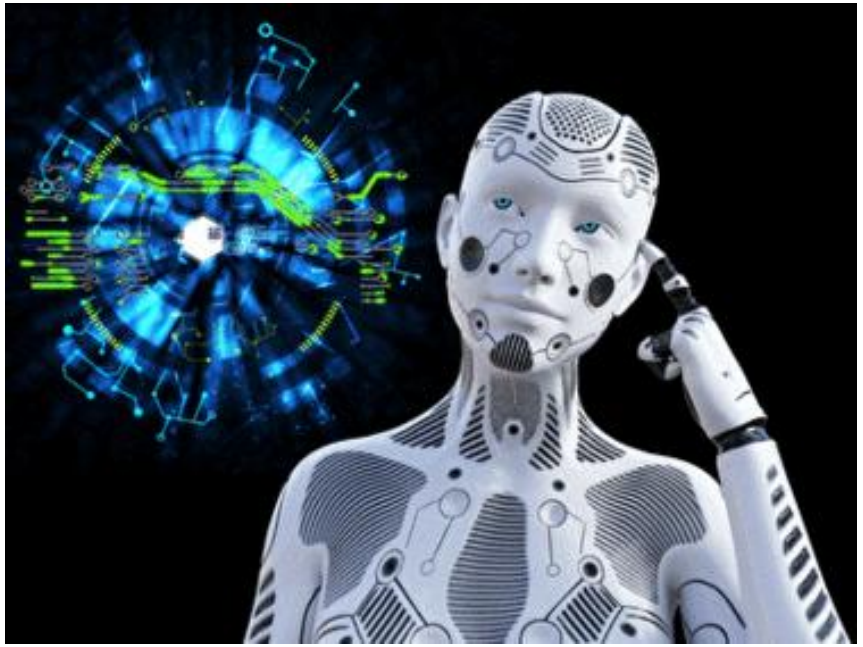
Algorithmen I

Dr. Philipp Hurni, Kantonsschule Sursee

Algorithmen

- Was ist ein Algorithmus?
  - Eigenschaften eines Algorithmus
- Algorithmen und ihre Komplexität
  - Lineare Suche
  - "Dilettantische" Lineare Suche
  - Komplexitätsklassen / O-Notation
  - Binäre Suche
- Beispiele und deren Komplexität
  - Sortieren
  - Foto-Probleme
  - String Suche



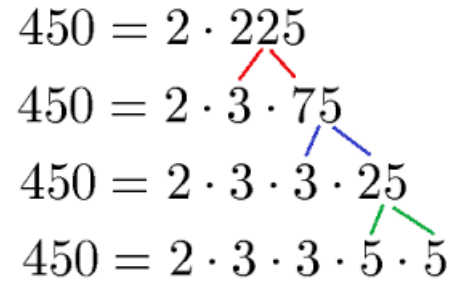
Was ist ein Algorithmus

- Ein Algorithmus ist eine **eindeutige Handlungsvorschrift** zur **Lösung eines Problems** oder einer Klasse von Problemen. Algorithmen bestehen aus endlich vielen, wohldefinierten **Einzelschritten**. [Wikipedia]

- Ein Algorithmus ist ein Ablauf bzw. eine **Schrittfolge**, mit der ein Problem **eindeutig**, in endlich vielen Schritten **gelöst** wird. Der Lösungsprozess ist nachvollziehbar und funktioniert immer gleich. [informatik-verstehen.de]

Eigenschaften eines Algorithmus

Eigenschaft	Beschreibung
Endlichkeit	Ein Algorithmus besteht aus einer endlichen Anzahl von Verarbeitungsschritten. Er kommt für jede Eingabe in endlichen vielen Verarbeitungsschritten zu einer Ausgabe.
Ausführbarkeit	Jede einzelne Anweisung eines Algorithmus muss so klar formuliert sein, dass sie (z.B. von einem Computer) eindeutig ausgeführt werden kann.
Eindeutigkeit	Die Abfolge der einzelnen Verarbeitungsschritte ist exakt festgelegt, das Resultat eindeutig
Allgemeinheit	Der Algorithmus kann eine ganze Klasse von Problemen lösen – nicht nur ein isoliertes Problem z.B. mit bestimmten Zahlen. Beispiel: Multiplikation von beliebigen Zahlen a, b.
Korrektheit	Der Algorithmus führt tatsächlich zur richtigen Lösung des Problems.



