Letzte Lektion: Algorithmen

Es gibt nie nur die eine perfekte Datenstruktur und den einen perfekten Algorithmus
für die Lösung eines Problems.

Wanted:

Eine **allgemeine Sprache** für einen formalen Weg,
die **Zeitkomplexität** konkurrierender Datenstrukturen und Algorithmen auszudrücken und zu vergleichen.

Eine Größen-Ordnung für den Vergleich der Schritte – Gross O...

Big O (Groeßen-Ordnung)

Konzept aus der Welt der Mathematiker, die:

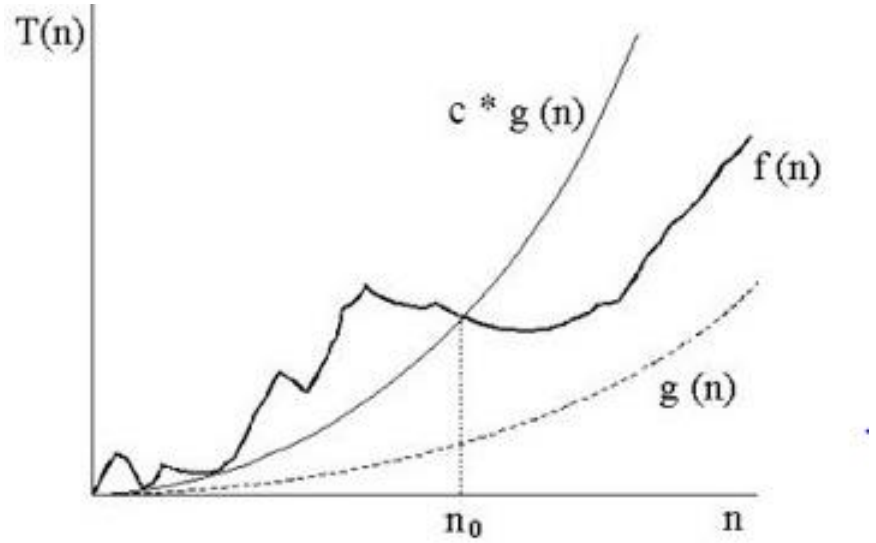
Big-O-Notation! *(oder Groß-O-Notation)*

= Anzahl von Schritten pro Inputdaten (worst case, obere Schranke)

Bsp.: Die Effizienz einer linearen Suche feststellen: **max. N** Schritte für n Datenelemente des Arrays: f(n) = O(N)

