FUNKTIONSVERGLEICH

< Groß-Oh >	$f = O(g)$	„$f \leq g$“	f wächst höchstens so schnell wie g
< Klein-oh >	$f = o(g)$	„$f < g$“	f wächst langsamer als g
< Groß-Theta >	$f = \Theta(g)$	„$f = g$“	f und g wachsen größenordnungsmäßig gleich schnell
< Klein-omega >	$f = \omega(g)$	„$f > g$“	f wächst schneller als g
< Groß-Omega >	$f = \Omega(g)$	„$f \geq g$“	f wächst mindestens so schnell wie g

Anmerkungen

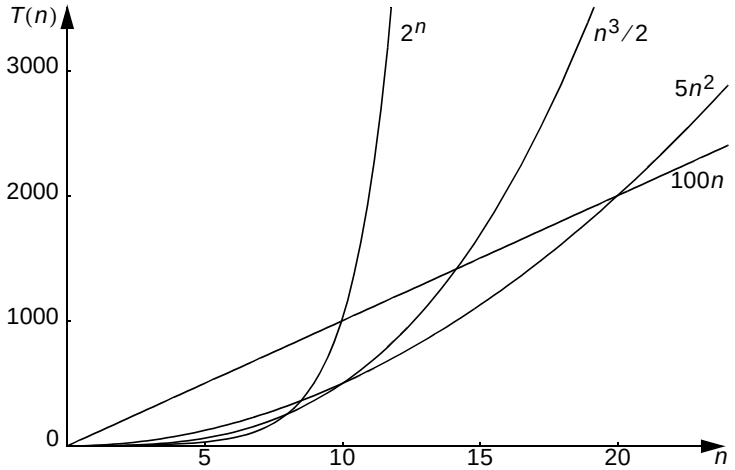
- Die “Gleichungen” dürfen nur von links nach rechts gelesen werden.
- Damit hat man praktisch so etwas wie die üblichen Vergleichsoperationen, um Funktionen größenordnungsmäßig zu vergleichen.

GÄNGIGE KLASSEN VON FUNKTIONEN

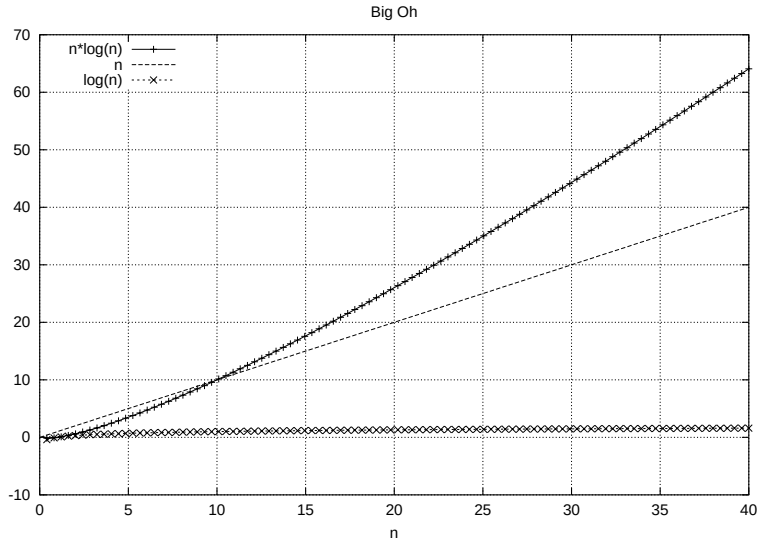
	Sprechweise	Typische Algorithmen
$O(1)$	konstante Laufzeit	liefere Kopf einer Liste
$O(\log n)$[a)]	logarithmisch	binäres Suchen auf einer Menge
$O(n^r)$, $0 < r < 1$	sublinear	
$O(n)$	linear	Bearbeiten eines jeden Elementes einer Menge; lineares Suchen
$O(n \log n)$		gute Sortierverfahren (z.B. Heapsort)
$O(n^r)$, $1 < r < 2$	subquadratisch	
$O(n^2)$	quadratisch	primitive Sortierverfahren
$O(n^3)$	kubisch	primitive Matrixmultiplikation
$O(n^k)$, $k \geq 2$	polynomial	
$O(2^n)$	exponentiell	Backtracking-Algorithmen

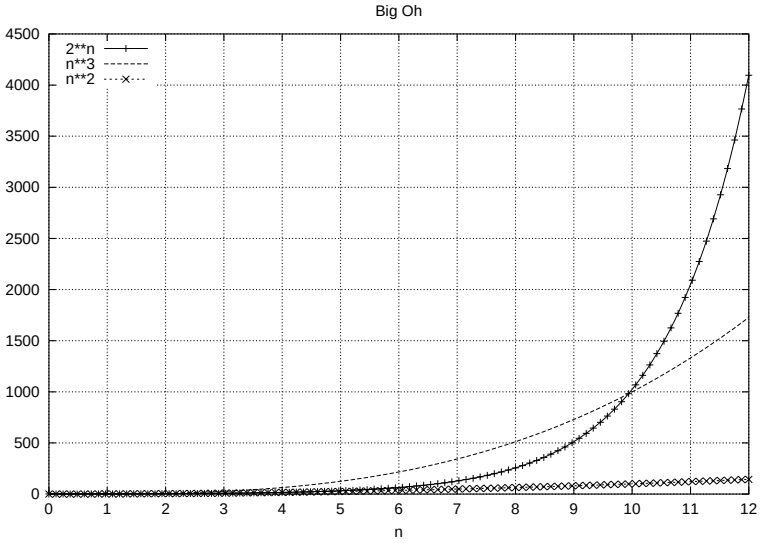
a) innerhalb der O-Notation spielt die Basis des Logarithmus keine Rolle, da Logarithmen zu verschiedenen Basen (a, b) sich nur durch einen konstanten Faktor unterscheiden; ($\log_b x = \log_b a * \log_a x$)

BEISPIEL



Laufzeit T (n)	n = max. Problemgröße für		Zuwachs bei max. Problemgröße
	10^3 sec	10^4 sec	
100 n	10	100	10.0 fache
$5\ n^2$	14	45	3.2
$n^3/2$	12	27	2.3
2^n	10	13	1.3





BERECHNUNG DER LAUFZEIT, ALLGEMEINE REGELN FÜR NICHTREKURSIVE PROGRAMME

1. $T(< \text{simple statement} >) = O(1)$

2. $T(< \text{statement sequence} >) = T_1(< \text{statement}_1 >) + \ldots + T_n(< \text{statement}_n >) = O(\text{MAX}(T_i(< \text{statement}_i >)))$

3. $T(< \text{if statement} >) = T(< \text{condition} >) + \max(T(< \text{then branch} >), T(< \text{else branch} >))$

4. $T(< \text{loop statement} >) = T(< \text{termination condition} >) + < \text{number of iterations} > * T(< \text{loop body} >)$

5. $T(< \text{procedure call} >) = T(< \text{procedure body} >)$

Anmerkungen

- diese Regeln nutzen die Wohlstrukturiertheit eines Programms aus;
- Analyse von Programmen mit unstrukturierten goto‘s
  (i. allg. Rückwärtssprünge)
  - schwierig
  - vermeiden!
- keine Probleme mit O-Abschätzung (worst-case)
  bei strukturierten goto‘s zum vorzeitigen Verlassen einer Schleife
  - break (Sprung nach endloop)
  - continue (Sprung vor endloop)

  (i. allg. Vorwärtssprünge)
- Achtung: rekursive Programme erfordern Sonderbetrachtungen!