Algorithmen und ihre Komplexität

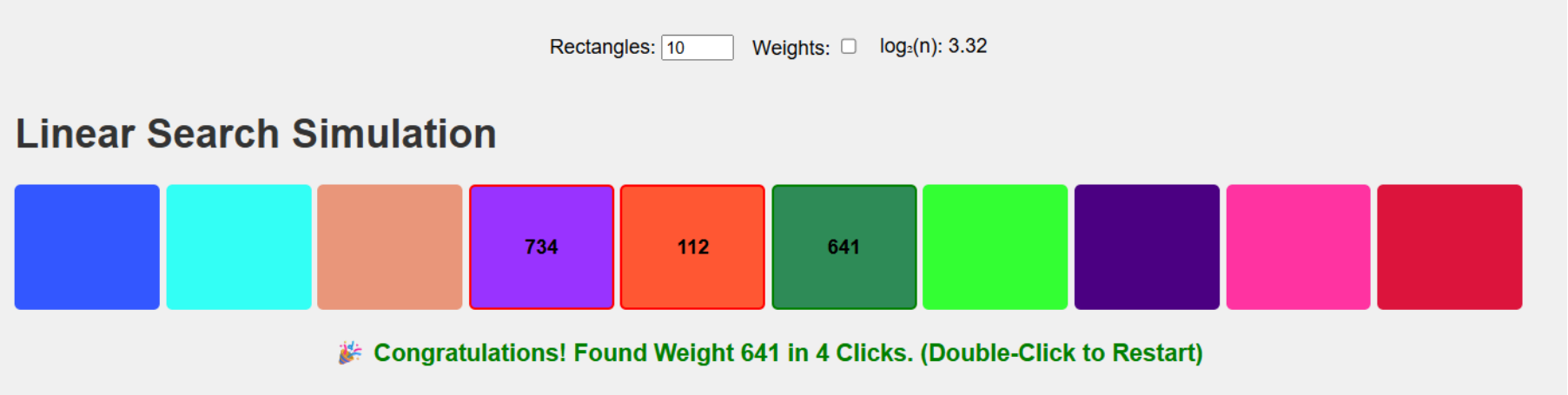
- Algorithmen kann man charakterisieren durch ihre **Komplexität**
- Man spricht von **Laufzeitkomplexität** und von **Speicherkomplexität**. Wir wollen uns hier auf die **Laufzeitkomplexität** fokussieren.
- Man kann ganze "Probleme" charakterisieren durch ihre Komplexität. Die Komplexität eines Algorithmus ist immer höher oder gleich der Komplexität des Problems selbst.
- Problemkomplexität beweisen ist teilweise sehr schwierig. Für viele Probleme ist die **Komplexität nicht bewiesen** (Paradebeispiel: Primfaktorzerlegung).

Suchen eines Objekts in einer Liste

https://hurni.4lima.de/search/

Suchen in Liste – Lineare Suche:

- Wie viele Suchschritte (Doppelklicks) hast du im Minimum?
- Wie viele Suchschritte (Doppelklicks) hast du im Maximum?
- Wie viele im Durschnitt?



Laufzeitkomplexität der Linearen Suche

- Wie viele Suchschritte (Doppelklicks) hast du im Minimum? 1
- Wie viele Suchschritte (Doppelklicks) hast du im Maximum? n
- Wie viele im Durchschnitt? (n+1)/2 = n/2 + ½

⇨ Klassifizierung von Algorithmen in Ordnungen

⇨ "O-Notation": Maximum Schritte (Worst Case) reduziert auf die "Stammfunktion"

Lineare Suche: O(n)

Paul Heinrich Bachmann
1837-1920

"Dilettantische" Lineare Suche

Angenommen, jedes Element wird drei Mal angeklickt, ehe man weitergeht. Und am Anfang macht man noch 100 Doppelklicks ins Leere.

- Wie viele Suchschritte (Doppelklicks) hast du im Minimum? 101
- Wie viele Suchschritte (Doppelklicks) hast du im Maximum? 3*n + 101

Worst-Case: 3*n + 101 ⇨ das führt zu O(n)

- Die Multiplikation mit 3 wird ignoriert – sie spielt bei grossen Zahlen keine wesentliche Rolle
- Der Konstante Term (101) ebenfalls

Die O-Notation

- Die O-Notation beschreibt die Laufzeit eines Algorithmus als Funktion der Eingabegrösse (meistens n). Sie gibt die obere Grenze der Wachstumsrate an – es ist also immer eine Worst-Case-Betrachtung!