f(N) = O(g(N)), wenn gilt: f(N) <= c*g(N) für alle N >= N0

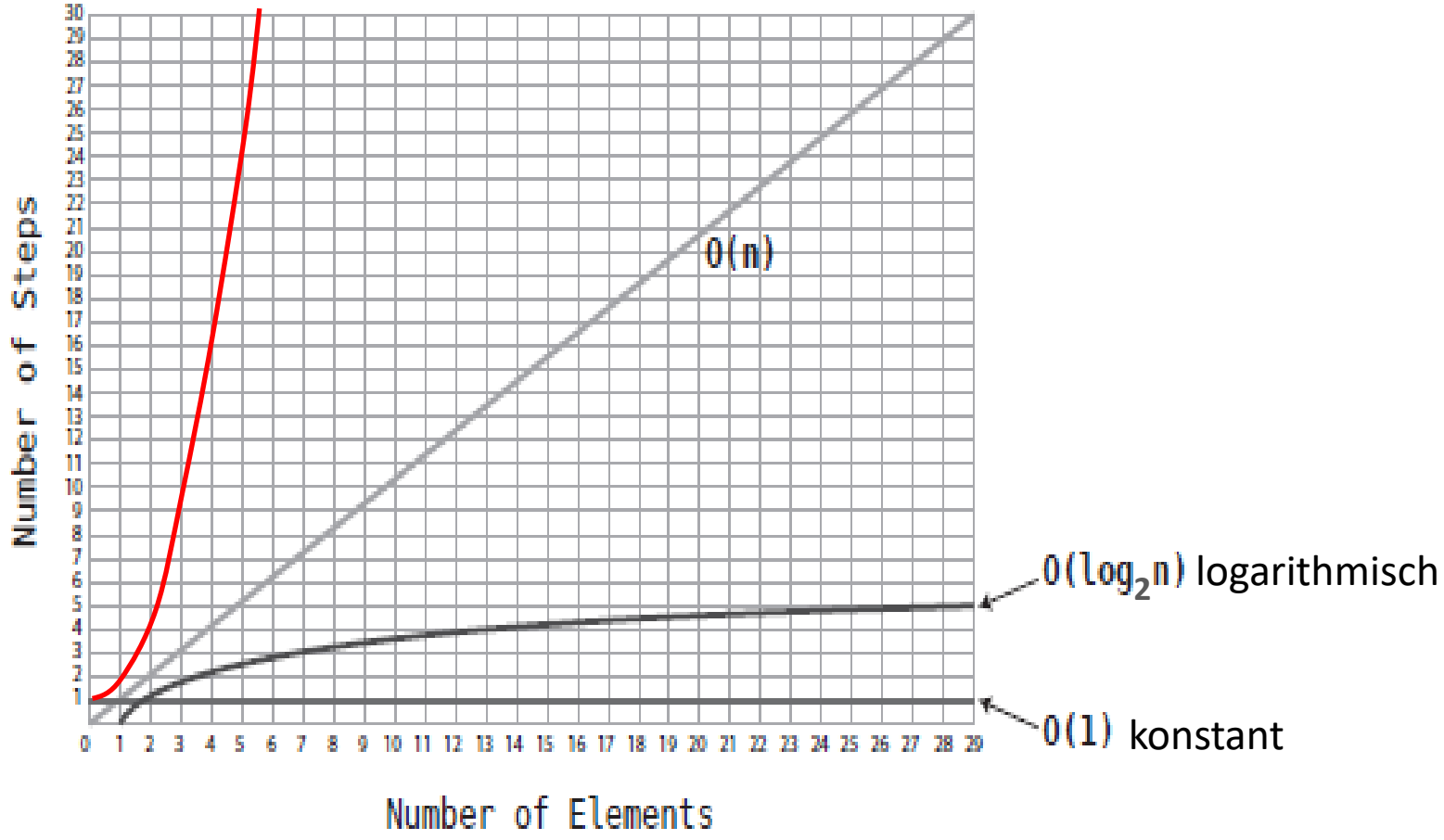
heißt: f(N) wächst nicht schneller als g(N)



