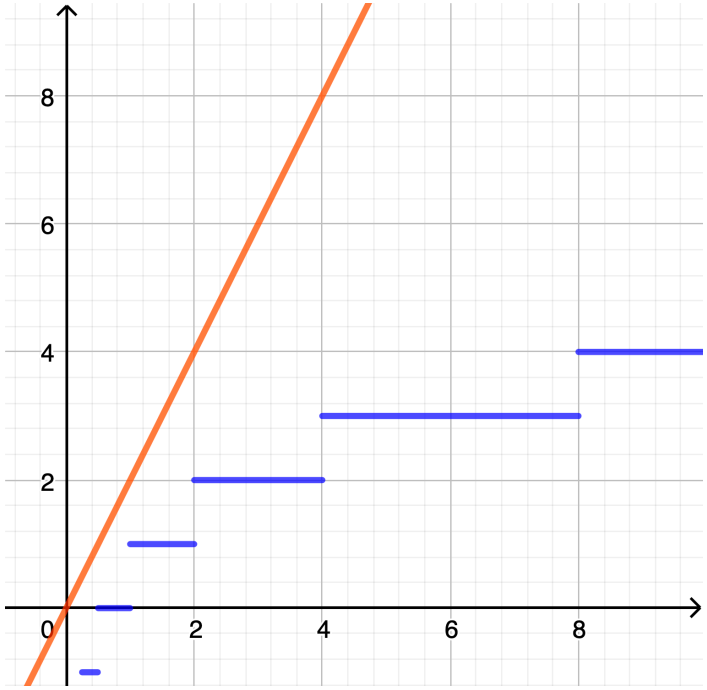
Landau-Notation

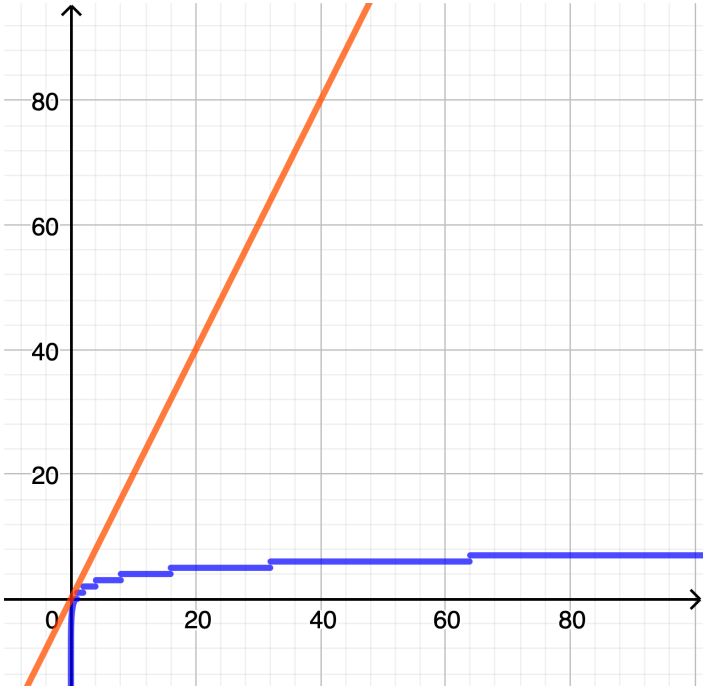
Inhalt

- In diesem Video nähern wir uns weiter, die Effizienz eines Algorithmus zu beschreiben und führen eine dafür geeignete Notation ein – die Landau-Notation.
- Diese ermöglicht auch den Vergleich des Verhaltens zweier Algorithmen hinsichtlich ihrer Effizienz, wenn sie zur Lösung unterschiedlicher Probleme konzipiert wurden.