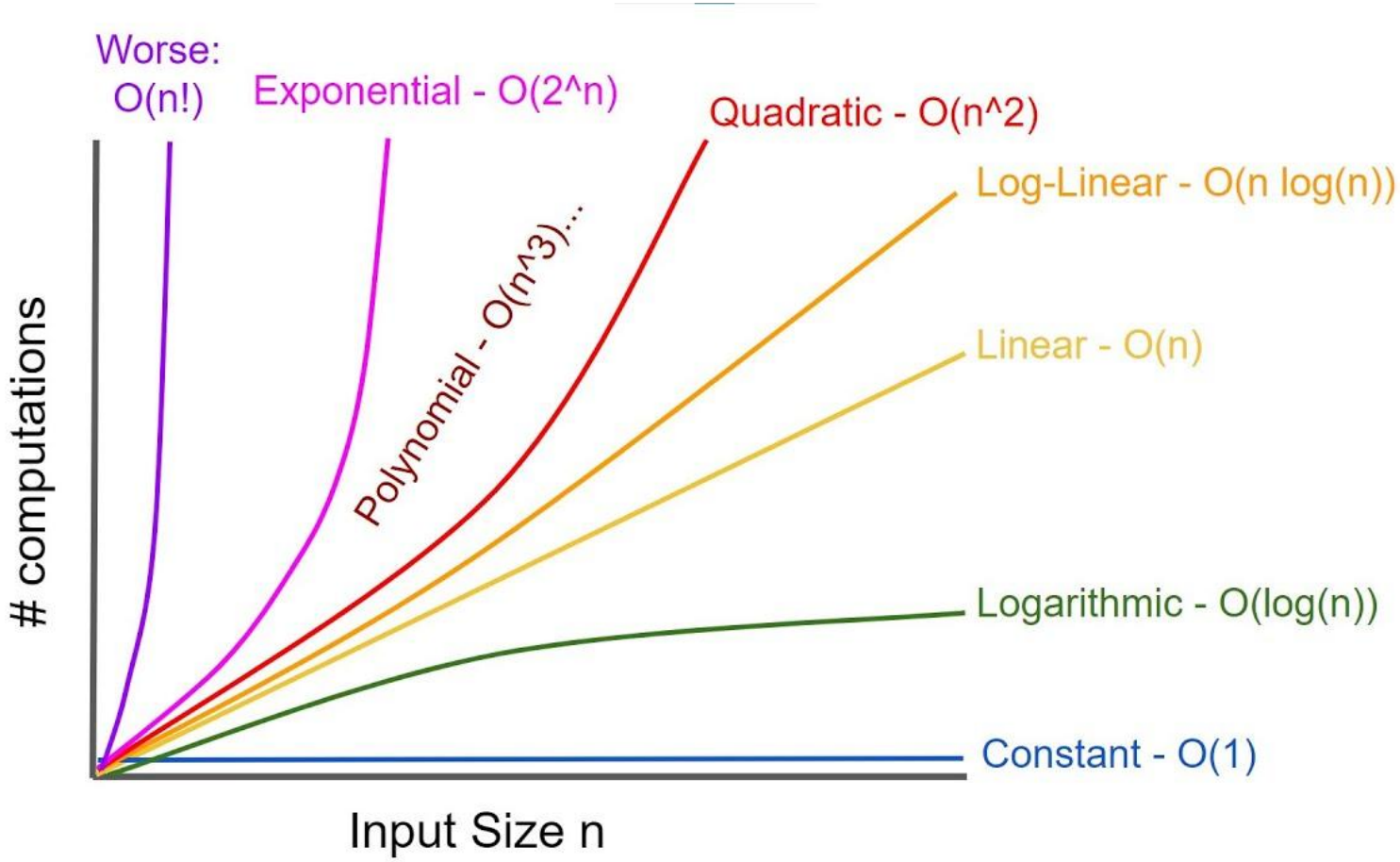
- Die O-Notation fokussiert auf den **dominanten Term**: Wenn wir die Laufzeit $T(n)=5n^4+20n^3+100$ betrachten, dann ist für grosse n der Term $5n^4$ viel grösser als die anderen.

  ⇨Die O-Notation ignoriert die niedrigeren Potenzen und Konstanten, da sie die Wachstumsrate des Algorithmus kaum beeinflussen.

