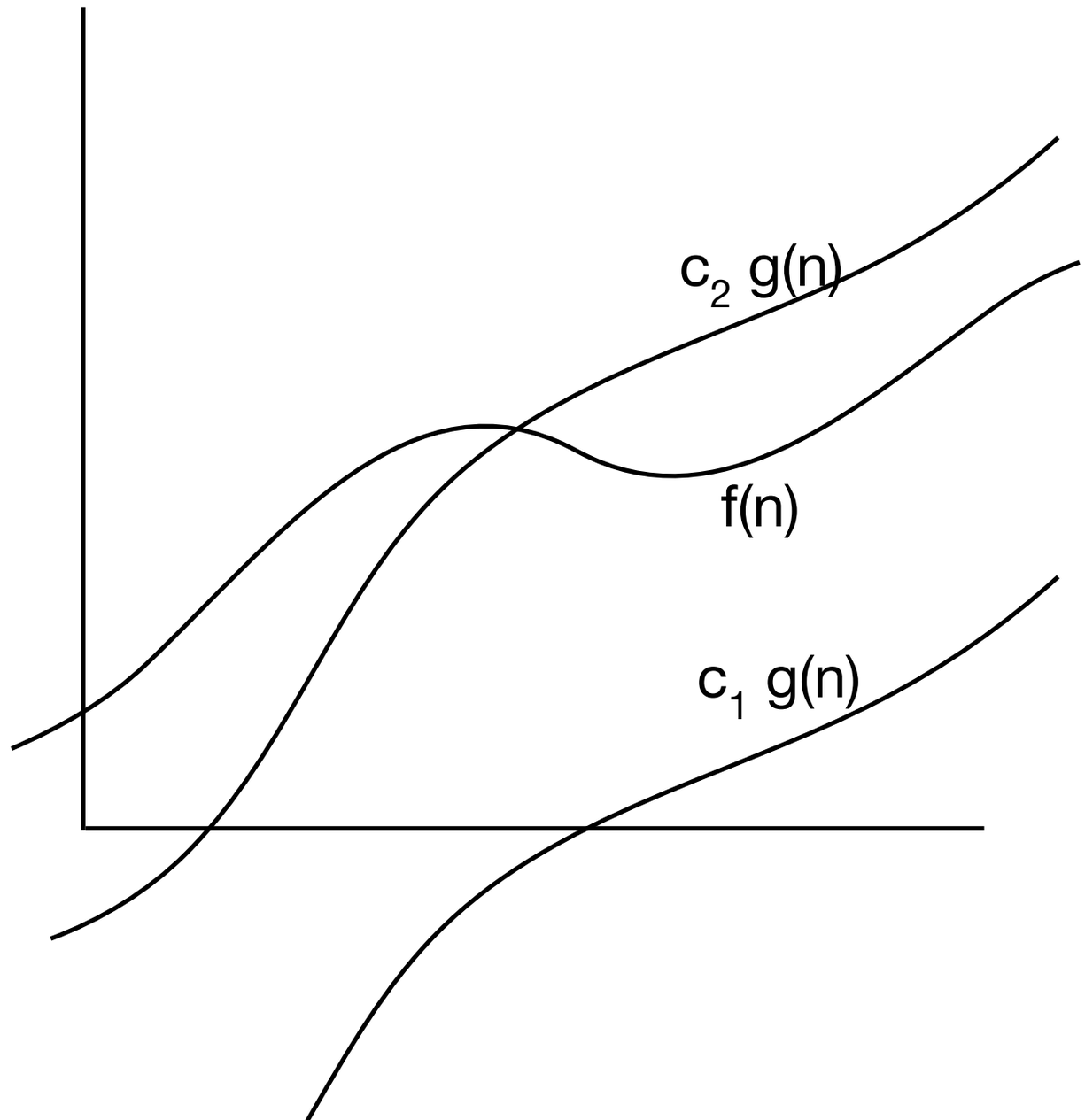
Ω-Notation

- Intuitive Definition. f(n) = Ω(g(n)), wenn f(n) ≥ cg(n) für große n.
- Definition. f(n) = Ω(g(n)), wenn es Konstanten c, $n_0 > 0$ gibt,
  so dass für alle $n \geq n_0$ gilt: f(n) ≥ cg(n)