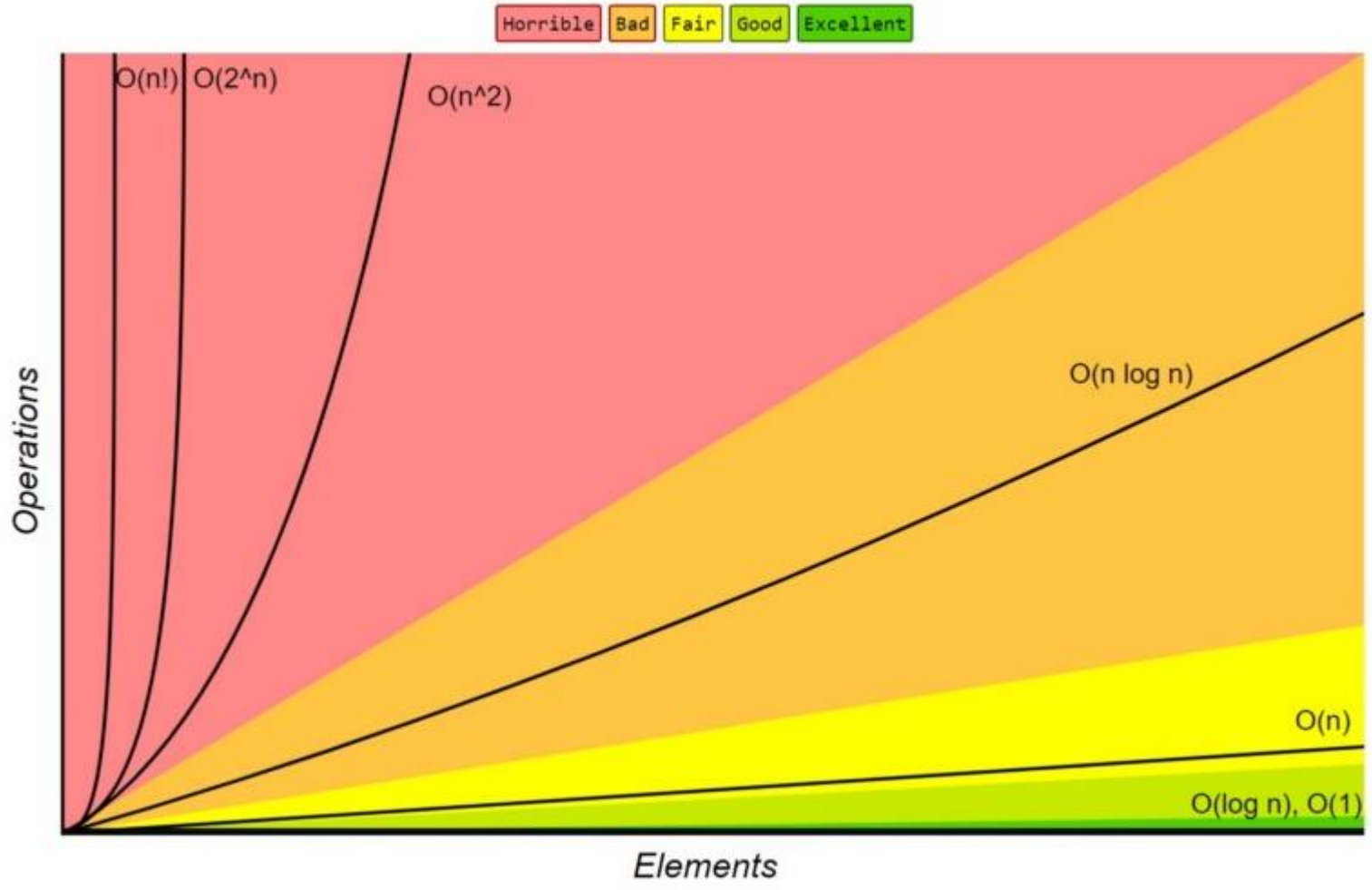
Big O – Größenordnungen (Beispiele)

