Typische Zeitkomplexitäten

Eine kurze Zusammenfassung der wichtigsten Laufzeitkomplexitäten und in welchen praktischen Anwendungen man sie wiederfinden kann:

a) logarithmische Komplexität: O(log(n)); kommt z.B. bei der "binären Suche" vor sowie bei allen "square-and-multiply"-Verfahren; diese Komplexität ist sehr günstig, da die Laufzeit noch nicht einmal linear wächst, beispielsweise kommt es durch Quadrieren von n gerade einmal zu einer Verdopplung der Laufzeit
b) lineare Komplexität: O(n); z.B. bei der Suche in einem unsortierten Feld; diese Komplexität sollte das primäre Ziel eines Algorithmenentwurfs sein, ist aber nur selten erreichbar
c) n*log(n)-Komplexität: O(n*log(n)); die Laufzeit einiger der schnellsten Sortieralgorithmen wird dadurch charakterisiert (Quicksort etc.); ist noch sehr günstig, wenn auch nicht mehr linear
d) quadratische Komplexität: $O(n^2)$; in der Praxis charakterisierend für den Bubblesort-Algorithmus oder z.B. die Multiplikation einer Matrix mit einem Vektor; ist nicht mehr so günstig, da eine Verdopplung der Eingabedaten eine Vervierfachung der Laufzeit nach sich zieht, jedoch immernoch vertretbar
e) exponentielle Komplexität: $O(c^n)$; katastrophal schon für kleine n

Die folgende Abbildung stellt das generelle Wachstumsverhalten der wichtigsten Funktionen noch einmal grafisch dar:

In der Praxis kann beim heutigen Stand der Technik als **Faustregel** gelten: eine Komplexität bis hin zu $O(n^2)$ kann toleriert werden. Aufwände, die darüber liegen, sind bereits bei kleinen bis mittleren Eingabemengen problematisch und Aufwände mit exponentiellem Wachstum oder höher sind in der Regel nicht zu verarbeiten (sondern allenfalls in Ausnahmefällen).