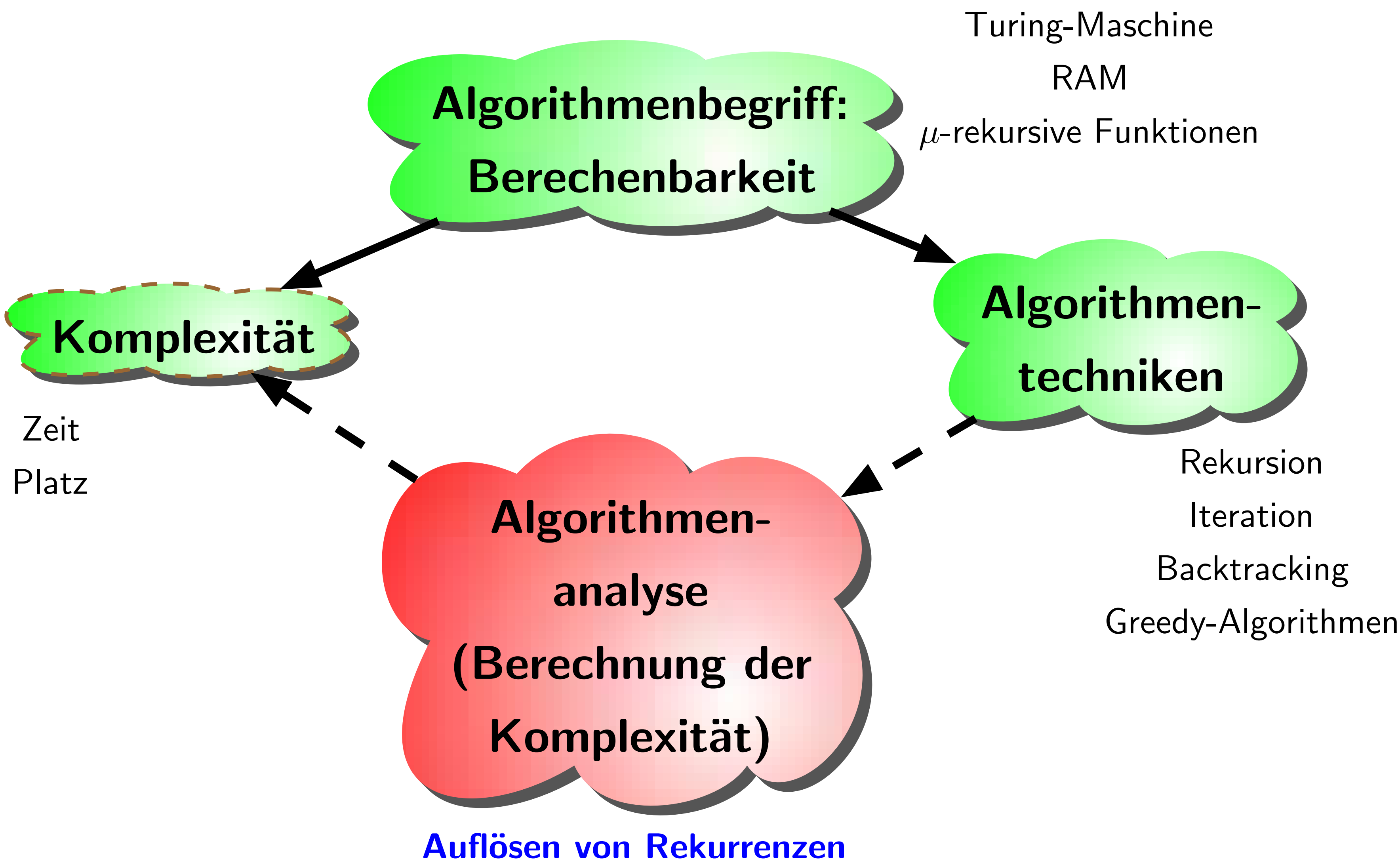
Über-/Rückblick



Einige Feststellungen

- *Mergesort* hat eine *worst-case*-Zeitkomplexität von $O(n \log n)$.
- *Insertsort* hat eine *worst-case*-Zeitkomplexität von $\Theta(n^2)$.
- Somit ist Mergesort *asymptotisch schneller* als Insertsort.

Manchmal ist es möglich, die exakte Zeit für einen gegebenen Computer festzustellen, aber das ist meistens nicht lohnenswert.

