Komplexität

Informatik

Die **Komplexität** eines Algorithmus bezieht sich auf die Frage, wieviel Zeit er für seine Arbeit benötigt.

Die **Komplexität** eines Algorithmus bezieht sich auf die Frage, wieviel Zeit er für seine Arbeit benötigt.

Nicht: absolute CPU-Zeit angeben
Nicht: Anzahl der Maschinenbefehle zählen

Sondern: Wachstum der Laufzeit in Abhängigkeit von der Größe der Eingabe

Größe der Eingabe kann z.B. sein: Zahl der Daten, Wert der Eingabezahl, Länge der Codierung.

Uns interessiert die Größenordnung dieser Änderung. Dazu nutzen wir die O-Notation.

$f(n) \in O(n^2) \Rightarrow$ wenn sich die Eingabegröße verdoppelt,

Uns interessiert die Größenordnung dieser Änderung. Dazu nutzen wir die O-Notation.

$f(n) \in O(n^2) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, vervierfacht sich (ungefähr) die Laufzeit. Wenn sich die Eingabegröße verdreifacht,

Uns interessiert die Größenordnung dieser Änderung. Dazu nutzen wir die O-Notation.

$f(n) \in O(n^2) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, vervierfacht sich (ungefähr) die Laufzeit. Wenn sich die Eingabegröße verdreifacht, verneunfacht sich die Laufzeit.

$f(n) \in O(n) \Rightarrow$ wenn sich die Eingabegröße verdoppelt,

Uns interessiert die Größenordnung dieser Änderung. Dazu nutzen wir die O-Notation.

$f(n) \in O(n^2) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, vervierfacht sich (ungefähr) die Laufzeit. Wenn sich die Eingabegröße verdreifacht, verneunfacht sich die Laufzeit.

$f(n) \in O(n) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, verdoppelt sich die Laufzeit.

$f(n) \in O(2^n) \Rightarrow$ wenn sich die Eingabegröße um 10 erhöht,

Uns interessiert die Größenordnung dieser Änderung. Dazu nutzen wir die O-Notation.

$f(n) \in O(n^2) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, vervierfacht sich (ungefähr) die Laufzeit. Wenn sich die Eingabegröße verdreifacht, verneunfacht sich die Laufzeit.

$f(n) \in O(n) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, verdoppelt sich die Laufzeit.

$f(n) \in O(2^n) \Rightarrow$ wenn sich die Eingabegröße um 10 erhöht, vertausendfacht (≈ 1024) sich die Laufzeit.

$f(n) \in O(\log n) \Rightarrow$ wenn sich die Eingabegröße verdoppelt,

Uns interessiert die Größenordnung dieser Änderung. Dazu nutzen wir die O-Notation.

$f(n) \in O(n^2) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, vervierfacht sich (ungefähr) die Laufzeit. Wenn sich die Eingabegröße verdreifacht, verneunfacht sich die Laufzeit.

$f(n) \in O(n) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, verdoppelt sich die Laufzeit.

$f(n) \in O(2^n) \Rightarrow$ wenn sich die Eingabegröße um 10 erhöht, vertausendfacht (≈ 1024) sich die Laufzeit.

$f(n) \in O(\log n) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, erhöht sich die Laufzeit um 1.

$f(n) \in O(1) \Rightarrow$

Uns interessiert die Größenordnung dieser Änderung. Dazu nutzen wir die O-Notation.

$f(n) \in O(n^2) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, vervierfacht sich (ungefähr) die Laufzeit. Wenn sich die Eingabegröße verdreifacht, verneunfacht sich die Laufzeit.

$f(n) \in O(n) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, verdoppelt sich die Laufzeit.

$f(n) \in O(2^n) \Rightarrow$ wenn sich die Eingabegröße um 10 erhöht, vertausendfacht (≈ 1024) sich die Laufzeit.

$f(n) \in O(\log n) \Rightarrow$ wenn sich die Eingabegröße verdoppelt, erhöht sich die Laufzeit um 1.

