Das große "O"

- Aufwandsklasse $O(g)$ für $g: \mathbb{N} \longrightarrow \mathbb{R}^+$ enthält alle Funktionen $f: \mathbb{N} \longrightarrow \mathbb{R}^+$ mit

$$f(n) \leq c \cdot g(n) \text{ für alle } n \geq n_0$$

  c, n_0: konstant und größer als 0

- $O(g)$ beschreibt alle Probleme, die eine algorithmische Lösung op besitzen mit $T^{op} \in O(g)$
- Obere Schranke, bei der konstante Faktoren und kleine Eingaben nicht zählen.

Das große “O”

- Idee: wähle g einfach und einprägsam, z.B.

$$\log n, n, n \cdot \log n, n^2, n^3, \ldots 2^n, \ldots$$

  - $\log n$: logarithmisch
  - n: linear
  - n^2: quadratisch
  - n^3: kubisch
  - 2^n: exponentiell

Beispiele

- $T^{mergesort} \in O(n \cdot \mathrm{ld}\, n)$
- $T^{insort} \in O(n^2)$
- $T^{xmpl}(n) = 1/3n \cdot \mathrm{ld}\, n + 22n + 1/2n^2 + 7$

$$\rightsquigarrow \quad T^{xmpl} \in O(n^2),$$

d.h. $xmpl$ hat quadratischen Aufwand.

Eigenschaften

1. $f \in O(f)$.
2. $O(f + c) = O(f)$ (c konstant).
3. $O(c * f) = O(f)$ (c konstant).
4. $f_i \in O(g_i)$ $(i = 1, 2) \Longrightarrow$ $f_1 + f_2 \in O(g_1 + g_2)$ und $f_1 * f_2 \in O(g_1 * g_2)$.
5. $O(f_1 + f_2) = O(\max(f_1, f_2))$.
6. Falls f ein Polynom vom Grad k ist, gilt $f \in O(n^k)$.
7. $f \in O(g)$ und $g \in O(h) \Longrightarrow f \in O(h)$.

Zusammenhang zwischen verschiedenen Aufwandsklassen

$$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^3) \subset \cdots \subset O(2^n) \subset O(3^n) \subset \cdots \subset O(n!) \subset O(n^n).$$

Linear, quadratisch und exponentiell im Vergleich

n	n^2	2^n
10	100	$1024 \sim 10^3$
20	400	$1048576 \sim 10^6$
30	900	$1073741824 \sim 10^9$
40	1600	$10999511627776 \sim 10^{13}$
50	2500	$1125899906842624 \sim 10^{15}$
60	3600	$1152921504600686976 \sim 10^{18}$
70	4900	$1180591620717411303424 \sim 10^{21}$

ohne Gewähr

Mit Supercomputer gegen Aufwand (?)

- Szenario: entwickle ein Programm, das bei Eingabegröße n gerade $n \mid n^2 \mid 2^n$ FLOPs benötigt und miete für 3 Stunden ($\sim 10^4$ Sek.) einen Supercomputer mit 100 Tera-FLOPs pro Sekunde (10^{14} FLOPs/Sek.).
- Welche Eingabegröße lässt sich bearbeiten?

n	n^2	2^n
10^{18}	10^9	60

Mit Supercomputer gegen Aufwand (?)