$O(n^2)$, quadratisch

linear, proportional

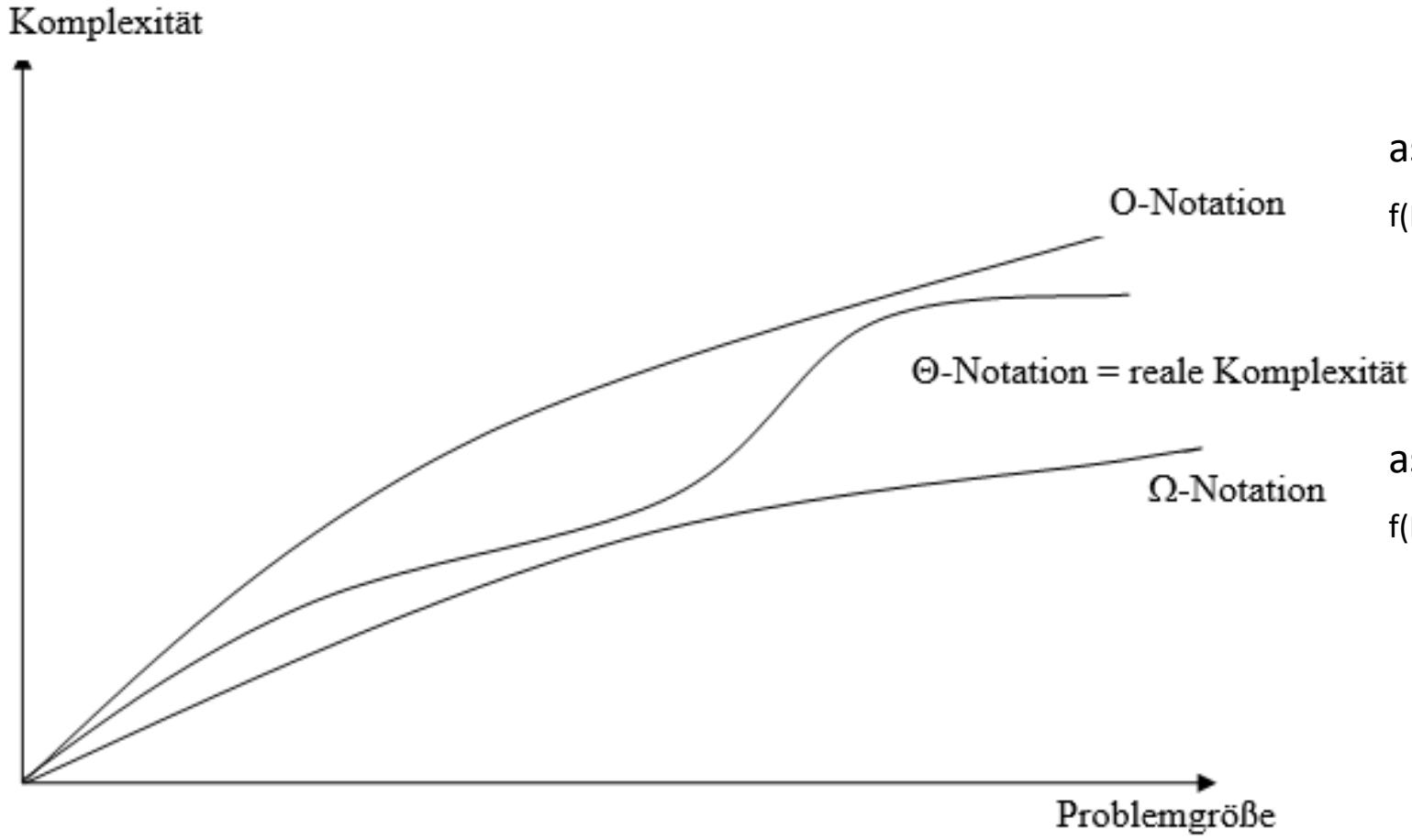


Big-O Komplexitätschart



Complexity Growth Illustration from Big O Cheatsheet

Komplexitätsnotationen (Landau-Symbole)



asymptotische obere Schranke

f(N) = O(g(N)), wenn:

f(N) <= c*g(N) für alle N >= N_0

asymptotische untere Schranke

f(N) = Ω(g(N)), wenn:

f(N) >= c*g(N) für alle N >= N_0

Big O

- Statt sich auf Zeiteinheiten zu fokussieren, erreicht Big-O Vergleichbarkeit durch das Fokussieren auf die *Anzahl der Schritte,* die ein Algorithmus braucht (im **worst case**).
- Ein “pessimistischer” (“worst case”) Ansatz kann dabei ein nützliches Tool für die Algorithmenauswahl sein

- **Lineare Suche (Worst case):** **N** Schritte (Vergleiche) bei n Elementen
  = O (N) - Big O in der Größen-**O**rdnung von N bei n Elementen
  = bei n Datenelementen, wieviele Schritte braucht der Algorithmus?