$f(n) \in O(1) \Rightarrow$ die Laufzeit ist unabhängig von der Größe der Eingabe.

Im O-Kalkül interessiert nur die Funktion, die bei großem n das Wachstum dominiert. Faktoren und weniger schnell wachsende Terme werden vernachlässigt.

$f(n) = 3n^2 + 4n + 7 \in$

Im O-Kalkül interessiert nur die Funktion, die bei großem n das Wachstum dominiert. Faktoren und weniger schnell wachsende Terme werden vernachlässigt.

$f(n) = 3n^2 + 4n + 7 \in O(n^2)$
$f(n) = 1000n^2 \in$

Im O-Kalkül interessiert nur die Funktion, die bei großem n das Wachstum dominiert. Faktoren und weniger schnell wachsende Terme werden vernachlässigt.

$f(n) = 3n^2 + 4n + 7 \in O(n^2)$
$f(n) = 1000n^2 \in O(n^2)$
$f(n) = 20n + 4 \cdot 2^n \in$

Im O-Kalkül interessiert nur die Funktion, die bei großem n das Wachstum dominiert. Faktoren und weniger schnell wachsende Terme werden vernachlässigt.

$f(n) = 3n^2 + 4n + 7 \in O(n^2)$
$f(n) = 1000n^2 \in O(n^2)$
$f(n) = 20n + 4 \cdot 2^n \in O(2^n)$
$f(n) = n + 2 \cdot \log n \in$

Im O-Kalkül interessiert nur die Funktion, die bei großem n das Wachstum dominiert. Faktoren und weniger schnell wachsende Terme werden vernachlässigt.

$f(n) = 3n^2 + 4n + 7 \in O(n^2)$
$f(n) = 1000n^2 \in O(n^2)$
$f(n) = 20n + 4 \cdot 2^n \in O(2^n)$
$f(n) = n + 2 \cdot \log n \in O(n)$

In der Informatik nutzen wir standardmäßig den Logarithmus zur Basis 2. $\log x = \log_2 x =$ die Zahl, hoch die ich 2 nehmen muss, um zu x zu gelangen.

In der Informatik nutzen wir standardmäßig den Logarithmus zur Basis 2.
$\log x = \log_2 x =$ die Zahl, hoch die ich 2 nehmen muss, um zu x zu gelangen.

$\log 1 = 0$ denn $2^0 = 1$
$\log 2 = 1$ denn $2^1 = 2$
$\log 4 = 2$ denn $2^2 = 4$
$\log 8 = 3$ denn $2^3 = 8$

Wenn sich das Argument verdoppelt,

In der Informatik nutzen wir standardmäßig den Logarithmus zur Basis 2.
$\log x = \log_2 x =$ die Zahl, hoch die ich 2 nehmen muss, um zu x zu gelangen.

$\log 1 = 0$ denn $2^0 = 1$
$\log 2 = 1$ denn $2^1 = 2$
$\log 4 = 2$ denn $2^2 = 4$
$\log 8 = 3$ denn $2^3 = 8$

Wenn sich das Argument verdoppelt, erhöht sich der Logarithmus um 1
Wenn sich das Argument halbiert,

In der Informatik nutzen wir standardmäßig den Logarithmus zur Basis 2.
$\log x = \log_2 x =$ die Zahl, hoch die ich 2 nehmen muss, um zu x zu gelangen.

$\log 1 = 0$ denn $2^0 = 1$
$\log 2 = 1$ denn $2^1 = 2$
$\log 4 = 2$ denn $2^2 = 4$
$\log 8 = 3$ denn $2^3 = 8$

Wenn sich das Argument verdoppelt, erhöht sich der Logarithmus um 1
Wenn sich das Argument halbiert, erniedrigt sich der Logarithmus um 1.

Der ganzzahlige Logarithmus gibt an, wie oft ich eine Zahl halbieren muss, um zu einer 1 vor dem Komma zu gelangen.