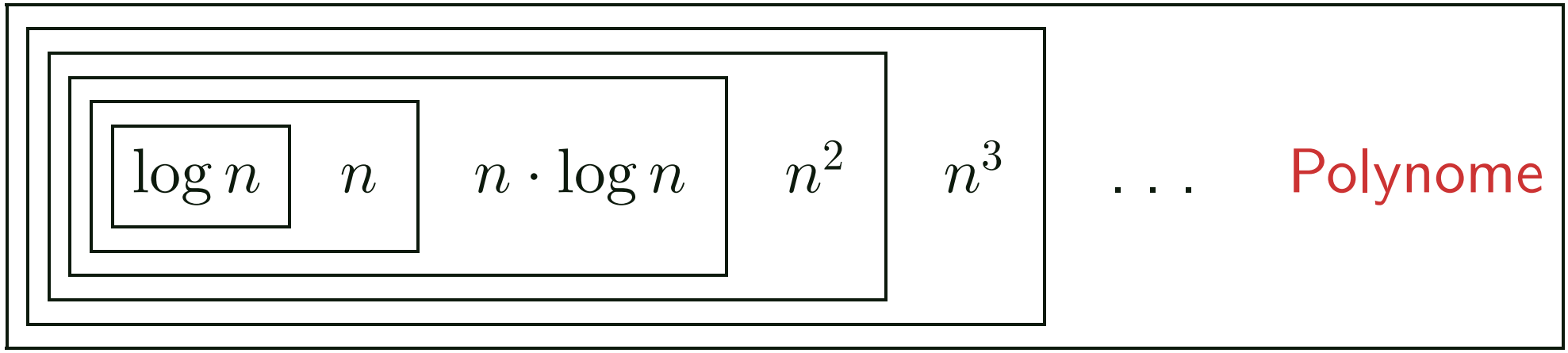
- Szenario: entwickle ein Programm, das bei Eingabegröße n gerade $n \mid n^2 \mid 2^n$ FLOPs benötigt und miete für 3 Stunden ($\sim 10^4$ Sek.) einen Supercomputer mit 100 Tera-FLOPs pro Sekunde (10^{14} FLOPs/Sek.).
- Welche Eingabegröße lässt sich bearbeiten?

n	n^2	2^n
10^{18}	10^9	60
10^{21}	$32 \cdot 10^9$	70 $(*)$

$(*)$: bei Rechenzeitverbesserung um Faktor 1000

Polynomieller Aufwand

- Probleme mit polynomiellem Aufwand können mit herkömmlichen Rechnern in verfügbarer Zeit gelöst werden (wenn der Exponent nicht zu groß ist).



Die Klassen P und NP

P enthält die Probleme, die mit herkömmlichen Computern zeitgerecht lösbar sind

NP enthält eine Vielzahl praktisch relevanter Probleme (wie Tourenplanung, Maschinenbelegung, Stundenplanung, Lagerhaltung, . . .)

Entscheidungsprobleme

Entscheidungsproblem

Abbildung der Form $dp\colon A^* \to BOOL$

Lösung von dp

CE-S-Operation $sol\colon A^* \to BOOL$, so dass für alle $w \in A^*$ gilt:

$$dp(w) = T \text{ g.d.w. } sol(w) \overset{*}{\longleftrightarrow} T$$

(aber: $dp(w) = F$ g.d.w. $sol(w) \overset{*}{\longleftrightarrow} T$ nicht gilt)

Definition von NP und P

$dp \in NP$, falls eine Lösung sol und ein $k \in \mathbb{N}$ existieren mit $T^{sol} \in O(n^k)$

$dp \in P$, falls zusätzlich für alle $w \in A^*$ gilt:

$$dp(w) = T \text{ und } sol(w) \overset{*}{\longleftrightarrow} t \text{ impl. } t \overset{*}{\longleftrightarrow} T$$

offensichtlich: $P \subseteq NP$

offen: $NP \subseteq P$

Beispiele

Problem aus P	Aufwand
Suchen in balancierten Bäumen	$\log n$
Suchen in Wörtern, Transponieren, Zählen, Filtern	n
Sortieren durch Mischen	$n \cdot \log n$
Sortieren durch Einsortieren, $quicksort$	n^2
Matrizenmultiplikation, Wortproblem kontextfreier Sprachen	n^3

Das $P = NP$-Problem

Besitzen (Entscheidungs-)Probleme mit einer **N**icht-deterministischen **P**olynomiellen Lösung immer auch eine deterministische **P**olynomielle Lösung?

- Eines der bekanntesten offenen Probleme der Informatik
- Nobelpreis-verdächtig

Anmerkungen zum Nichtdeterminismus

- Manches ist nichtdeterministisch

  Suchen in Bereichen, unsicheres und unvollständiges Wissen, Expertensysteme, Spiele

- Gleichwertigkeit ist nichtdeterministisch
- Manchmal bequem (und schadet nicht)
- Einschränken, wenn gewünscht und nötig

Beispiel: Little-Solitaire

solitaire

opns: $solitaire, won\colon \{0,1\}^* \to BOOL$

$move\colon \{0,1\}^* \to \{0,1\}^*$

vars: $u, v, w \in \{0,1\}^*$

eqns: $solitaire(w) = won(move(w))$

$won(w) = count(1, w) = 1$

$move(u110v) = move(u001v)$

$move(u011v) = move(u100v)$

$move(w) = w$