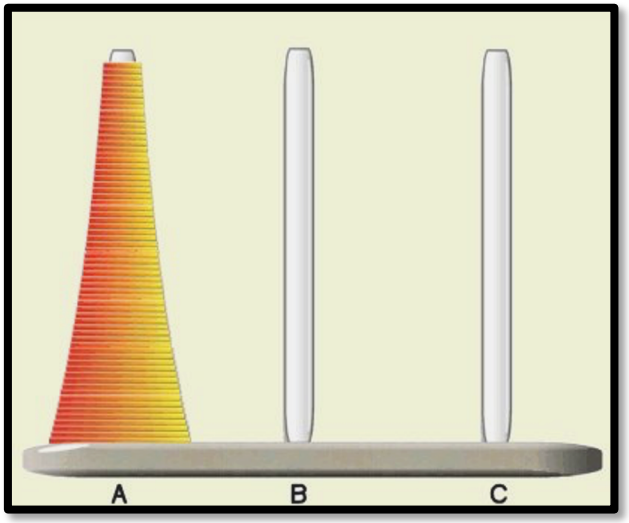
Effizienz von Algorithmen

- Wir haben schon gesehen: Die Anzahl durchzuführender Schritte S eines Algorithmus hängt von der Inputgröße ab:
  - Potenz berechnen: mindestens $S(n) = \lceil \log_2 n \rceil$ Rechenoperationen
  - Polynom auswerten: mindestens $S(n) = 2n$ Rechenoperationen

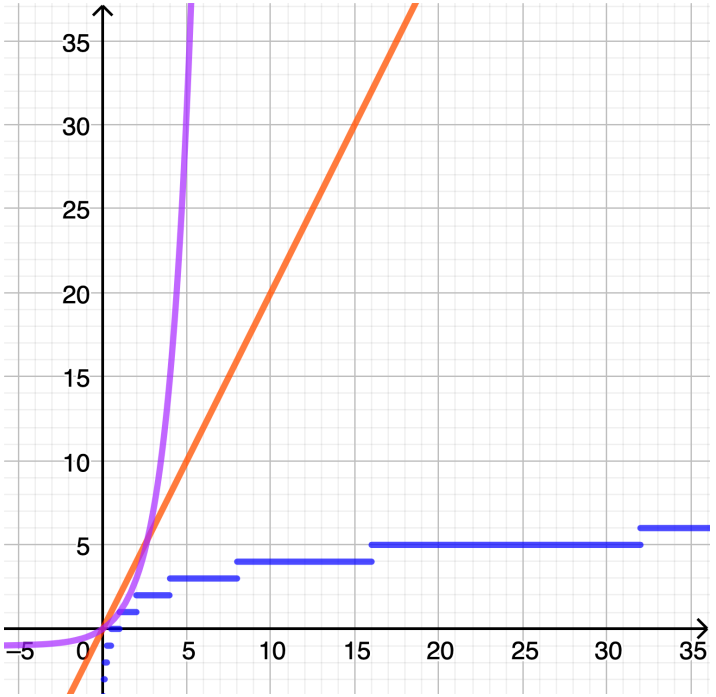




Türme von Hanoi



- Bei den Türmen von Hanoi geht es nicht um Rechenoperationen, sondern um Verlegeoperationen.
- Wir haben schon gesehen: Die Anzahl der benötigten Schritte zur Lösung des Problems hängt von der Anzahl an Scheiben (hier $n = 64$) ab.
- Diese Abhängigkeit ist sogar exponentiell, so dass es schnell „ewig“ dauert.
- Die Anzahl der Schritte ist also auch hier abhängig von der Inputgröße n: $S(n) = 2^n - 1$.
- Was genau die betrachteten Schritte sowie die jeweiligen Inputgrößen in einem Algorithmus sind, hängt vom betrachteten Problem ab.



Quelle: https://www.lehrer.uni-karlsruhe.de/~za714/informatik/infkurs/alg_brek2.html