O(N): lineare Zeit = lineare Komplexität = lineare Kosten (linear = proportional zur Menge der Elemente)

- **Direktzugriff auf ein Arrayelement:** 1 Schritt, unabhängig von der Zahl der Arrayelemente
  = O (1)

O(1): konstante Zeit = schnellster Algorithmus, unabhängig von der Menge der Elemente

Die Seele von Big O

Big O: ***Bei n Datenelementen, wieviele Schritte braucht der Algorithmus (worst case)?***

- Nehmen wir mal an, wir hätten einen Algorithmus, der immer 3 Schritte benötigt, unabhängig von der Menge der Inputdaten. Wie würde man das mit Big O ausdrücken?
- Wir würden sagen: O(3).
- Es ist jedoch: die Größenordnung O(1)

Big O: *Wie ändert sich die Performance eines Algorithmus relativ zur Menge der Elemente?*

- *bleibt konstant?* *O(1)*
- *wächst logarithmisch?*
- *wächst linear?* *O(n)*
- *wächst exponentiell?*

Big O für Binäre Suche

- … ist nicht O(1), weil sich die Anzahl der Schritte mit der Anzahl der Elemente ändert
- … ist nicht O(N), weil die Anzahl der Schritte deutlich geringer ist als die der Elemente
- Binäre Suche hat eine Komplexität von: $O(\log_2 N)$

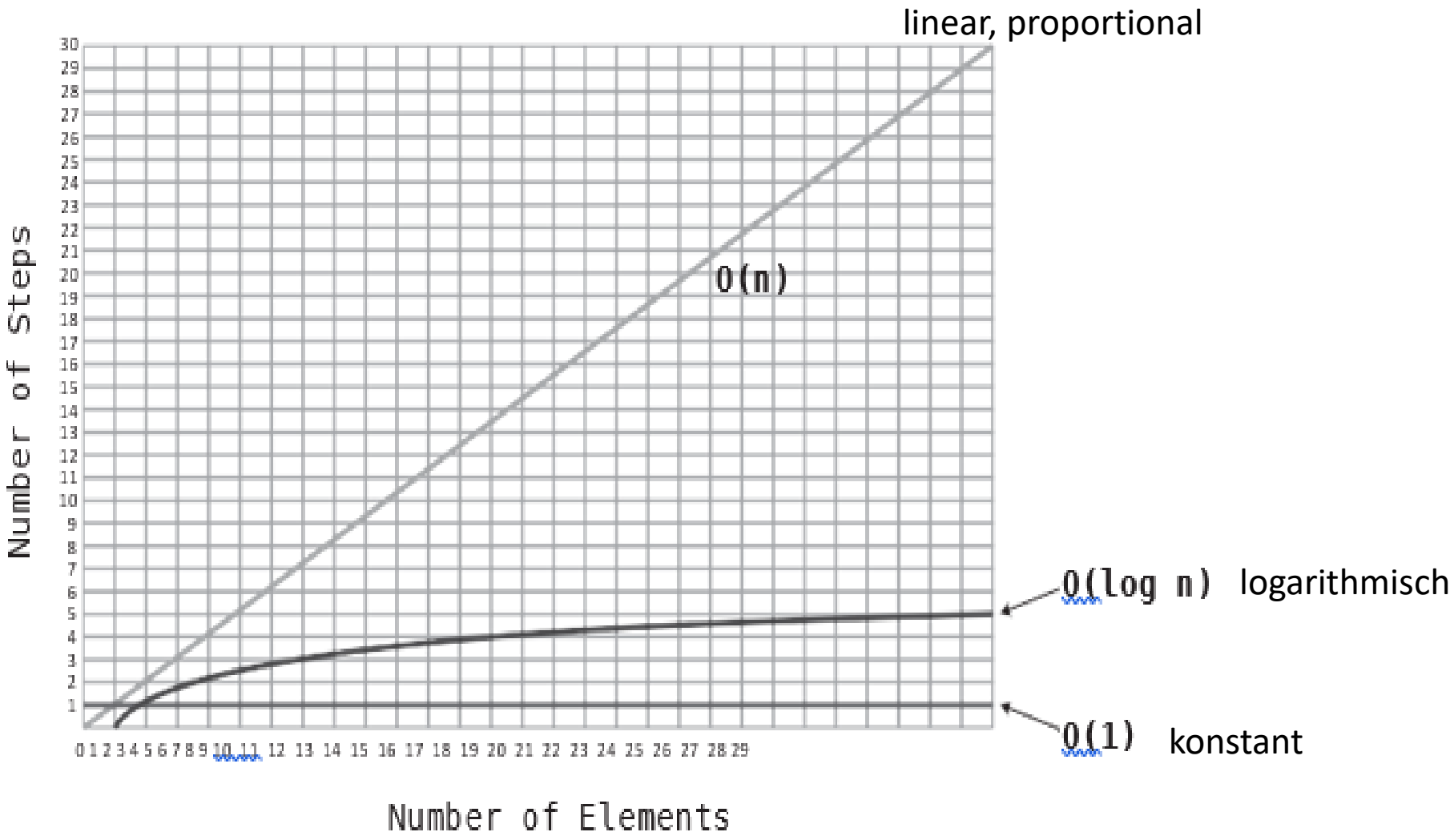
  = Big O in der Größenordnung von $\log_2 N$