Wenn ein Algorithmus in der Komplexitätsklasse $O(\log n)$ ist und die Eingabegröße wird vertausendfacht, dann wächst die Laufzeit um

In der Informatik nutzen wir standardmäßig den Logarithmus zur Basis 2.
$\log x = \log_2 x =$ die Zahl, hoch die ich 2 nehmen muss, um zu x zu gelangen.

$\log 1 = 0$ denn $2^0 = 1$
$\log 2 = 1$ denn $2^1 = 2$
$\log 4 = 2$ denn $2^2 = 4$
$\log 8 = 3$ denn $2^3 = 8$

Wenn sich das Argument verdoppelt, erhöht sich der Logarithmus um 1
Wenn sich das Argument halbiert, erniedrigt sich der Logarithmus um 1.

Der ganzzahlige Logarithmus gibt an, wie oft ich eine Zahl halbieren muss, um zu einer 1 vor dem Komma zu gelangen.

Wenn ein Algorithmus in der Komplexitätsklasse $O(\log n)$ ist und die Eingabegröße wird vertausendfacht, dann wächst die Laufzeit um 10 Schritte.

In der Informatik nutzen wir standardmäßig den Logarithmus zur Basis 2.
$\log x = \log_2 x =$ die Zahl, hoch die ich 2 nehmen muss, um zu x zu gelangen.

$\log 1 = 0$ denn $2^0 = 1$
$\log 2 = 1$ denn $2^1 = 2$
$\log 4 = 2$ denn $2^2 = 4$
$\log 8 = 3$ denn $2^3 = 8$

Wenn sich das Argument verdoppelt, erhöht sich der Logarithmus um 1
Wenn sich das Argument halbiert, erniedrigt sich der Logarithmus um 1.

Der ganzzahlige Logarithmus gibt an, wie oft ich eine Zahl halbieren muss, um zu einer 1 vor dem Komma zu gelangen.

Wenn ein Algorithmus in der Komplexitätsklasse $O(\log n)$ ist und die Eingabegröße wird vertausendfacht, dann wächst die Laufzeit um 10 Schritte. Wächst die Eingabegröße um das millionenfache, dann wächst die Laufzeit um

In der Informatik nutzen wir standardmäßig den Logarithmus zur Basis 2.
$\log x = \log_2 x =$ die Zahl, hoch die ich 2 nehmen muss, um zu x zu gelangen.

$\log 1 = 0$ denn $2^0 = 1$
$\log 2 = 1$ denn $2^1 = 2$
$\log 4 = 2$ denn $2^2 = 4$
$\log 8 = 3$ denn $2^3 = 8$

Wenn sich das Argument verdoppelt, erhöht sich der Logarithmus um 1
Wenn sich das Argument halbiert, erniedrigt sich der Logarithmus um 1.

Der ganzzahlige Logarithmus gibt an, wie oft ich eine Zahl halbieren muss, um zu einer 1 vor dem Komma zu gelangen.

Wenn ein Algorithmus in der Komplexitätsklasse $O(\log n)$ ist und die Eingabegröße wird vertausendfacht, dann wächst die Laufzeit um 10 Schritte. Wächst die Eingabegröße um das millionenfache, dann wächst die Laufzeit um 20 Schritte.

Annahme: 1 Schritt dauert $1\mu s = 0.000001s$

$n =$	10	20	30	40	50	60
$\log n$	$3.3\mu s$	$4.3\mu s$	$4.9\mu s$	$5.3\mu s$	$5.6\mu s$	$5.9\mu s$

Annahme: 1 Schritt dauert $1\mu s = 0.000001s$

$n =$	10	20	30	40	50	60
$\log n$	$3.3\mu s$	$4.3\mu s$	$4.9\mu s$	$5.3\mu s$	$5.6\mu s$	$5.9\mu s$
n	$10\mu s$	$20\mu s$	$30\mu s$	$40\mu s$	$50\mu s$	$60\mu s$

