

Bubblesort

1 Grundidee

Bubblesort vergleicht **benachbarte Elemente** und vertauscht sie, wenn sie in der falschen Reihenfolge stehen. In jedem Durchlauf „wandert“ das **größte Element nach rechts** – ähnlich wie eine Blase nach oben.

- Mehrere Durchläufe über die Liste
 - Nach jedem Durchlauf ist das letzte Element korrekt positioniert
 - Optimierte Variante: Abbruch, wenn kein Tausch mehr erfolgt
-

2 Voraussetzungen

- **X** keine
 - funktioniert auf **beliebigen vergleichbaren Elementen**
-

3 Laufzeiten & Eigenschaften

Eigenschaft	Wert
Best Case	$O(n)$ (optimiert)
Average Case	$O(n^2)$
Worst Case	$O(n^2)$
Speicherbedarf	$O(1)$
In-place	ja
Stabil	ja

Hinweis: Ohne Optimierung ist der Best Case ebenfalls $O(n^2)$.

4 Schritt-für-Schritt-Beispiel

Ausgangsliste:

[5, 3, 4, 1]

Durchlauf 1

- (5,3) tauschen → [3, 5, 4, 1]
- (5,4) tauschen → [3, 4, 5, 1]
- (5,1) tauschen → [3, 4, 1, 5]

Durchlauf 2

- (3,4) ok
- (4,1) tauschen → [3, 1, 4, 5]

Durchlauf 3

- (3,1) tauschen → [1, 3, 4, 5]

Ergebnis:

[1, 3, 4, 5]

Bubble Sort – Algorithmus in Worten

Der Algorithmus betrachtet wiederholt benachbarte Elemente der Liste.

Er beginnt beim ersten Element und vergleicht es mit dem nächsten.

Sind die beiden Elemente in der falschen Reihenfolge, werden sie vertauscht.

Anschließend wird ein Element weiter nach rechts gegangen und erneut verglichen.

Nach einem vollständigen Durchlauf befindet sich das größte Element am Ende der Liste.

Dieser Vorgang wird für den verbleibenden, noch unsortierten Teil der Liste wiederholt, bis in einem Durchlauf kein Tausch mehr notwendig ist oder die Liste vollständig sortiert ist.\

5 Besonderheiten / Prüfungsrelevante Hinweise

- Sehr **einfach**, aber **ineffizient**
 - Gut geeignet zur Erklärung von:
 - Stabilität
 - Best-Case-Optimierung
 - Kaum praxisrelevant bei großen Datenmengen
-

6 Vor- und Nachteile

Vorteile

- stabil
- leicht zu verstehen
- in-place

Nachteile

- sehr langsam bei großen n
 - nur für Lernzwecke sinnvoll
-



Merksatz für die Prüfung

Bubblesort vertauscht benachbarte Elemente und ist stabil, aber ineffizient mit $O(n^2)$.



Python-Implementierung

```
In [1]: def bubble_sort(arr):
    n = len(arr)

    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
            if not swapped:
                break

    return arr
```