  = der Algorithmus benötigt einen Schritt mehr pro Verdopplung der Elemente

Big O: *Wie ändert sich die Performance eines Algorithmus relativ zur Menge der Elemente?*

- *bleibt konstant?* *O(1)*
- *wächst logarithmisch?* *O (log N)*
- *wächst linear?* *O(N)*
- *wächst exponentiell?*

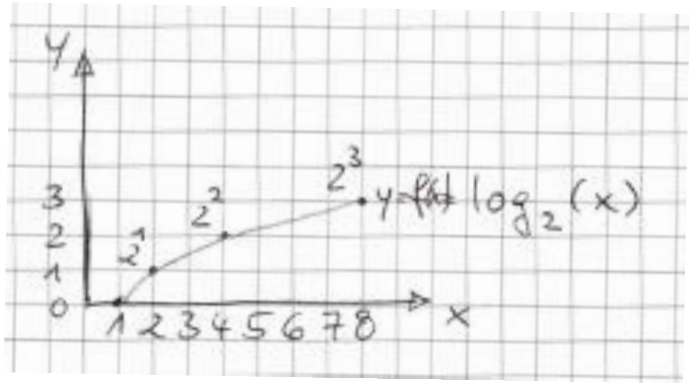
Big O

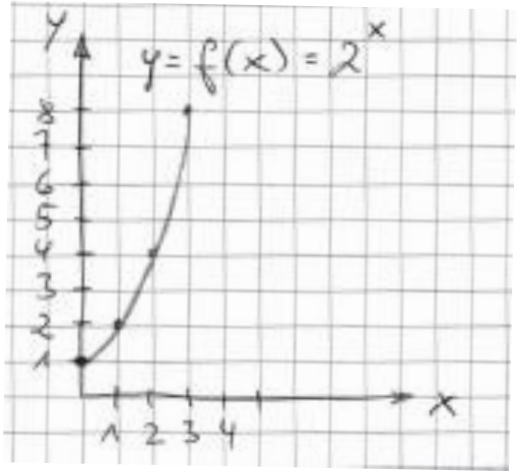

linear, proportional
O(n)
O(log n) logarithmisch
O(1) konstant
Number of Steps
Number of Elements
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Logarithmus?

- Der Logarithmus ist invers zur Exponentialfunktion
- **2^3** (Exponentialfunktion) ist : 2 * 2 * 2 = **8**
- **$\log_2(8)$** (Logarithmus) = wie oft muss ich 2 mit sich selbst multiplizieren um 8 zu erhalten? **= 3**

  = wieviele Bits braucht man für 8 verschiedene binäre Zahlen (0…7)? **= 3**





- **2^6** = 2 * 2 * 2 * 2 * 2 * 2 = **64**
- **$\log_2(64)$** = wie oft muss ich 2 mit sich selbst multiplizieren um 64 zu erhalten? **= 6**

  = wieviele Bits braucht man für 64 verschiedene binäre Zahlen (0…63)? **= 6**

Logarithmus? Anderer Versuch der Erklärung…

$\log_2 8$: wenn wir 8 solange *durch* 2 teilen bis wir 1 erhalten, wieviele 2en haben wir in der Gleichung?