Annahme: 1 Schritt dauert $1\mu s = 0.000001s$

$n =$	10	20	30	40	50	60
$\log n$	$3.3\mu s$	$4.3\mu s$	$4.9\mu s$	$5.3\mu s$	$5.6\mu s$	$5.9\mu s$
n	$10\mu s$	$20\mu s$	$30\mu s$	$40\mu s$	$50\mu s$	$60\mu s$
$n \cdot \log n$	$33\mu s$	$86\mu s$	$147\mu s$	$212\mu s$	$280\mu s$	$354\mu s$

Annahme: 1 Schritt dauert $1\mu s = 0.000001s$

$n =$	10	20	30	40	50	60
$\log n$	$3.3\mu s$	$4.3\mu s$	$4.9\mu s$	$5.3\mu s$	$5.6\mu s$	$5.9\mu s$
n	$10\mu s$	$20\mu s$	$30\mu s$	$40\mu s$	$50\mu s$	$60\mu s$
$n \cdot \log n$	$33\mu s$	$86\mu s$	$147\mu s$	$212\mu s$	$280\mu s$	$354\mu s$
n^2	$100\mu s$	$400\mu s$	$900\mu s$	$1.6ms$	$2.5ms$	$3.6ms$

Annahme: 1 Schritt dauert $1\mu s = 0.000001s$

$n =$	10	20	30	40	50	60
$\log n$	$3.3\mu s$	$4.3\mu s$	$4.9\mu s$	$5.3\mu s$	$5.6\mu s$	$5.9\mu s$
n	$10\mu s$	$20\mu s$	$30\mu s$	$40\mu s$	$50\mu s$	$60\mu s$
$n \cdot \log n$	$33\mu s$	$86\mu s$	$147\mu s$	$212\mu s$	$280\mu s$	$354\mu s$
n^2	$100\mu s$	$400\mu s$	$900\mu s$	$1.6ms$	$2.5ms$	$3.6ms$
n^3	$1ms$	$8ms$	$27ms$	$64ms$	$125ms$	$216ms$

Annahme: 1 Schritt dauert $1\mu s = 0.000001s$

$n =$	10	20	30	40	50	60
$\log n$	$3.3\mu s$	$4.3\mu s$	$4.9\mu s$	$5.3\mu s$	$5.6\mu s$	$5.9\mu s$
n	$10\mu s$	$20\mu s$	$30\mu s$	$40\mu s$	$50\mu s$	$60\mu s$
$n \cdot \log n$	$33\mu s$	$86\mu s$	$147\mu s$	$212\mu s$	$280\mu s$	$354\mu s$
n^2	$100\mu s$	$400\mu s$	$900\mu s$	$1.6ms$	$2.5ms$	$3.6ms$
n^3	$1ms$	$8ms$	$27ms$	$64ms$	$125ms$	$216ms$
2^n	$1ms$	$1s$	$18min$	13 Tage	$36J$	$366Jh$

