

Selection-Sort

1 Grundidee

Selection-Sort sucht in jedem Durchlauf das **kleinste Element im unsortierten Teil** der Liste und **tauscht es an die richtige Position** am Anfang.

- Die Liste wird in zwei Teile gedacht:
 - **links:** bereits sortiert
 - **rechts:** noch unsortiert
 - In jedem Schritt wächst der sortierte Teil um **ein Element**
-

2 Voraussetzungen

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

3 Laufzeiten & Eigenschaften

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

Wichtig: Auch wenn die Liste bereits sortiert ist, führt Selection-Sort immer alle Vergleiche aus → Laufzeit bleibt **$O(n^2)$** .

4 Schritt-für-Schritt-Beispiel

Ausgangsliste:

[5, 3, 4, 1]

Durchlauf 1 (i = 0)

- Minimum in [5, 3, 4, 1] → 1
- Tausch mit Position 0

[1, 3, 4, 5]

Durchlauf 2 (i = 1)

- Minimum in [3, 4, 5] → 3
- kein Tausch nötig

Durchlauf 3 (i = 2)

- Minimum in [4, 5] → 4
- kein Tausch nötig

Ergebnis:

[1, 3, 4, 5]

Selection Sort – Algorithmus in Worten

Der Algorithmus beginnt beim ersten Element der Liste.

Er sucht im gesamten noch unsortierten Teil der Liste das kleinste Element.

Dieses kleinste Element wird mit dem aktuellen Element am Anfang des unsortierten Bereichs vertauscht.

Dadurch befindet sich das kleinste Element an seiner endgültigen Position.

Der unsortierte Bereich der Liste beginnt nun eine Position weiter rechts.

Dieser Vorgang wird für jedes weitere Element wiederholt,
bis die gesamte Liste sortiert ist.

5 Warum Selection-Sort nicht stabil ist

Beispiel:

[(2, "A"), (1, "X"), (2, "B")]

- erstes Minimum: (1, "X")
- wird mit (2, "A") getauscht
- die relative Reihenfolge von (2, "A") und (2, "B") kann sich ändern

→ **Selection-Sort ist nicht stabil**

6 Vor- und Nachteile

Vorteile

- sehr **einfach** zu verstehen
- **wenige Tauschoperationen**
- konstanter Speicherbedarf

Nachteile

- **immer $O(n^2)$**
- ungeeignet für grosse Datenmengen

Merksatz für die Prüfung

Selection-Sort ist ein einfacher, in-place Sortieralgorithmus mit $O(n^2)$ -Laufzeit, der in jedem Schritt das Minimum auswählt, aber nicht stabil ist.

7 Python-Implementierung

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

        for i in range(n - 1):
            min_index = i

            # Minimum im unsortierten Teil suchen
            for j in range(i + 1, n):
                if arr[j] < arr[min_index]:
                    min_index = j

            # Tauschen, falls nötig
            if min_index != i:
                arr[i], arr[min_index] = arr[min_index], arr[i]

        return arr
```