Asymptotische Algorithmenanalyse

- Analyse der Hauptidee des Algorithmus.
- **Mergesort** ist ein *divide-and-conquer*-Algorithmus:
  - Ein Problem wird auf zwei Probleme etwa halber Größe reduziert (mit linearem Zeitaufwand für *decomposition* und *composition*). $\Rightarrow \Theta(n \log n)$
- **Insertsort** ist ein Sortieralgorithmus, für den nach dem i-ten Zyklus die ersten i Elemente der Folge sortiert sind und im i-ten Schritt das i-te Element in die richtige Position gebracht wird.

$$\underbrace{a_1 \ldots\ldots a_{i-1}}_{\text{sortiert}} a_i \ldots\ldots a_n$$

i-ter Schritt

Analyse (2)

- Vorsicht ist bei der asymptotischen Analyse geboten – besonders, wenn die Komplexität nicht nur vom Umfang der Eingabedaten abhängt.
  - Für die Zeitkomplexität von Insertsort gilt:

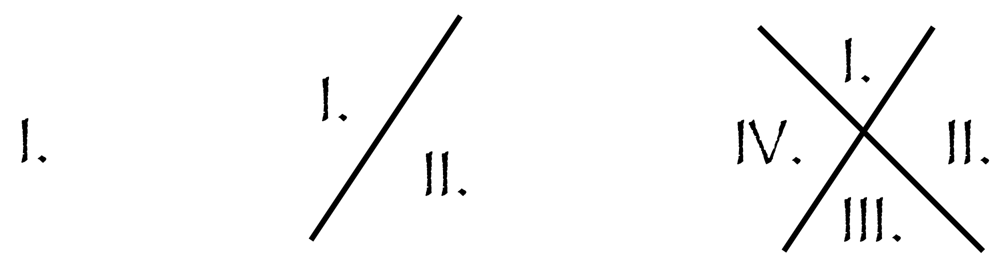
$$
\begin{aligned}
T(n) &\in \Omega(n) \\
T(n) &\in O(n^2)
\end{aligned}
$$

    - Kann man nun schreiben „$T(n) \in \Omega(n^2)$“ ?
    - Nein! Es gibt Eingaben, für die Insertsort $\Theta(n)$ Zeit braucht. Aber für die *worst-case*-Zeitkomplexität gilt:

$$T_{worst}(n) \in \Omega(n^2)$$

Einführungsbeispiel: Ebenen

- In maximal wieviele Gebiete zerteilen n Geraden die euklidsche Ebene?
  - **keine Gerade ($n = 0$):** das Gebiet bleibt unverändert. Also $L_0 = 1$.
  - **eine Gerade ($n = 1$):** zwei Gebiete, d.h. $L_1 = 2$.
  - **zwei Geraden ($n = 2$):** vier Gebiete, d.h. $L_2 = 4$.

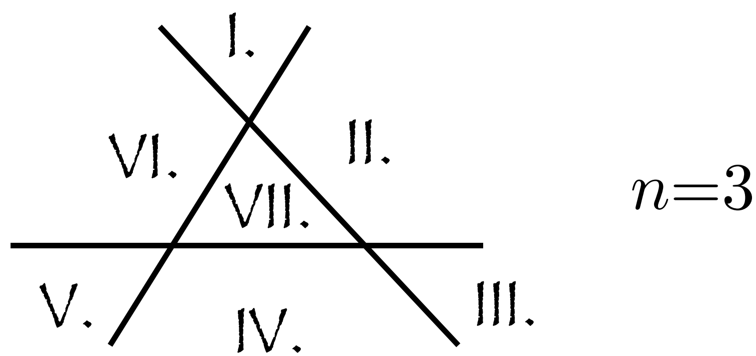


$n{=}0$ $\quad$ $n{=}1$ $\quad$ $n{=}2$

Ebenen (2)

Die Vermutung $L_n = 2^n$, die für $n = 0, 1, 2$ stimmt, wird durch L_3 widerlegt:

- Für $n = 3$ ergibt sich folgendes Bild:



- Also: $L_3 = 7$.
- Ist die neue Gerade zu keiner anderen parallel, so schneidet sie alle vorherigen Geraden genau einmal (höchstens in $n - 1$ verschiedenen Punkten).