Annahme: 1 Schritt dauert $1\mu s = 0.000001s$

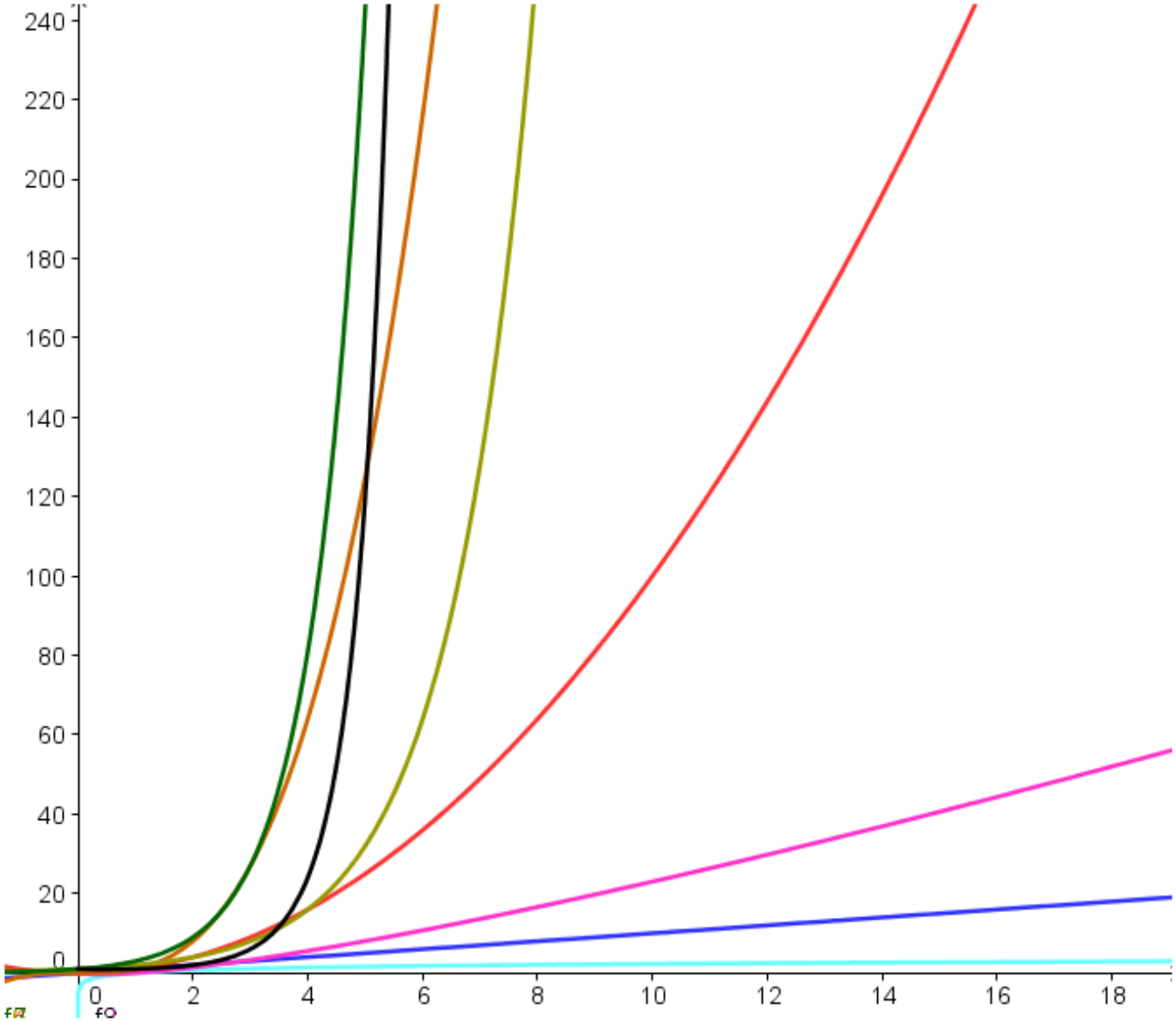
$n =$	10	20	30	40	50	60
$\log n$	$3.3\mu s$	$4.3\mu s$	$4.9\mu s$	$5.3\mu s$	$5.6\mu s$	$5.9\mu s$
n	$10\mu s$	$20\mu s$	$30\mu s$	$40\mu s$	$50\mu s$	$60\mu s$
$n \cdot \log n$	$33\mu s$	$86\mu s$	$147\mu s$	$212\mu s$	$280\mu s$	$354\mu s$
n^2	$100\mu s$	$400\mu s$	$900\mu s$	$1.6ms$	$2.5ms$	$3.6ms$
n^3	$1ms$	$8ms$	$27ms$	$64ms$	$125ms$	$216ms$
2^n	$1ms$	$1s$	$18min$	13 Tage	$36J$	$366Jh$
3^n	$59ms$	$58min$	$6,5J$	$3855Jh$	$10^8 Jh$	$10^{13} Jh$