Θ-Notation

- Definition. f(n) = Θ(g(n)), wenn f(n) = O(g(n)) **und** f(n) = Ω(g(n))



Asymptotische Notation

- $f(n) = O(g(n))$ wenn $f(n) \leq cg(n)$ für große n.
- $f(n) = \Omega(g(n))$ wenn $f(n) \geq cg(n)$ für große n.
- $f(n) = \Theta(g(n))$ wenn $f(n) = O(g(n))$ **und** $f(n) = \Omega(g(n))$.

- Übung. Welche Aussagen sind wahr? ($\log^k n$ ist $(\log n)^k$)
  - $n \log^3 n = O(n^2)$
  - $2^n + 5n^7 = \Omega(n^3)$
  - $n^2(n - 5)/5 = \Theta(n^2)$
  - $4\ n^{1/100} = \Omega(n)$
  - $n^3/300 + 15 \log n = \Theta(n^3)$
  - $2^{\log n} = O(n)$
  - $\log^2 n + n + 7 = \Omega(\log n)$

Asymptotische Notation

- Grundlegende Eigenschaften.
  - Jedes Polynom wächst proportional zu seinem Leitmonom.

$$a_0 + a_1 n + a_2 n^2 + \cdots + a_d n^d = \Theta(n^d)$$

  - Alle Logarithmen sind asymptotisch gleich.

$$\log_a(n) = \frac{\log_b n}{\log_b a} = \Theta(\log_c(n)) \qquad \text{for all constants } a, b > 0$$

  - Logarithmen wachsen langsamer als alle Polynome.

$$\log(n) = O(n^d) \qquad \text{for all } d > 0$$

  - Polynome wachsen langsamer als alle Exponentialfunktionen.

$$n^d = O(r^n) \qquad \text{for all } d > 0 \text{ and } r > 1$$

Typische Laufzeiten von Schleifen

```
for i = 1 to n
   {Θ(1) Schritte}
```

Laufzeit $\Theta(n)$

```
for i = 1 to n
  for j = 1 to n
    {Θ(1) Schritte}
```

Laufzeit $\Theta(n^2)$

```
for i = 1 to n
  for j = 1 to i
    {Θ(1) Schritte}
```

Laufzeit $\Theta(1+2+3+...+n) = \Theta(n^2)$

Typische Rekursionsgleichung

$$T(n) = \begin{cases} T(n/2) + \Theta(1) & \text{if } n > 1 \\ \Theta(1) & \text{if } n = 1 \end{cases}$$

wie binäre Suche
T(n) = Θ(log n)

$$T(n) = \begin{cases} 2T(n/2) + \Theta(n) & \text{if } n > 1 \\ \Theta(1) & \text{if } n = 1 \end{cases}$$

wie merge sort
T(n) = Θ(n log n)

$$T(n) = \begin{cases} 2T(n/2) + \Theta(1) & \text{if } n > 1 \\ \Theta(1) & \text{if } n = 1 \end{cases}$$

T(n) = Θ(n)