Rekurrenzgleichung

- Wenn die n-te Gerade *nicht* durch einen früheren Schnittpunkt geht, so wird die Maximalzahl erreicht. Es ergibt sich die Rekurrenz $L_n := L_{n-1} + n$ für die gesuchte Zahl L_n mit dem Anfangswert $L_0 := 1$.
- Um nun eine geschlossene Formel für L_n zu gewinnen, wickeln wir die ersten Rekurrenzen ab:

$$
\begin{aligned}
L_0 &= 1 \\
L_1 &= L_0 + 1 = 1 + 1 = 2 \\
L_2 &= L_1 + 2 = (L_0 + 1) + 2 = 1 + 1 + 2 = 4 \\
L_3 &= (L_1 + 2) + 3 = ((L_0 + 1) + 2) + 3 \\
&= 1 + 1 + 2 + 3 = 7
\end{aligned}
$$

Summenformel

- Die offensichtliche Vermutung ist also

$$L_n \quad = \quad 1 + \sum_{i=1}^{n} i$$

- Diese Vermutung muß aber noch formal bewiesen werden, was wir durch vollständige Induktion tun werden.

Die **Verankerung** für $n = 0$ ergibt sich wie gewünscht:

$$L_0 = 1 + \sum_{i=1}^{0} i = 1$$

Summenformel (2)

Induktionsannahme:

$$L_m = 1 + \sum_{i=1}^{m} i \quad \text{für ein festes} \quad m \in I\!N$$

Induktionsschritt:

$$\begin{aligned}
L_{m+1} &= L_m + (m+1) && \text{(entsprechend der Rekurrenz)} \\
&= 1 + \sum_{i=1}^{m} i + (m+1) && \text{(nach Induktionsannahme)} \\
&= 1 + \sum_{i=1}^{m+1} i && \text{(trivial)}
\end{aligned}$$

Damit ist die Formel $L_n = 1 + \sum_{i=1}^{n} i$ bewiesen.

geschlossene Formel

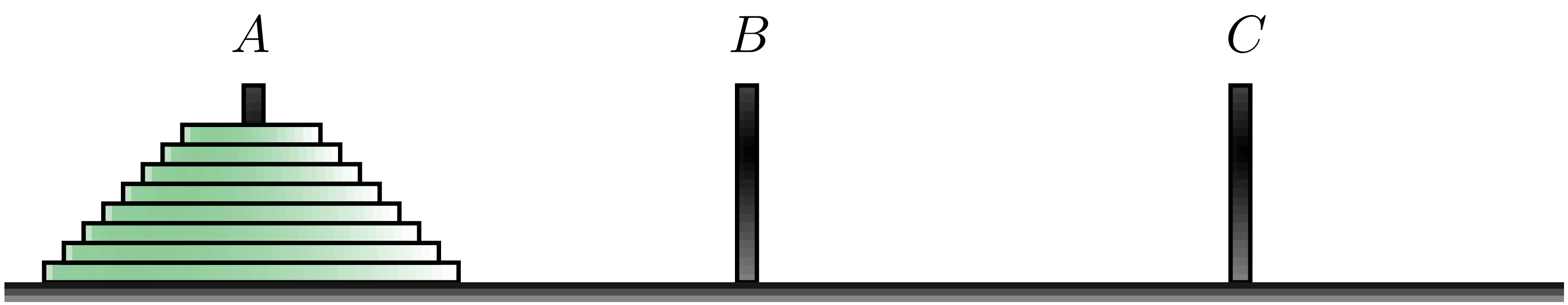
- Die Summenformel ist keine *geschlossene* Formel!
- Mit Hilfe der Gaußschen Formel $\sum_{i=1}^{m} i = \frac{n(n+1)}{2}$ erhalten wir nun eine geschlossene Formel für L_n:

$$L_n = 1 + \frac{n(n+1)}{2}$$

Grundsätzliches Vorgehen beim Lösen von Rekurrenzgleichungen: Zuerst die Werte für kleine Argumente berechnen. Diese Werte können helfen

1. die Lösung zu finden,
2. die Lösung zu verifizieren.

2. Beispiel: Türme von Hanoi



Ausgangssituation:

Seien drei Stangen gegeben und n Scheiben, die der Größe nach sortiert auf Stange A liegen.

Aufgabe:

Die Scheiben von Stange A sollen auf Stange C verschoben werden, wobei in jedem Schritt nur eine Scheibe bewegt und niemals eine größere Scheibe auf eine kleinere gelegt werden kann.

rekursiver Algorithmus

1. Verschiebe die $n-1$ oberen Scheiben von A auf B.
2. Verschiebe die größte Scheibe von A auf C.
3. Verschiebe alle $n-1$ Scheiben von B auf C.