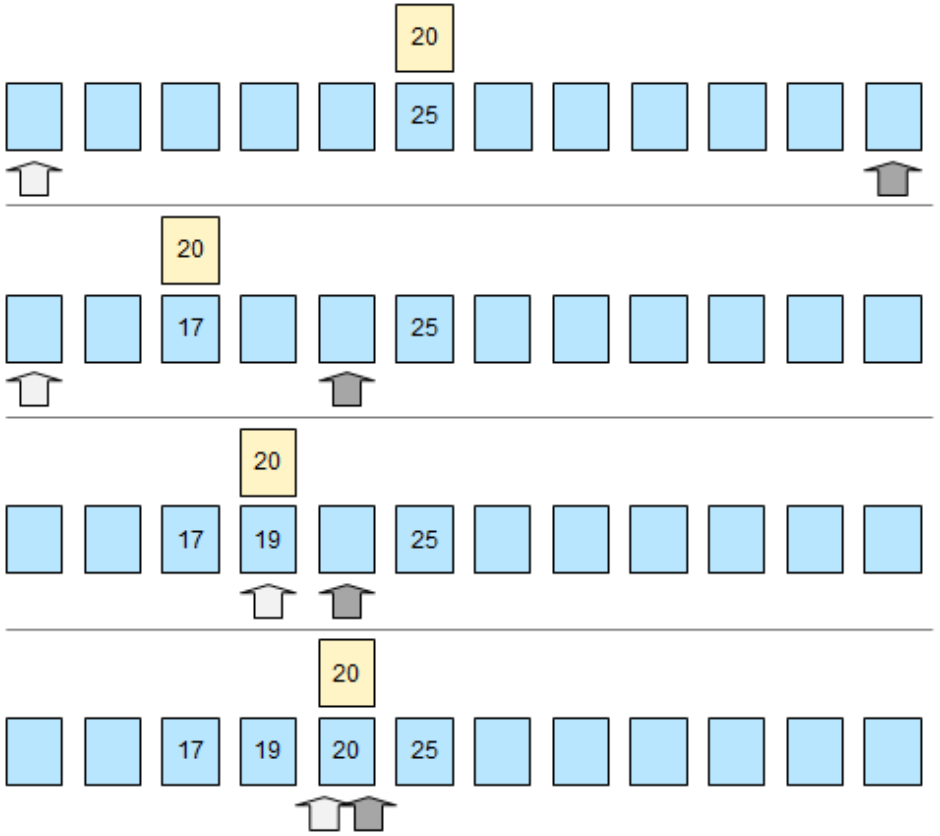
  ⇨Aus diesem Grund ist die Komplexität O(n4), weil n^4 das dominante Wachstum bestimmt.

- Die O-Notation ist ein Werkzeug, um zu verstehen, wie skalierbar ein Algorithmus ist.

