🟦 Bubblesort

1️⃣ 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

---

2️⃣ Voraussetzungen

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

3️⃣ 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 $O(n^2)$.

4️⃣ 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]
```

5️⃣ Besonderheiten / Prüfungsrelevante Hinweise

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

6️⃣ 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 $O(n^2)$.

7️⃣ Python-Implementierung

In [1]:
```python
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
```