Verschiebungs-Analyse: Sei $T(n)$ die Anzahl der Verschiebungen, die obiger Algorithmus braucht, um n Scheiben zu verschieben. Es gilt

$$T(n) = \begin{cases} 1 & \text{falls } n = 1 \\ 2T(n-1)+1 & \text{falls } n > 1 \end{cases}$$

Frage: Kann man das schneller tun? **Antwort:** Nein.

Hanoi-Rekurrenz

Rekurrenzgleichung für die Türme von Hanoi:

$$T(n) = \begin{cases} 1 & \text{falls } n = 1 \\ 2T(n-1)+1 & \text{falls } n \geq 2 \end{cases}$$

Tabellarische Darstellung:

n	1	2	3	4	5	6	7	8	9	10
$T(n)$	1	3	7	15	31	63	127	255	511	1023
2^n	2	4	8	16	32	64	128	256	512	1024

Vermutung: $T(n) = 2^n - 1$?

Beweis

Wir betrachten zwei Beweismöglichkeiten:

1. vollständige Induktion
2. Reduzieren auf Summen

Induktionsbeweis für: $T(n) = 2^n - 1$ ist Lösung.

Induktionsanfang: Für $n = 1$ gilt $T(1) = 2^1 - 1 = 1$.

Induktionsschritt: Nach der *Induktionsannahme* sei für ein gegebenes $n \geq 1 \quad T(n) = 2^n - 1$ eine Lösung der Rekurrenzgleichung .

Nach *Rekurrenzgleichung:*

$$T(n+1) = 2T(n) + 1 = 2(2^n - 1) + 1 = 2^{n+1} - 1$$

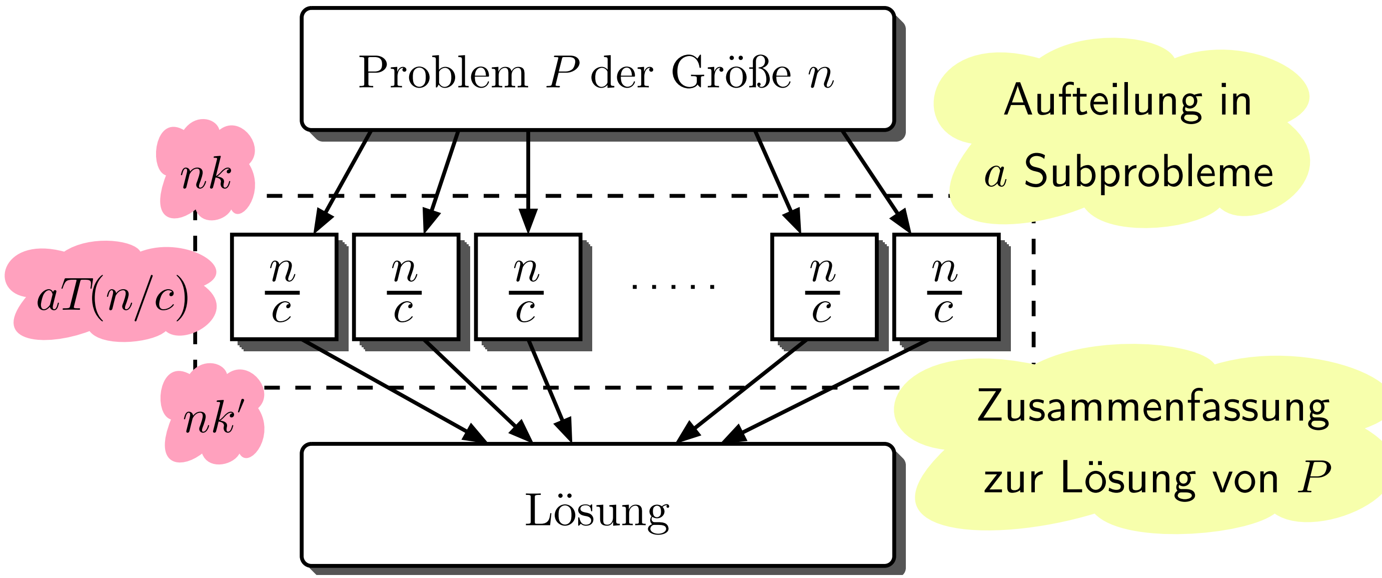
Reduzierung auf Summen

- Es kann vorkommen, daß nach dem Studium endlich vieler Anfangswerte keine Vermutung naheliegt, die beweisbar eine Lösung darstellt.
- Hier kann nur noch formal nach einer Lösung gesucht werden.
  - *Abwickeln der Rekursion*
  - Zusammenfassen / Ersetzen bekannter (Teil-)Summen
- Die Methode kann zu einem standardisierten Verfahren zum Auffinden einer geschlossenen Formel für beliebige *lineare* Rekurrenzgleichungen erweitert werden!