Komplexitätsklassen



Binäre Suche



Binäre Suche

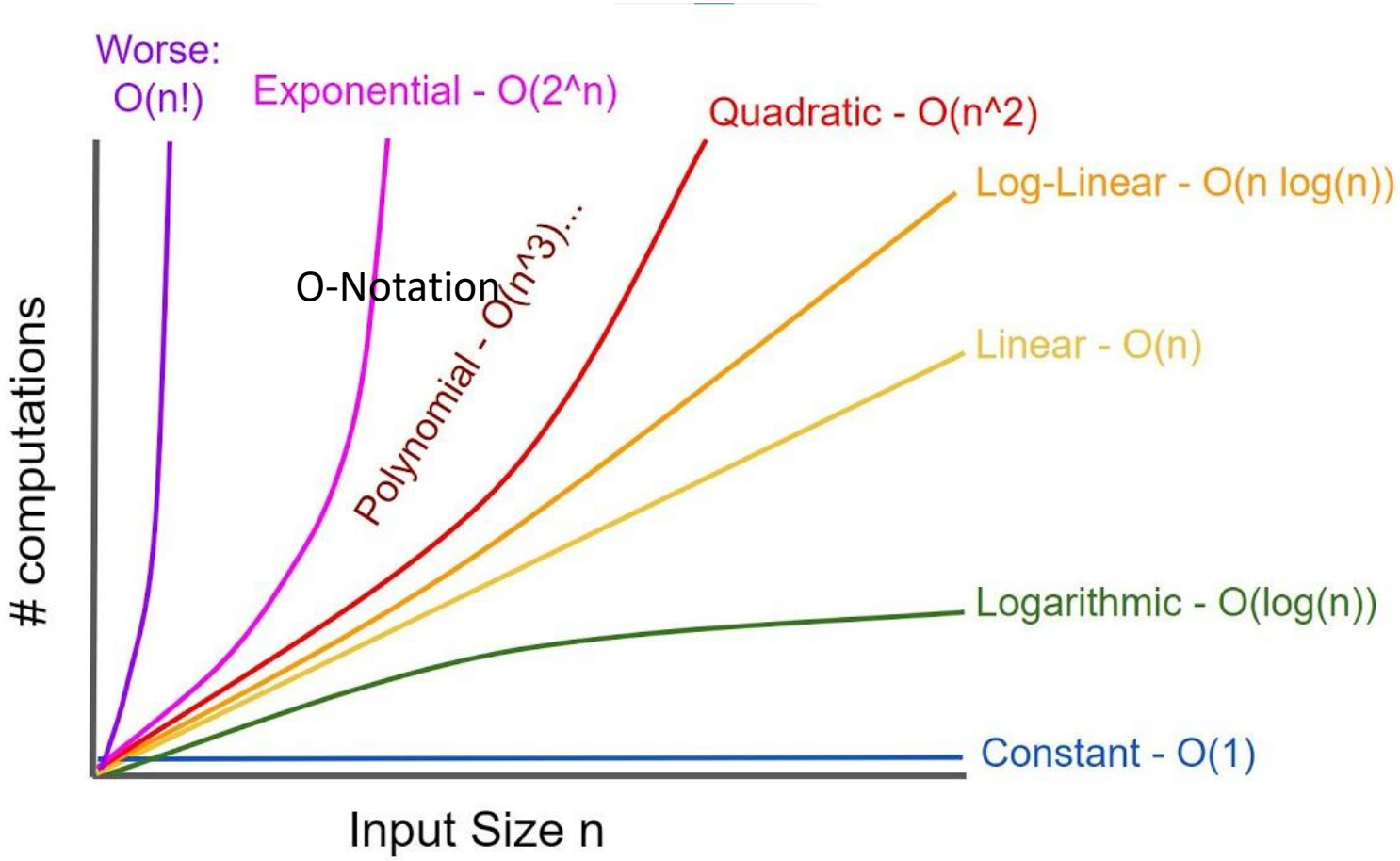
Suchen in Liste – Binäre Suche:

- Wie viele Suchschritte (Doppelklicks) hast du im Minimum?
- Wie viele Suchschritte (Doppelklicks) hast du im Maximum?

Suchen in Liste – Binäre Suche:

- Wie viele Suchschritte (Doppelklicks) hast du im Minimum? 1
- Wie viele Suchschritte (Doppelklicks) hast du im Maximum? $\log_2(n)$

Wo wären wir hier?



Beispiele und deren Komplexität

Algorithmen Beispiele: Der Bauer auf seinem Feld

- Der Bauer hat ein Feld von 10'000m^2
- Für 100m^2 braucht er 6 Minuten.

  linear
- Wie lange braucht er für das ganze Feld?

Algorithmen Beispiele: Der Bauer mit seinen Kürbissen

- Der Bauer hat 100 Kürbisse geerntet
- Alle sehen gleich gross aus, unterscheiden sich aber im Gewicht
- Er muss sie für einen Kunden gemäss Gewicht sortieren. Er hat eine Waage zur Verfügung
- 10 Kürbisse hat er in 10 Minuten sortiert.
- Wie lange hat er für 100 Kürbisse? Länger als 10*10 Minuten? Weniger als 10*10 Minuten? Oder gerade 10*10min? (keine genaue Berechnung nötig!)