Bubblesort

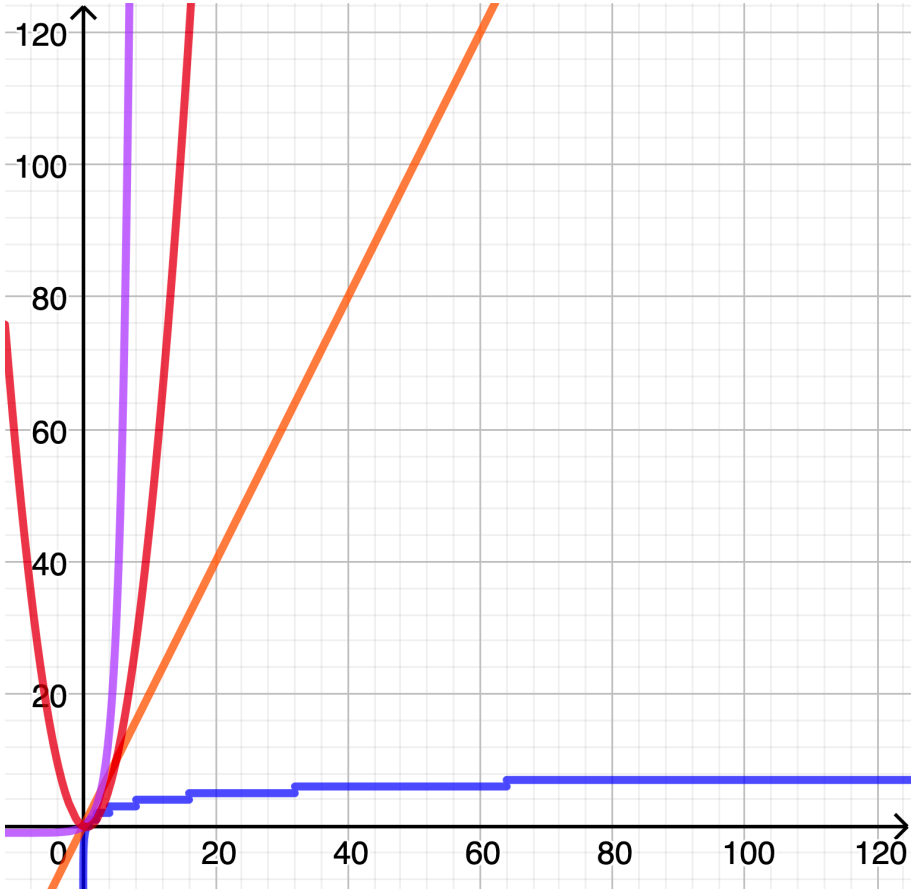
```
1 BUBBLESORT(A)
2   n = Länge(A)
3   FÜR i=n-1,…,1
4     FÜR j=0,…,i-1
5       WENN A[j]>A[j+1]
6         SWAP(A, j, j+1)
7 RETURN A
```

- Was ist hier sinnvoll als Inputgröße, was als (Rechen-)Schritte zu betrachten?
- Klar ist: Je länger das zu sortierende Array, desto länger dauert der Algorithmus. Die Inputgröße ist also hier die Länge n des übergebenen Arrays.
- Für das Verfahren werden der Reihe nach die nebeneinanderstehenden Einträge verglichen und ggfs. getauscht (`SWAP`).
- Wie oft getauscht wird, lässt sich nicht vorhersehen – aber wie oft vergleichen wird!
- Die Vergleiche stellen die Schritte des Algorithmus dar.
- Für den ersten Durchlauf der inneren Schleife finden $n - 1$ Vergleichsoperationen statt.
- Das größte Element „bubblet“ dann ganz nach rechts.
- Im nächsten Durchlauf sind dann nur noch $n - 2$ Vergleichsoperationen notwendig, usw.

Bubblesort

- Insgesamt ergibt sich so folgende Anzahl an Vergleichsoperationen:

$$S(n) = (n-1) + (n-2) + \cdots + 3 + 2 + 1 = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