Annahme: 1 Schritt dauert $1\mu s = 0.000001s$

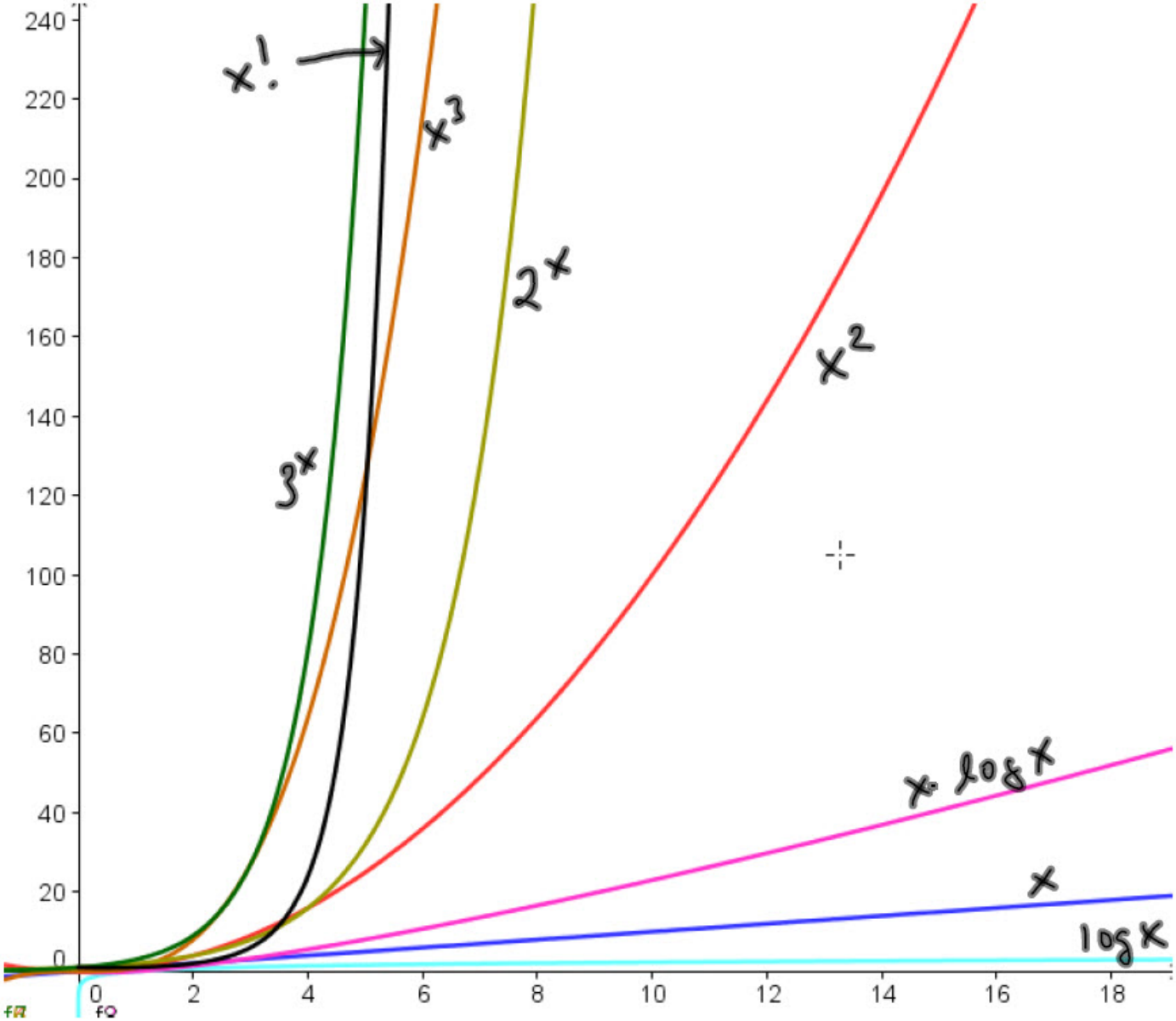
$n =$	10	20	30	40	50	60
$\log n$	$3.3\mu s$	$4.3\mu s$	$4.9\mu s$	$5.3\mu s$	$5.6\mu s$	$5.9\mu s$
n	$10\mu s$	$20\mu s$	$30\mu s$	$40\mu s$	$50\mu s$	$60\mu s$
$n \cdot \log n$	$33\mu s$	$86\mu s$	$147\mu s$	$212\mu s$	$280\mu s$	$354\mu s$
n^2	$100\mu s$	$400\mu s$	$900\mu s$	$1.6ms$	$2.5ms$	$3.6ms$
n^3	$1ms$	$8ms$	$27ms$	$64ms$	$125ms$	$216ms$
2^n	$1ms$	$1s$	$18min$	13 Tage	$36J$	$366Jh$
3^n	$59ms$	$58min$	$6,5J$	$3855Jh$	$10^8 Jh$	$10^{13} Jh$
$n!$	$6.62s$	$771Jh$	$10^{16} Jh$	$10^{32} Jh$	$10^{49} Jh$	$10^{66} Jh$