weniger

Bauern-Aufgaben: Lösungen

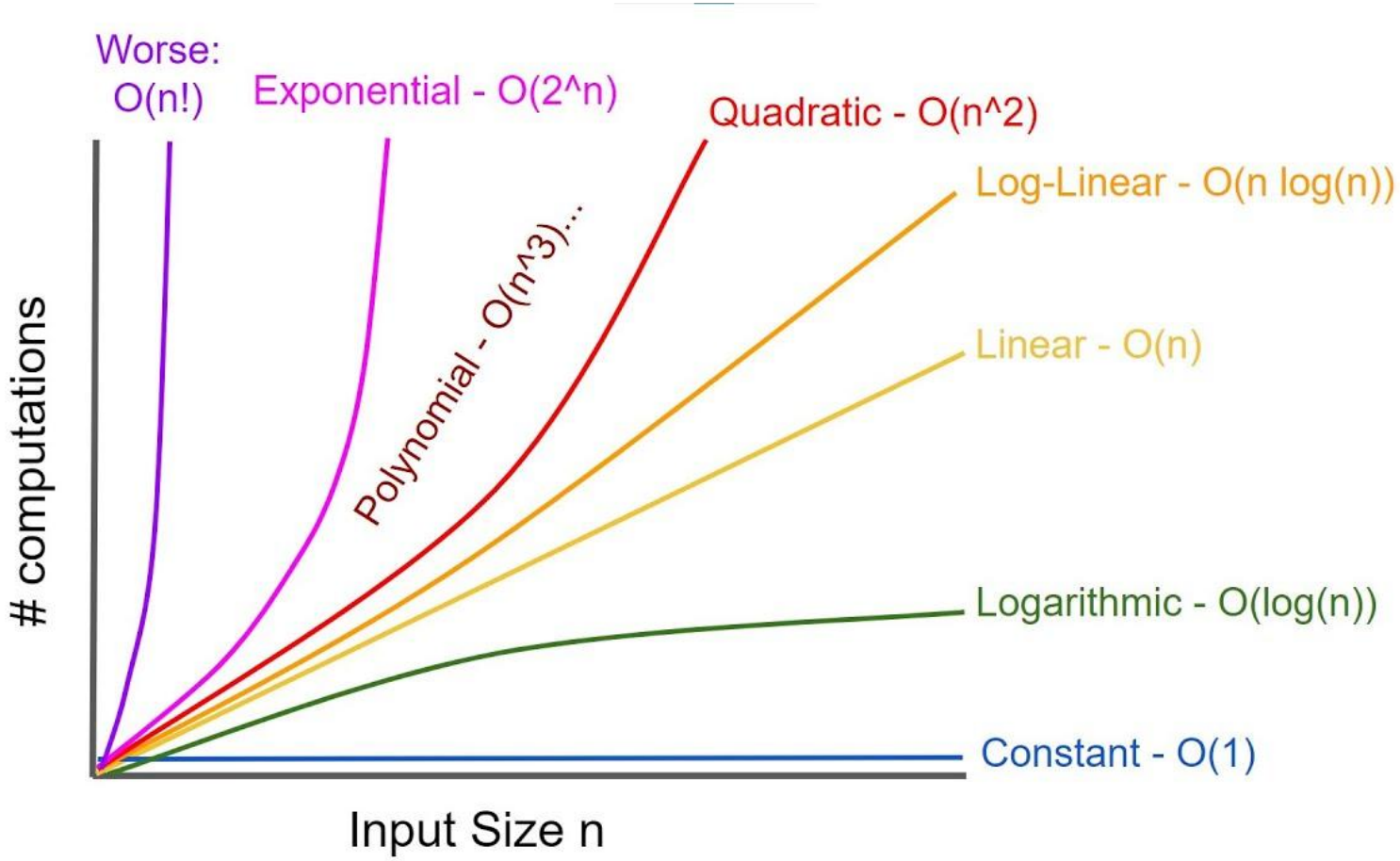
- Der Bauer hat ein Feld von 10'000m^2. Für 100m^2 braucht er 6 Minuten.
- Wie lange braucht er für das ganze Feld?

⇨Faktor ist 10'000/100 = 100
⇨6min * 100 = 600min = 10h

Sortierproblem

- Sortieren ist bestenfalls O(n*log2(n))
- Der Bauer braucht daher länger als 10*10min

Wo wären wir hier?



Beispiele und deren Komplexität

Foto-Probleme

Aufgaben zu Foto-Problemen

- Es gibt ein Fest mit Herrn Bundesrat Rösti
- Du bist Fotograf und musst eine Offerte machen
- Es sind zusammen mit Bundesrat Rösti 10 Leute eingeladen
- Für ein Foto brauchst du 10 Minuten.
- Berechne, wie viele Fotos du machen musst, wenn
  - a) Es ein Gruppenfoto geben soll mit allen Teilnehmenden 10 : O(1)
  - b) Es jeweils ein Bild geben soll mit Bundesrat Rösti und einer anderen Person n*10 = 9 *10 = 90 : O(
  - c) Es jeweils ein Bild geben soll von allen möglichen 2 Personen 2^n* 10 = 2^9 * 10 = 5'120: O(2