Hanoi — auf Summen reduzieren

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 \\
&= 2(2T(n-2)+1)+1 = 4T(n-2)+2+1 \\
&= 4(2T(n-3)+1)+2+1 \\
&= 2^3T(n-3)+2^2+2^1+2^0 \\
&= 2^kT(n-k)+\sum_{i=0}^{k-1} 2^i \\
&= 2^{n-1}T(1)+\sum_{i=0}^{n-2} 2^i \\
&= \sum_{i=0}^{n-1} 2^i \quad \text{(geometrische Reihe)} \\
&= \frac{2^n-1}{2-1} = 2^n-1
\end{aligned}
$$

3. Beispiel: *divide and conquer*



Gesucht: Zeitkomplexität $T(n)$ dieses Algorithmus.

$$T(n) = \begin{cases} d & \text{falls } n = 1 \\ aT(\frac{n}{c}) + kn + k'n & \text{falls } n > 1 \end{cases}$$

Vereinfachung

- Zur Analyse des *divide-and-conquer*-Algorithmus vereinfachen wir die Rekurrenz zu

$$T(n) = \begin{cases} b & \text{falls} \quad n = 1 \\ aT(\frac{n}{c}) + bn & \text{falls} \quad n > 1. \end{cases}$$

- k und k' sind für jedes Problem bekannt und konstant!
- $d \leq k + k'$
- Also wählen wir $b := k + k'$

Abwickeln der Rekurrenz

Durch rekursives Einsetzen (Abwickeln) erhalten wir:

$$
\begin{aligned}
T(n) &= aT\left(\frac{n}{c}\right) + bn = a\left(aT\left(\frac{n}{c^2}\right) + b\frac{n}{c}\right) + bn \\
&= a^2T\left(\frac{n}{c^2}\right) + bn\frac{a}{c} + bn \\
&= a^2\left(aT\left(\frac{n}{c^3}\right) + b\frac{n}{c^2}\right) + bn\frac{a}{c} + bn \\
&= a^3T\left(\frac{n}{c^3}\right) + bn\left(\frac{a}{c}\right)^2 + bn\frac{a}{c} + bn \\
&= a^kT\left(\frac{n}{c^k}\right) + bn \cdot \sum_{i=0}^{k-1}\left(\frac{a}{c}\right)^i
\end{aligned}
$$

Wenn $n = c^k$ ist, so bricht das Verfahren bei $T(n) = a^k b + bn\sum_{i=0}^{k-1}(\frac{a}{c})^i$ ab, denn es ist $T(1) = b$.

Der Weg zur geschlossenen Formel

Mit $n = c^k$ folgt $k = \log_c(n)$ und somit

$$
\begin{aligned}
T(n) &= a^k b + bn \sum_{i=0}^{k-1} \left(\frac{a}{c}\right)^i \\
&= a^k b + bn \sum_{i=0}^{k} \left(\frac{a}{c}\right)^i - bn \left(\frac{a}{c}\right)^k \\
&= a^k b \left(1 - \frac{n}{c^k}\right) + bn \sum_{i=0}^{k} \left(\frac{a}{c}\right)^i \\
&= bn \sum_{i=0}^{\log_c(n)} \left(\frac{a}{c}\right)^i
\end{aligned}
$$

Je nach dem Verhältnis von a zu c ergeben sich unterschiedliche Lösungen!

Fallunterscheidung

Fall 1, $a < c$**:** $\sum_{i=0}^{\infty} \left(\frac{a}{c}\right)^i$ konvergiert $\Rightarrow T(n)$ proportional zu n

Fall 2, $a = c$**:** $T(n) = bn(\log_c(n) + 1) \Rightarrow T(n)$ prop. $n \log(n)$

Fall 3, $a > c$**:** $T(n)$ proportional zu $n^{\log_c(a)}$

Dann folgt
$$\begin{aligned} T(n) &= bn \sum_{i=0}^{\log_c(n)} \left(\frac{a}{c}\right)^i \\ &= bn \frac{\left(\frac{a}{c}\right)^{\log_c(n)+1} - 1}{\frac{a}{c} - 1} \approx bn \left(\frac{a}{c}\right)^{\log_c(n)} \\ &= bn \frac{a^{\log_c(n)}}{n} = ba^{\log_c(n)} = bn^{\log_c(a)} \end{aligned}$$

weil $a^{\log_c(n)} = n^{\log_c(a)}$ stets gilt.