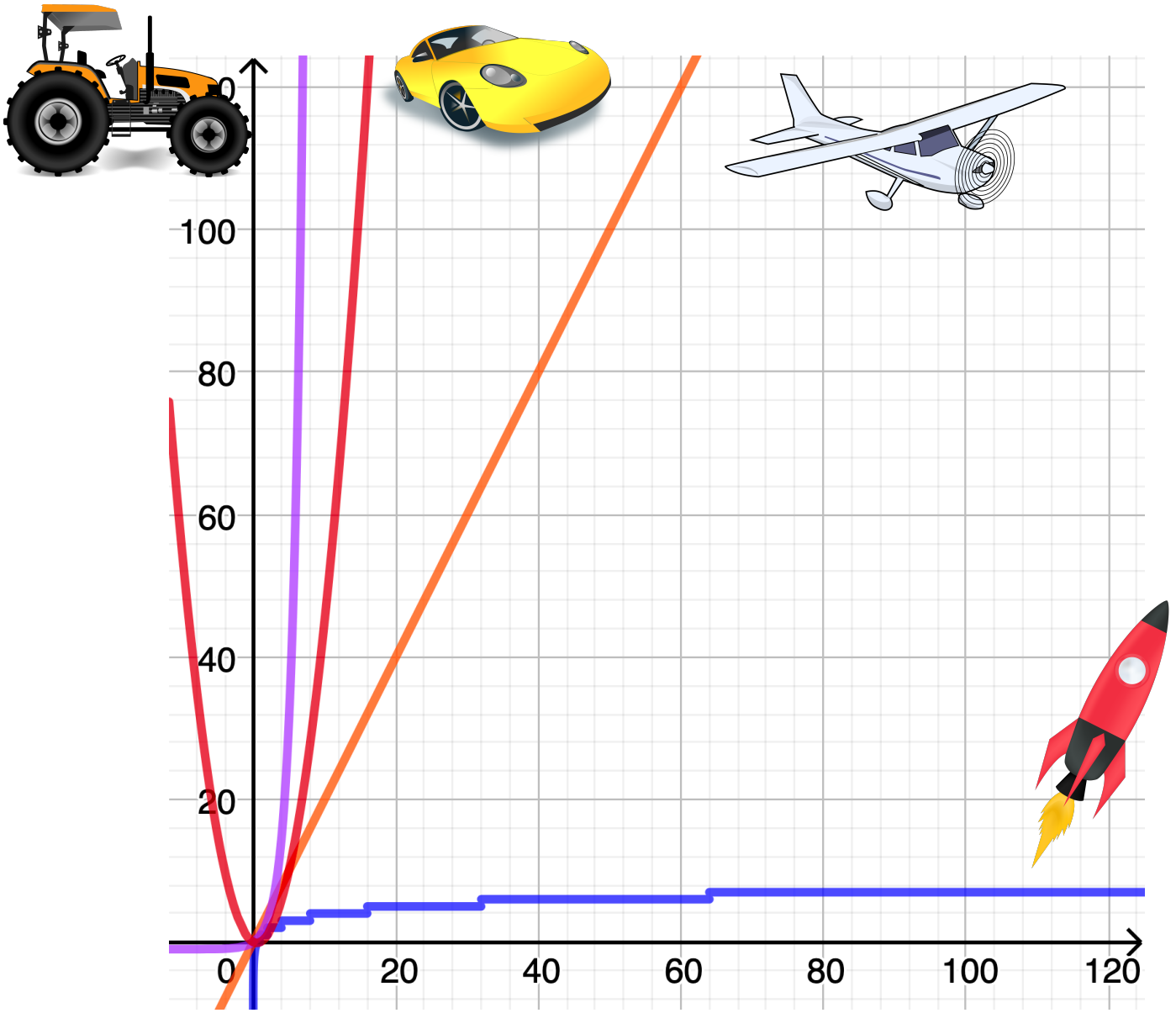


Gaußsche Summenformel
(auch *kleiner Gauß*):

$$1 + 2 + 3 + \cdots + n$$

$$= \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

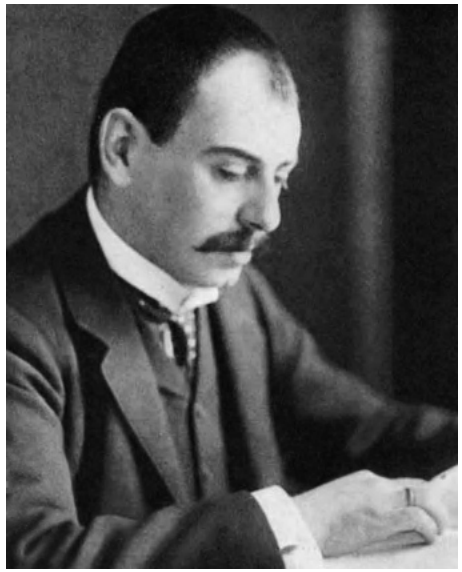
Offenbar: Unterschiedliche Geschwindigkeitsklassen



- Hierbei sind manche Verfahren zu Beginn vielleicht schneller als andere.
- Entscheidend ist, was „hinten heraus“ passiert.
- Man stelle sich vor, was passiert, wenn die Inputgröße „riesengroß“ wird.

Grafiken via http://www.pixabay.de

Landau-Notation



- Hier kommt eine spezielle Notation, die auf den deutschen Mathematiker Edmund Landau (1877–1938) zurückgeht, ins Spiel.
- Diese schafft sinnvolle „Töpfe“ für die durch verschiedene Algorithmen benötigte Anzahl von Schritten.
- Grob gesprochen:
  Wir betrachten nur den für das Laufzeitverhalten ausschlaggebenden Faktor.