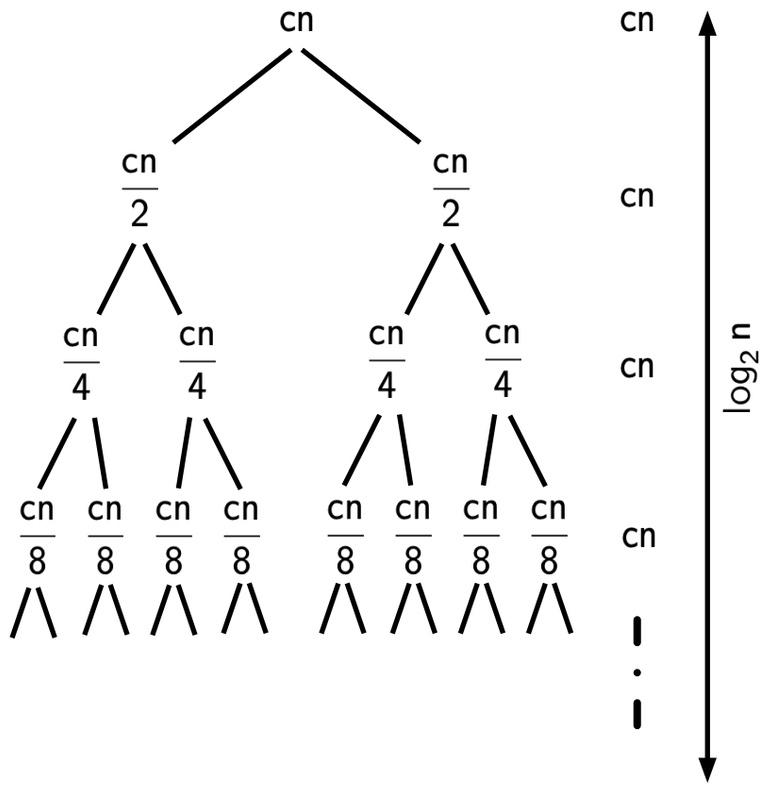
$$T(n) = \begin{cases} T(n/2) + \Theta(n) & \text{if } n > 1 \\ \Theta(1) & \text{if } n = 1 \end{cases}$$

T(n) = Θ(n)

Rekursionsgleichungen lösen

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{falls } n > 1, \\ \Theta(1) & \text{falls } n = 1. \end{cases}$$

- Raten. Geschlossene Form raten und mit Induktion beweisen.
- Rekursionsbaum. Nützlich, um geschlossene Form zu finden.



- Mastertheorem. Allgemeine Methode.

Mastertheorem

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{falls } n > 1, \\ \Theta(1) & \text{falls } n = 1. \end{cases}$$

- Hier sind a≥1 und b>1 konstant.
- Abhängig von f(n) gibt es drei Fälle:

1. Falls $f(n) = O(n^{(\log_b a)-\varepsilon})$ für eine Konstante $\varepsilon > 0$ gilt, dann gilt $T(n) = \Theta(n^{\log_b a})$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, dann gilt $T(n) = \Theta(n^{\log_b a} \log n)$.

3. Falls $f(n) = \Omega(n^{(\log_b a)+\varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und hinreichend große n gelten, dann ist $T(n) = \Theta(f(n))$.

o und ω Notation

- f(n) = o(g(n)) wenn $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

  “f wächst asymptotisch echt langsamer als g”

- f(n) = ω(g(n)) wenn $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

  “f wächst asymptotisch echt schneller als g”

- Beispiele. Es gilt:
  - $n = o(n^2)$
  - $2^n = \omega(n^{100})$
  - $\log^{100} n = o(n^{0.001})$

Analyse von Algorithmen

- Analyse von Algorithmen
  - Laufzeit
  - Platz
- Asymptotische Notation
  - O, Θ, und Ω-Notation
  - Rekursionsgleichungen, Mastertheorem
- Experimentelle Analyse von Algorithmen

Experimentelle Analyse

- Frage. Können wir die theoretische Laufzeit experimentell abschätzen?
- Verdopplungstechnik.
  - Lass das Programm laufen und miss die Zeit auf Eingaben verschiedener Größe.
  - Untersuche die zusätzliche Zeit, wenn wir die Größe verdoppeln.
- Beispiel.
  - Eingabegröße x 2 und Zeit x 4.
  - $\Rightarrow$ Algorithmus läuft wahrscheinlich in **quadratischer** Zeit.
  - $T(n) = cn^2$
  - $T(2n) = c(2n)^2 = c2^2n^2 = c4n^2$
  - Bruch $T(2n)/T(n) = 4$

n	Zeit	Bruch
5000	0	-
10000	0.2	-
20000	0.6	3
40000	2.3	3.8
80000	9.4	4
160000	37.8	4

Analyse von Algorithmen

- Analyse von Algorithmen
  - Laufzeit
  - Platz
- Asymptotische Notation
  - O, Θ, und Ω-Notation
  - Rekursionsgleichungen, Mastertheorem
- Experimentelle Analyse von Algorithmen