Schlußfolgerung

Wichtige Feststellung zu *divide-and-conquer*-Algorithmen:

Die Zeitkomplexität eines *divide-and-conquer*-Algorithmus hängt nur von dem Verhältnis $\frac{a}{c}$ ab – und nicht von der Art des Problems oder vom Lösungsweg –, wenn der Zeitbedarf für die Zerlegung in Teilprobleme und die Zusammenfassung der Teillösungen proportional zur Größe des Problems ist.

Ein praktisches Problem

Für das Platinenlayout sind Anordnungen von Chips der Maße 1×2 cm auf einer „Bahn“ von 2 cm Höhe und bisher unbestimmter Länge n nötig. Mögliche Anordnungen sind z.B.:

$n{=}5$ $n{=}6$

- Sei T_n die Anzahl der verschiedenen Anordnungen auf einer Bahn der Länge n.
- Es gilt $T_n = T_{n-1} + T_{n-2}$ für $n \geq 2$
- Anfangswerte: $T_0 := 1$ und $T_1 := 1$

Fibonacci-Zahlen

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2} \quad \text{für } n > 1$$

- Wir suchen eine Lösung in der Form $F_n = r^n$ (r ist unbekannte Konstante).
- Existiert eine solche Lösung, dann gilt:

$$r^n = r^{n-1} + r^{n-2} \quad \text{für jedes } n > 1$$

und daraus folgt, dass entweder $r = 0$ oder $r^2 = r + 1$.

- Diese Gleichung hat zwei Lösungen:

$$r_1 = \frac{1+\sqrt{5}}{2}, \qquad r_2 = \frac{1-\sqrt{5}}{2}$$

Fibonacci-Zahlen (2)

- Eine allgemeine Lösung der Gleichung hat die Form $\lambda r_1^n + \mu r_2^n$ wobei λ und μ Konstanten sind.
- Daraus folgt: $\lambda + \mu = F_0 = 0, \quad \lambda r_1 + \mu r_2 = F_1 = 1$
- $\lambda = -\mu = \frac{1}{\sqrt{5}}$ und

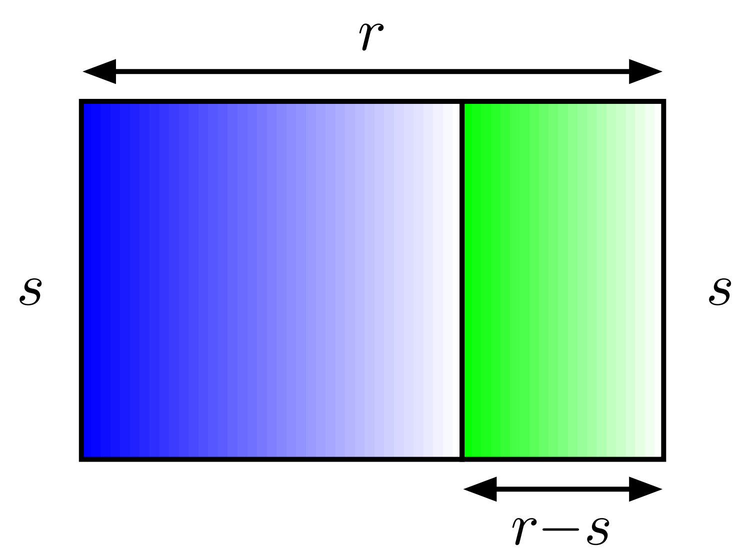
$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

- Wegen $\lim_{n\to\infty}\left(\frac{1-\sqrt{5}}{2}\right)^n = 0$ gilt:

$$F_n \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n \qquad \text{für } n \to \infty$$

Der goldene Schnitt

- Das abgetrennte Rechteck soll das gleiche Seitenverhältnis haben, wie das ursprüngliche Rechteck.



- Dafür muß $\frac{r}{s} = \frac{s}{r-s}$ gelten.
- Es folgt $x = \frac{r}{s} = \frac{s}{r-s} = \frac{1}{x-1}$ oder $x^2 - x - 1 = 0$.
- Lösungen: $x = \Phi = \frac{1+\sqrt{5}}{2}$ und $x = \hat{\Phi} = \frac{1-\sqrt{5}}{2}$
- Eine Größe r wird nach dem goldenen Schnitt geteilt, wenn der Teil s das geometrische Mittel von r und $r-s$ ist: $s = \sqrt{r(r-s)}$.