Potenz berechnen:

$$S(n) = \lceil \log_2 n \rceil = O(\log_2 n)$$

Polynom auswerten:

$$S(n) = 2n = O(n)$$

Gesprochen:

Groß O von …

Türme von Hanoi lösen:

$$S(n) = 2^n - 1 = O(2^n)$$

Mit Bubblesort sortien:

$$S(n) = \frac{1}{2}n^2 - \frac{1}{2}n = O(n^2)$$

Wikimedia Commons / Public Domain

Landau-Notation

- Allgemein: Wir schreiben $S(n) = O\big(f(n)\big)$ genau dann, wenn die Werte der Funktion $S(n)$ schließlich (also ab einer bestimmten Stelle n_0) kleiner oder gleich $c \cdot f(n)$ für eine Konstante $c \in \mathbb{R}^+$ sind.
- Formal ausgedrückt:
  $S(n) = O\big(f(n)\big) \Leftrightarrow \exists n_0, c: S(n) \leq c \cdot f(n)\ \forall n \geq n_0.$
- „O“ ist eigentlich ein großes griechisches Omikron

Potenz berechnen:

$S(n) = \lceil \log_2 n \rceil$

$= O(\log_2 n)$

Konstante $c = 2$

Polynom auswerten:

$S(n) = 2n = O(n)$

Konstante $c = 2$

Logarithmischer, linearer, exponentieller und quadratischer Aufwand.

Türme von Hanoi lösen:

$S(n) = 2^n - 1 = O(2^n)$

Konstante $c = 1$

Mit Bubblesort sortien:

$S(n) = \frac{1}{2}n^2 - \frac{1}{2}n = O(n^2)$

Konstante $c = \frac{1}{2}$

Beispiel

- Ein Algorithmus berechnet von n Zahlen einen spezifischen Rückgabewert.
- Hierzu benötigt er $S(n) = 3n^3 + 2n^2 + 7$ Rechenoperationen.
- Wie lautet eine Funktion f, so dass $S(n) = O(f(n))$ gilt?
  - Wir können $S(n) = O(n^3)$ schreiben.
- Warum? Wie lautet das zugehörige c, so dass $S(n) \leq c \cdot f(n)$ gilt?
  - Konkret muss $3n^3 + 2n^2 + 7 \leq c \cdot n^3$ (ab irgendeinem hinreichend großen n_0) gelten.
  - $c = 4$ tut es z.B., aber auch $c = 3{,}1$, nicht aber $c = 3$
  - Forme hierzu z.B. um: $c \geq 3 + \frac{2}{n} + \frac{7}{n^3}$
  - Für hinreichend großes n konvergiert dieser Ausdruck gegen 3, erreicht sie aber nie.

Weitere Klassen

- Es gibt nicht nur Algorithmen, die den Klassen $O(\log_2 n)$, $O(n)$, $O(n^2)$ und $O(2^n)$ zuzuordnen sind.
- Denkbar sind beliebige Terme, die sich hinsichtlich ihres asymptotischen Verhaltens unterscheiden.
- Z.B.:
  - $O(1)$ (*konstanter Aufwand*)
  - $O(n^3)$ (*kubischer Aufwand*)
  - $O(n^k)$ mit $k \in \mathbb{R}^+$ (*polynomieller Aufwand*)
  - $O(n!)$ (*faktorieller Aufwand*)
  - $O(d^n)$ (*exponentieller Aufwand*)
- Außerdem gibt es Mischformen:
  - $O(n \cdot \log n)$ (*linearithmischer Aufwand*)

Zusammenfassung

- Die Anzahl durchzuführender (Rechen)-Schritte ist abhängig von der Inputgröße n und lässt sich für die gezeigten Beispiele anhand entsprechender Überlegungen durch einen Term abschätzen.
- Hinsichtlich ihres asymptotischen Verhaltens bzgl. des notwendigen Aufwands zur Lösung eines Problems erweisen sich die betrachteten Algorithmen als unterschiedlich.
- Dank Landaus Notation können wir alle existierenden Algorithmen in eine überschaubare Anzahl an Klassen einteilen.
- Konkret haben wir jeweils einen Algorithmus mit logarithmischem, mit linearem, mit kubischem und mit exponentiellem Aufwand gesehen.
- Es sind aber auch diverse andere Klassen denkbar.