- 8 / 2 / 2 / 2 = 1

In anderen Worten: wie oft müssen wir 8 halbieren bis wir 1 erhalten? In diesem Beispiel: **3 mal**. Deshalb:

- $\log_2 8 = 3$

$\log_2 64$: wenn wir 64 solange *durch* 2 teilen bis wir 1 erhalten, wieviele 2en haben wir in der Gleichung?

- 64 / 2 / 2 / 2/ 2/ 2/ 2 = 1

In anderen Worten: wie oft müssen wir 64 halbieren bis wir 1 erhalten? In diesem Beispiel: **6 mal**. Deshalb:

- $\log_2 64 = 6$

Sie erinnern sich an den Seerosenteich?

O (log N) erklärt…

- O (log N) ist die verkürzte Ausdrucksform für $O(\log_2 N)$
- O (log N) heißt für N Datenelemente, dass der Algorithmus $\log_2 N$ Schritte braucht.
- Wäre N=8 Elemente, würde der Algorithmus 3 Schritte brauchen, weil $\log_2 8 = 3$

Anders ausgedrückt, wenn wir fortgesetzt 8 durch 2 teilen, brauchen wir 3 Schritte, bis wir 1 ankommen.

O(log N) heißt, dass der Algorithmus soviele Schritte braucht, wie es kostet,
die Datenelemente solange immer wieder zu halbieren, bis man bei der unteilbaren 1 ankommt.

O ($\log_2$ N) erklärt…

N Elements	O(N)	O(log N)
8	8	3
16	16	4
32	32	5
64	64	6
128	128	7
256	256	8
512	512	9
1024	1024	10

Praxisbeispiele

```
char *fruits [] = {"oranges", "lemons", "pineapples", "grapes"};
for (int i=0; i<4; i++)    printf ("Here's a fruit: %s\n", fruits[i]);
```

Effizienz in Big O-Notation für die sequentielle Ausgabe? 4 Früchte, 4 Zyklen: O (N)

Primzahlbestimmung:

```
bool IsPrime (int number)
{
    bool result =true;
    for (int i=2; i<number; i++)
    {
        if (0 == number % i)
        {
            result=false;
            break;
        }
    }
        return result;
}
```

Effizienz in Big O-Notation für das Prüfen einer Zahl N? Worst case der Anzahl von Tests: O (N)

DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim

Übung 1.1. – Frage 3

1. Normalerweise sucht die Suchoperation in einem Array nach dem ersten Vorkommen eines gegebenen Wertes. Manchmal möchte man aber alle Vorkommen dieses Wertes finden. Sagen, wir z.B., wie oft der Wert “Apfel” in einem Array gefunden wird. Wieviele Schritte werden benötigt, alle “Äpfel” zu finden? Geben Sie Ihre Antwort unter Nutzung von N.

A: O (1)
B: O (logN)
C: O (N)
D: O (N^2)

Übung 1.2. - Frage 1

1. Wieviele Schritte würde eine lineare Suche nach der Nummer 8 in folgendem sortierten Array erfordern?

   [2, 4, 6, 8, 10, 12, 13]?

A: 1
B: 2
C: 3
D: 4
E: 7

Übung 1.2. - Frage 2

2. Wieviele Schritte würde eine binäre Suche nach der Zahl 8 für dieses Beispiel erfordern? [2, 4, 6, 8, 10, 12, 13]?

A: 1
B: 2
C: 3
D: 4
E: 7

Übung 1.2. - Frage 3

3. Welche maximale Anzahl an Schritten würde es erfordern, eine binäre Suche über ein Array von 100.000 Elementen durchzuführen?

A: 16
B: 64
C: 1562
D: 50.000
E: 100.000