

# Algorithmen und Datenstrukturen

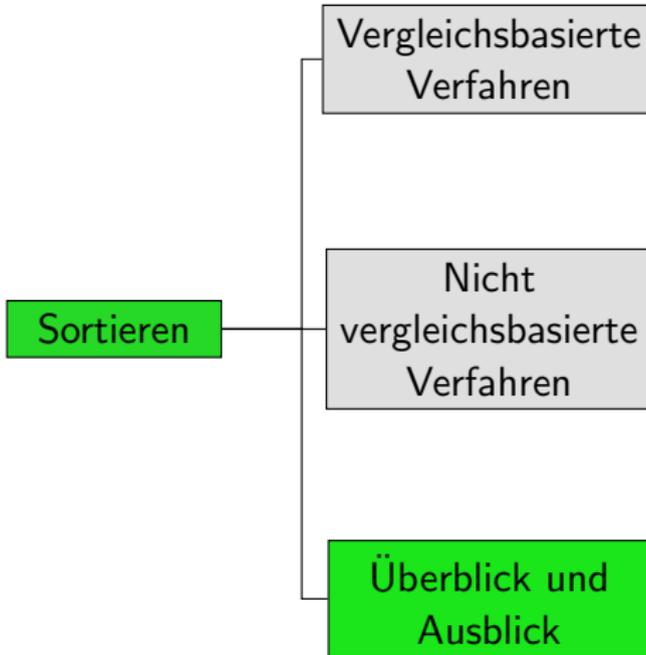
## A8. Sortieren IV

Marcel Lüthi and Gabriele Röger

Universität Basel

13. März 2020

# Sortierverfahren



# Überblick

# Vergleichsbasierte Verfahren: Übersicht

Algorithmus	Laufzeit $O(\cdot)$	Speicherbedarf $O(\cdot)$	stabil
Selectionsort	best/avg./worst $n^2$	best/avg./worst 1	nein
Insertionsort	$n/n^2/n^2$	1	ja
Mergesort	$n \log n$	$n$	ja
Quicksort	$n \log n/n \log n/n^2$	$\log n/\log n/n$	nein
Heapsort	$n \log n$	1	nein

# Vergleichsbasierte Verfahren: Übersicht

Algorithmus	Laufzeit $O(\cdot)$	Speicherbedarf $O(\cdot)$	stabil
	best/avg./worst	best/avg./worst	
Selectionsort	$n^2$	1	nein
Insertionsort	$n/n^2/n^2$	1	ja
Mergesort	$n \log n$	$n$	ja
Quicksort	$n \log n/n \log n/n^2$	$\log n/\log n/n$	nein
Heapsort	$n \log n$	1	nein

Sehr schöne [Visualisierung der Verfahren](https://www.toptal.com/developers/sorting-algorithms/) unter

<https://www.toptal.com/developers/sorting-algorithms/>

## Vergleichsbasierte Verfahren: Bemerkungen

- **Insertionsort** ist auf **kleinen Sequenzen** sehr **schnell** und wird daher zum Beispiel zur Verbesserung von Mergesort und Quicksort für kurze Aufrufe eingesetzt.

## Vergleichsbasierte Verfahren: Bemerkungen

- **Insertionsort** ist auf **kleinen Sequenzen** sehr **schnell** und wird daher zum Beispiel zur Verbesserung von Mergesort und Quicksort für kurze Aufrufe eingesetzt.
- **Quicksort** hat eine sehr kurze (= schnelle) innere Schleife. Mit Randomisierung tritt schlechtester Fall so gut wie nie auf.

## Vergleichsbasierte Verfahren: Bemerkungen

- **Insertionsort** ist auf **kleinen Sequenzen** sehr **schnell** und wird daher zum Beispiel zur Verbesserung von Mergesort und Quicksort für kurze Aufrufe eingesetzt.
- **Quicksort** hat eine sehr kurze (= schnelle) innere Schleife. Mit Randomisierung tritt schlechtester Fall so gut wie nie auf.
- **Mergesort** ist dafür **stabil**. Zudem ist der Mergeschritt auch für externes Sortieren relevant (**dazu gleich noch mehr**). Wird z.B. gerne bei Datenbankanwendungen eingesetzt.

## Vergleichsbasierte Verfahren: Bemerkungen

- **Insertionsort** ist auf **kleinen Sequenzen** sehr **schnell** und wird daher zum Beispiel zur Verbesserung von Mergesort und Quicksort für kurze Aufrufe eingesetzt.
- **Quicksort** hat eine sehr kurze (= schnelle) innere Schleife. Mit Randomisierung tritt schlechtester Fall so gut wie nie auf.
- **Mergesort** ist dafür **stabil**. Zudem ist der Mergeschritt auch für externes Sortieren relevant (**dazu gleich noch mehr**). Wird z.B. **gerne bei Datenbankanwendungen eingesetzt**.
- **Heapsort** ist in der Praxis etwas langsamer als Mergesort, als **in-place**-Verfahren aber dennoch interessant z.B. **für eingebettete Systeme**.