  Berechne zudem, wie lange die Aufgaben a-c jeweils Zeit erfordern
- Berechne nun dasselbe unter der Annahme, dass gesamthaft 100 Leute eingeladen sind.
- Bestimme die O-Notation der Probleme von a) b) und c)

Foto Probleme - Lösungen

a) Es ein Gruppenfoto geben soll mit allen Teilnehmenden
Konstante Laufzeitkomplexität: 10 Minuten
O-Notation: O(1)

b) Es jeweils ein Bild geben soll mit Bundesrat Rösti und einer anderen Person
2er Gruppen: (n-1)*10min = 99*10min = 990min = 16.5h
O-Notation: O(n)

c) Es jeweils ein Bild geben soll von allen möglichen 2 Personen (beliebige Paarungen)
n(n-1)/2 = 100 * 99 / 2 * 10min = 49’500min = 825h
O-Notation: $O(n^2)$

Übungsbeispiele

Zu welchen Komplexitätsklassen gehören die folgenden Beispiele?

- Algorithmus X hat eine Laufzeitkomplexitäts-Funktion von $l_x = 4n + 34$

  O(n)

- Algorithmus Y hat eine Laufzeitkomplexitäts-Funktion von $l_y = 3n^4+4^n$

  O(2^n)

- Algorithmus Z hat eine Laufzeitkomplexitäts-Funktion von $l_z = 3n! + n^4$

  O(n!)

Substring Suche

String Suche (n = Länge Inputstring text, m = Länge Suchstring searchterm)

text = "home is where your cat is"
searchterm = "cat"

(n-m)*m O(n*m)

Aufgaben

- Beschreibe einen Algorithmus in Prosa
- schreibe den Algorithmus in Python, welcher dieses Problem löst
- Beschreibe zuerst die Laufzeitkomplexitätsfunktion – den Worst-Case Fall
- Leite dann die O-Notation ab

String Suche: Lösung

text = "home is where your cat is" (n = Länge Inputstring)
searchterm = "cat" (m = Länge Suchstring)

1. **Beginn der Suche:** Wir starten am allerersten Buchstaben des langen Zugs (str) – hier "h"
2. **Vergleich mit kurzem String:** Von dieser Startposition aus vergleichen wir jeden den Buchstaben des langen Strings mit den Buchstaben des kurzen Strings (searchterm).
   - Falls Übereinstimmung gefunden: Wenn alle Buchstaben des kurzen Strings exakt mit den Buchstaben des langen Strings an dieser Position übereinstimmen, haben wir eine Übereinstimmung gefunden! Wir geben die Startposition (den Index) dieser Übereinstimmung zurück
   - Falls Keine Übereinstimmung: Wenn wir auch nur einen Buchstaben finden, der nicht übereinstimmt, wissen wir, dass der kurze String an dieser Position nicht passt.
3. **Verschieben und Wiederholen:** In diesem Fall verschieben wir den Startpunkt unserer Suche um einen Buchstaben nach rechts im langen String und wiederholen den Vergleich von Schritt 2.
4. **Ende der Suche:** Dieser Prozess wird fortgesetzt, bis wir entweder eine Übereinstimmung finden oder das Ende des langen Strings erreichen. Wenn wir das Ende erreichen, ohne eine Übereinstimmung gefunden zu haben, wissen wir, dass der Suchstring nicht im Input-String enthalten ist.

String Suche: Lösung

```
def substringsearch(text, searchterm):
    n = len(text)
    m = len(searchterm)

    # Äussere Schleife für die Suche über den text
    for i in range(n - m + 1):
        # innere Schleife für den Vergleich der Zeichen des Suchterms
        match = True
        for j in range(m):
            if text[i + j] != searchterm[j]:
                match = False
                break

        if match:
            return i

    return -1

text = "home is where your cat is"
searchterm = "cat"

result = substringsearch(text, searchterm)
print(result)
```

String Suche: Lösung

- Im schlechtesten Fall wissen wir erst im letzten Schritt, ob der searchterm im text vorkommt
- Beispiel: Wir suchen nach "see" in "sesesee" -> n=6, m=3
- Wir würden also zuerst n-3+1 mal m Schritte suchen
  = (n-m+1)*m
  = nm-m^2+m
- O-Notation davon wäre: O(n*m) – weil n >> m ist

Komplexitätsklassen