

Mergesort

1 Grundidee

Mergesort ist ein **Divide-and-Conquer-Algorithmus**. Die Liste wird rekursiv in **immer kleinere Teillisten** zerlegt, bis diese trivial sortiert sind, und anschliessend **geordnet wieder zusammengeführt (merge)**.

- Aufteilen → Sortieren → Zusammenführen
 - Rekursive Struktur
 - Sehr gut für grosse Datenmengen
-

2 Voraussetzungen

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

3 Laufzeiten & Eigenschaften

Eigenschaft	Wert
Best Case	$O(n \log n)$
Average Case	$O(n \log n)$
Worst Case	$O(n \log n)$
Speicherbedarf	$O(n)$
In-place	nein
Stabil	ja

Hinweis: Die Laufzeit ist **immer $O(n \log n)$** , unabhängig von der Eingabereihenfolge.

4 Schritt-für-Schritt-Beispiel

Ausgangsliste:

[5, 3, 4, 1]

Aufteilen

[5, 3] [4, 1]
[5] [3] [4] [1]

Zusammenführen

[3, 5] [1, 4]
[1, 3, 4, 5]

Ergebnis:

[1, 3, 4, 5]

Merge Sort – Algorithmus in Worten

Der Algorithmus teilt die Liste wiederholt in zwei möglichst gleich große Hälften.

Dieser Vorgang wird so lange fortgesetzt, bis jede Teilliste nur noch ein einzelnes Element enthält.

Ein einzelnes Element gilt dabei als bereits sortiert.

Anschließend werden jeweils zwei benachbarte Teillisten zusammengeführt.

Beim Zusammenführen werden die jeweils kleinsten Elemente der beiden Teillisten verglichen.

Das kleinere Element wird in die neue Liste übernommen.

Dieser Vorgang wird fortgesetzt, bis alle Elemente beider Teillisten übernommen wurden.

Durch das wiederholte Zusammenführen entsteht am Ende eine vollständig sortierte Liste.\

5 Besonderheiten / Prüfungsrelevante Hinweise

- Garantierte Laufzeit $O(n \log n)$
 - Sehr gut für **grosse Datenmengen**
 - Wird oft als **Referenzalgorithmus** verwendet
 - Nachteil: zusätzlicher Speicherbedarf
-

6 Vor- und Nachteile

Vorteile

- stabil
- vorhersehbare Laufzeit
- gut parallelisierbar

Nachteile

- nicht in-place
 - zusätzlicher Speicherbedarf
-

Merksatz für die Prüfung

Mergesort teilt die Liste rekursiv und fügt sie sortiert zusammen – stabil mit $O(n \log n)$.

7 Python-Implementierung

```
In [1]: def merge_sort(arr):
        if len(arr) <= 1:
            return arr

        mid = len(arr) // 2
        left = merge_sort(arr[:mid])
        right = merge_sort(arr[mid:])

        return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])
    return result
```