## Vergleichsbasierte Verfahren: Bemerkungen

- **Insertionsort** ist auf **kleinen Sequenzen** sehr **schnell** und wird daher zum Beispiel zur Verbesserung von Mergesort und Quicksort für kurze Aufrufe eingesetzt.
- **Quicksort** hat eine sehr kurze (= schnelle) innere Schleife. Mit Randomisierung tritt schlechtester Fall so gut wie nie auf.
- **Mergesort** ist dafür **stabil**. Zudem ist der Mergeschritt auch für externes Sortieren relevant (**dazu gleich noch mehr**). Wird z.B. **gerne bei Datenbankanwendungen eingesetzt**.
- **Heapsort** ist in der Praxis etwas langsamer als Mergesort, als **in-place**-Verfahren aber dennoch interessant z.B. **für eingebettete Systeme**.
- Gleiche asymptotische Laufzeit bedeutet nicht, dass Verfahren auch gleich lange brauchen (verschiedene Konstanten in  $O(\cdot)$ ). **Heapsort braucht doppelt so viele Vergleiche wie Mergesort**.

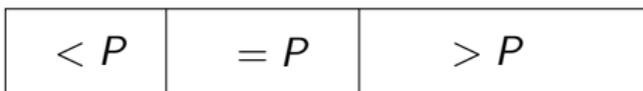
# Ausblick

# Vorsortierte Daten

- Oftmals sind **Teilsequenzen** der Eingabe bereits **vorsortiert**.
- Insertionsort profitiert davon direkt.
- Von manchen Verfahren gibt es Varianten, die Vorsortierung ausnutzen  
z.B. natürliches 2-Wege-Mergesort.

# Viele gleiche Schlüssel

- Tritt in praktischen Anwendungen häufig auf  
z.B. Sortieren von Studierendendaten nach Geschlecht
- Von manchen Algorithmen gibt es spezialisierte Varianten
- Zum Beispiel 3-Wege-Partitionierung in Quicksort



# Sortieren komplexer Objekte

- Meist will man nicht nur Zahlen, sondern **komplexe Objekte** sortieren.
- Hier wäre es sehr teuer, bei jeder Vertauschung die ganzen Objekte zu kopieren.
- **Stattdessen:** Sortiere Elemente, die nur aus Schlüssel und Zeiger/Referenz auf das tatsächliche Objekt bestehen.

# Externes Sortieren

- **Sehr grosse Datensätze** passen nicht in den internen Speicher.
- **Annahme: interner Speicher hat Grösse  $m$**
- Lese jeweils Bereich der Grösse  $m$  ein, sortiere ihn und schreibe ihn zurück auf den externen Speicher.
- Dann sukzessives Mergen ähnlich zu Mergesort. Konzeptuell:
  - lese jeweils Bereich der Grösse  $m/3$  der beiden vorsortierten Bereiche ein
  - es bleiben  $m/3$  für Ausgabe
  - falls ein Eingabebereich erschöpft ist → nachladen
  - falls Ausgabebereich voll → rausschreiben

# Weniger korrekte Verfahren

## INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):  
  IF LENGTH(LIST) < 2:  
    RETURN LIST  
  PIVOT = INT(LENGTH(LIST) / 2)  
  A = HALFHEARTEDMERGESORT(LIST[:PIVOT])  
  B = HALFHEARTEDMERGESORT(LIST[PIVOT:])  
  // UMMMMMM  
  RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):  
  // AN OPTIMIZED BOGOSORT  
  // RUNS IN  $O(N \log N)$   
  FOR N FROM 1 TO LOG(LENGTH(LIST)):  
    SHUFFLE(LIST):  
    IF ISSORTED(LIST):  
      RETURN LIST  
  RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBIINTERVIEWQUICKSORT(LIST):  
  OK SO YOU CHOOSE A PIVOT  
  ...
```

```
DEFINE PANICSORT(LIST):  
  IF ISSORTED(LIST):
```

vollständiger Comic unter <https://xkcd.com/1185/>  
(CC BY-NC 2.5)

# Andere Probleme durch Sortieren lösen

## $k$ -kleinstes Element

- zum Beispiel Finden des Medians ( $k = \lfloor n/2 \rfloor$ )
- Verwende Quicksort, aber mache rekursiven Aufruf nur für den relevanten Bereich.

# Andere Probleme durch Sortieren lösen

## $k$ -kleinstes Element

- zum Beispiel Finden des Medians ( $k = \lfloor n/2 \rfloor$ )
- Verwende Quicksort, aber mache rekursiven Aufruf nur für den relevanten Bereich.

## Duplikate

- Wie viele verschiedene Schlüssel gibt es? Welcher Wert ist am häufigsten? Gibt es doppelte Schlüssel?
- Kann man direkt mit quadratischen Algorithmen beantworten.
- Oder – schlauer – erst sortieren und dann mit einem Durchlauf lösen.