

# Algorithmen und Datenstrukturen

## A12. Sortieren: Quicksort (& Heapsort)

Marcel Lüthi and Gabriele Röger

Universität Basel

16. März 2023

# Algorithmen und Datenstrukturen

16. März 2023 — A12. Sortieren: Quicksort (& Heapsort)

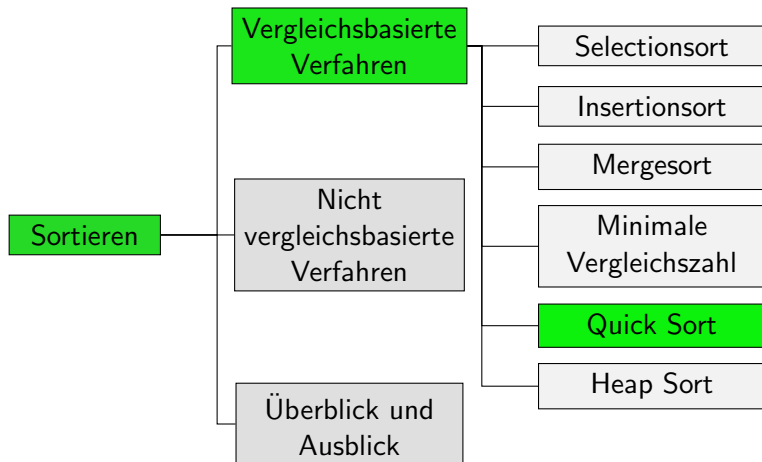
A12.1 Quicksort

A12.2 Heapsort

A12.3 Zusammenfassung

# A12.1 Quicksort

# Sortierverfahren



## Quicksort: Idee

- ▶ Wie Merge-Sort ein **Divide-and-Conquer-Verfahren**
- ▶ Die Sequenz wird nicht wie bei Mergesort nach Positionen aufgeteilt, sondern nach Werten.
- ▶ Hierfür wird ein Element  $P$  gewählt (das sogenannte **Pivotelement**).
- ▶ Dann wird so umsortiert, dass  $P$  an die endgültige Position kommt, vor  $P$  nur Elemente  $\leq P$  stehen, und hinten nur Elemente  $\geq P$ .



- ▶ Macht man das rekursiv für den vorderen und den hinteren Teil, ist die Sequenz am Ende sortiert.

# Quicksort: Algorithmus

---

```
1 def sort(array):
2     sort_aux(array, 0, len(array)-1)
3
4 def sort_aux(array, lo, hi):
5     if hi <= lo:
6         return
7     choose_pivot_and_swap_it_to_lo(array, lo, hi)
8     pivot_pos = partition(array, lo, hi)
9     sort_aux(array, lo, pivot_pos - 1)
10    sort_aux(array, pivot_pos + 1, hi)
```

---

# Wie wählt man das Pivot-Element?

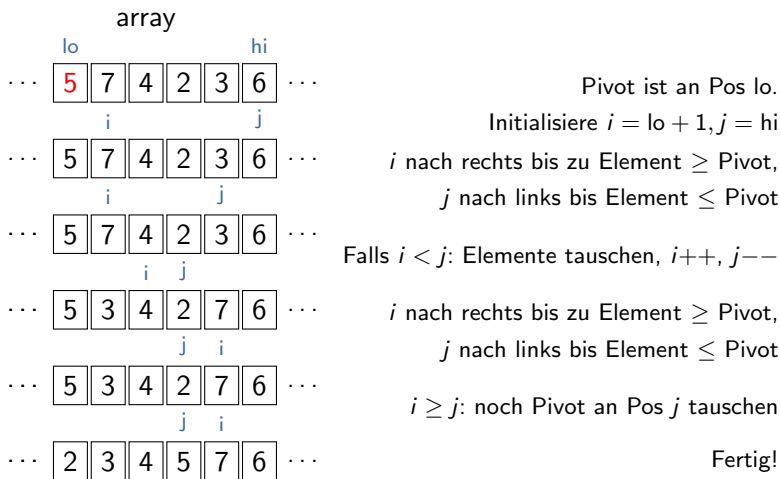
Für die Korrektheit des Verfahrens ist das egal. (Warum?)

Wir können zum Bsp. folgende Strategien wählen:

- ▶ **Naiv:** Nimm immer erstes Element
- ▶ **Median of Three:** Verwende Median aus erstem, mittlerem und letztem Element
- ▶ **Randomisiert:** Wähle zufällig ein Element aus

Gute Pivot-Elemente teilen Sequenz in etwa gleich grosse Bereiche.