Alter des Universums: ca. 15 Mrd. Jahre $= 1,5 \cdot 10^9 J = 1,5 \cdot 10^7 Jh$

Algorithmen, die exponentielle oder stärkere Laufzeiten haben, sind in der Regel nicht praktikabel (nicht effizient).



Wachstumskurven für $x, x^2, x^3, \log(x), x\log(x), 2^x, 3^x, x!$



Wachstumskurven für $x, x^2, x^3, \log(x), x\log(x), 2^x, 3^x, x!$

Beispiele für Komplexitätsklassen

$O(\log n)$	binäre Suche
$O(n)$	lineare Suche
$O(n \cdot \log n)$	schlaues Sortieren
$O(n^2)$	dummes Sortieren
$O(2^n)$	alle Teilmengen
$O(n!)$	alle Permutationen

Analoge Komplexitätsaussagen sind möglich bzgl. Speicherplatz.
Wir zählen nicht die Anzahl der Bytes, sondern betrachten das Wachstum des Platzbedarfs in Abhängigkeit von der Größe der Eingabe.

Analoge Komplexitätsaussagen sind möglich bzgl. Speicherplatz.
Wir zählen nicht die Anzahl der Bytes, sondern betrachten das Wachstum des Platzbedarfs in Abhängigkeit von der Größe der Eingabe.

Aussagen über Laufzeit und Platzbedarf sind möglich für den

- best case
- worst case
- average case

Beispiel1 : Minimumsuche in einer Liste a der Länge n.

```
def minimum(a):
```

Beispiel1 : Minimumsuche in einer Liste a der Länge n.

```
def minimum(a):
    min = a[0]
    for i in range(len(a)):
        if a[i] < min:
            min = a[i]
    return min
```

best case:

Beispiel1 : Minimumsuche in einer Liste a der Länge n.

```
def minimum(a):
    min = a[0]
    for i in range(len(a)):
        if a[i] < min:
            min = a[i]
    return min
```

best case: $O(n)$ worst case: $O(n)$ average case: $O(n)$

Beispiel2 : Gibt es doppelte Elemente in einer Liste a der Länge n?

```
def doppelte(a):
```

Beispiel2 : Gibt es doppelte Elemente in einer Liste a der Länge n?

```
def doppelte(a):
    n = len(a)
    for i in range(n-1):
        for j in range(i+1,n):
            if a[i] == a[j]:
                return True
    return False
```

best case:

Beispiel2 : Gibt es doppelte Elemente in einer Liste a der Länge n?

```
def doppelte(a):
    n = len(a)
    for i in range(n-1):
        for j in range(i+1,n):
            if a[i] == a[j]:
                return True
    return False
```

best case: $O(1)$ worst case: $O(n^2)$