# Wie macht man die Umsortierung?





# Quicksort: Partitionierung

---

```
1 def partition(array, lo, hi):
2     pivot = array[lo]
3     i = lo + 1
4     j = hi
5     while (True):
6         while i < hi and array[i] < pivot:
7             i += 1
8         while array[j] > pivot:
9             j -= 1
10        if i >= j:
11            break
12
13        array[i], array[j] = array[j], array[i]
14        i, j = i + 1, j - 1
15    array[lo], array[j] = array[j], array[lo]
16    return j
```

---

# Aufgabe

Wie sieht das Array  $[6, 5, 7, 8, 3]$  nach einem Aufruf von `partition` für den gesamten Bereich (von Position 0 bis 4) aus?



# Quicksort: Laufzeit I

**Best case:** Pivot-Element teilt in gleich grosse Bereiche

- ▶  $O(\log_2 n)$  rekursive Aufrufe
- ▶ jeweils hi-lo Schlüsselvergleiche in Partitionierung
- ▶ auf einer Rekursionsebene insgesamt  $O(n)$  Vergleiche in Partitionierung

→  $O(n \log n)$

**Worst case:** Pivot-Element immer kleinstes oder grösstes Element

- ▶ insgesamt  $n-1$  (nichttriviale) rekursive Aufrufe für Länge  $n, n-1, \dots, 2$ .
- ▶ jeweils hi-lo Schlüsselvergleiche in Partitionierung

→  $\Theta(n^2)$

## Quicksort: Laufzeit II

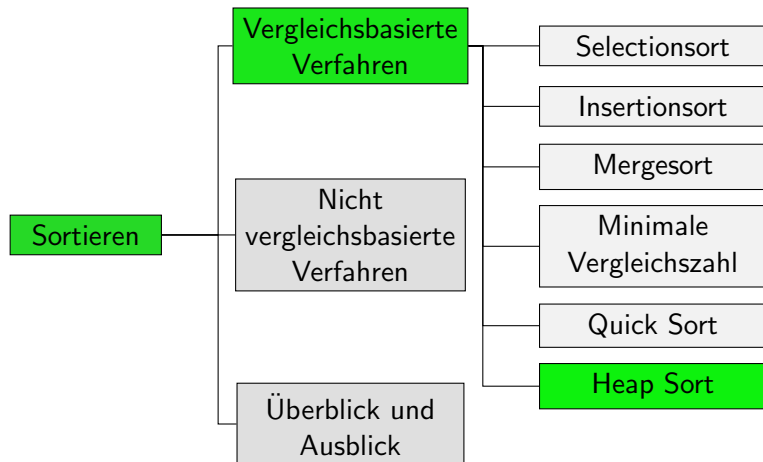
### Average case:

- ▶ Annahme:  $n$  verschiedene Elemente, jede der  $n!$  Permutationen gleich wahrscheinlich, Pivotelement zufällig gewählt
- ▶  $O(\log n)$  rekursive Aufrufe
- ▶ insgesamt  $O(n \log n)$
- ▶ etwa 39% langsamer als best case

Bei randomisierter Pivotwahl tritt worst-case quasi nicht auf. Quicksort wird daher oft als  $O(n \log n)$ -Verfahren betrachtet.

## A12.2 Heapsort

# Sortierverfahren



# Heapsort

- ▶ **Heap:** Datenstruktur, die das Finden und Entnehmen des **grössten** Elements besonders effizient unterstützt  
Finden:  $\Theta(1)$ , Entnehmen:  $\Theta(\log n)$
- ▶ **Grundidee analog zu Selectionsort:** Setze sukzessive das grösste Element an das Ende des unsortierten Bereichs.
- ▶ Kann den **Heap direkt in der Eingabesequenz repräsentieren**, so dass Heapsort nur konstanten zusätzlichen Speicherplatz benötigt.
- ▶ Die Laufzeit von Heapsort ist leicht überlinear.
- ▶ Wir besprechen die Details später, wenn wir Heaps genauer kennengelernt haben.

## A12.3 Zusammenfassung



# Zusammenfassung

- ▶ **Quicksort** ist ein **Divide-and-Conquer**-Verfahren, das die Elemente relativ zu einem **Pivotelement** aufteilt.
- ▶ Im **Worst-case** hat Quicksort ein **quadratisches Laufzeitverhalten**.
- ▶ Im **Average-case** ist die Laufzeit **leicht überlinear**.
- ▶ Bei randomisierter Pivotwahl tritt der Worst-case fast nie auf.