

# Algorithmen und Datenstrukturen

## A8. Laufzeitanalyse: Top-Down-Mergesort

Marcel Lüthi and Gabriele Röger

Universität Basel

9. März 2023

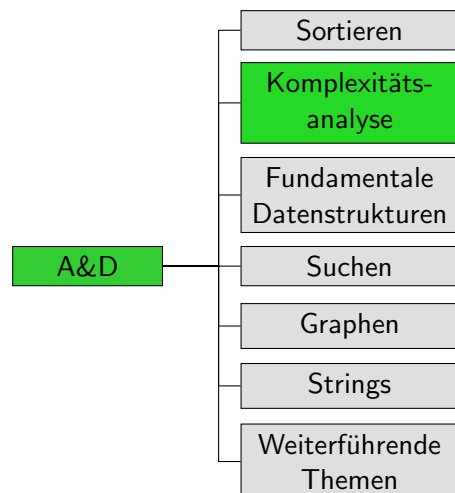
# Algorithmen und Datenstrukturen

9. März 2023 — A8. Laufzeitanalyse: Top-Down-Mergesort

## A8.1 Beispiel: Top-Down-Mergesort

## A8.2 Zusammenfassung

## Inhalt dieser Veranstaltung



## Was bisher geschah und wie es weiter geht

- ▶ **Zuletzt:** sehr detaillierte Laufzeitanalyse für Selectionsort und Bottom-Up-Mergesort
- ▶ jetzt noch analoge Analyse für Top-Down-Mergesort als Beispiel eines rekursiven Divide-and-Conquer-Verfahrens
- ▶ danach **Landau-Symbole** für asymptotisches Laufzeitverhalten
- ▶ und die „schnelle“ Laufzeitanalyse in der Praxis

## A8.1 Beispiel: Top-Down-Mergesort

## Merge-Schritt-Ergebnis vom letzten Mal

```

1 def merge(array, tmp, lo, mid, hi):
2     i = lo
3     j = mid + 1
4     for k in range(lo, hi + 1): # k = lo, ..., hi
5         if j > hi or (i <= mid and array[i] <= array[j]):
6             tmp[k] = array[i]
7             i += 1
8         else:
9             tmp[k] = array[j]
10            j += 1
11    for k in range(lo, hi + 1): # k = lo, ..., hi
12        array[k] = tmp[k]

```

### Theorem

Der Merge-Schritt hat *lineare Laufzeit*, d.h. es gibt Konstanten  $c, c', n_0 > 0$ , so dass für alle  $n \geq n_0$ :  $cn \leq T(n) \leq c'n$ .

## Top-Down-Mergesort

```

1 def sort(array):
2     tmp = [0] * len(array) # [0, ..., 0] with same size as array
3     sort_aux(array, tmp, 0, len(array) - 1)
4
5 def sort_aux(array, tmp, lo, hi):
6     if hi <= lo:
7         return
8     mid = lo + (hi - lo) // 2
9     sort_aux(array, tmp, lo, mid)
10    sort_aux(array, tmp, mid + 1, hi)
11    merge(array, tmp, lo, mid, hi)

```

Analyse für  $m = hi - lo + 1$

$c_0$  für Zeile 6–7

$c_1$  für Zeile 8–9

$c_2 m$  für Merge-Schritt

## Top-Down-Mergesort: Analyse I

Laufzeit `sort_aux`

- ▶  $T(m) = c_1 + 2T(m/2) + c_2 m$  für  $m = 2^k, k \in \mathbb{N}_0$
- ▶  $T(1) = c_0$
- ▶ Rekursive Gleichung
- ▶ Wir suchen obere Schranke, die nur von  $m$  abhängt.

## Top-Down-Mergesort: Analyse II

Betrachte  $m = 2^k$  mit  $k \in \mathbb{N}_{>0}$

$$\begin{aligned}
 T(m) &= c_1 + 2T(m/2) + c_2m \\
 &= c_1 + 2(c_1 + 2T(m/4) + c_2(m/2)) + c_2m \\
 &= c_1(1+2) + 2mc_2 + 4T(m/4) \\
 &= c_1(1+2) + 2mc_2 + 4(c_1 + 2T(m/8) + c_2(m/4)) \\
 &= c_1(1+2+4) + 3mc_2 + 8T(m/8) \\
 &= \dots \\
 &= c_1 \left( \sum_{i=0}^{k-1} 2^i \right) + kmc_2 + 2^k c_0 \\
 &= c_1 \left( \sum_{i=0}^{k-1} 2^i \right) + c_2m \log_2 m + mc_0 \quad (k = \log_2 m, 2^k = m) \\
 &\leq c_1 k 2^{k-1} + c_2m \log_2 m + mc_0 \\
 &\leq c_1 m \log_2 m + c_2m \log_2 m + mc_0 \\
 &\leq (c_0 + c_1 + c_2)m \log_2 m \quad (\log_2 m = k \geq 1)
 \end{aligned}$$

## Top-Down-Mergesort: Analyse III

$m$  keine Zweierpotenz?  $2^{k-1} < m < 2^k$

$$\begin{aligned}
 T(m) &= c_1 + T(\lfloor m/2 \rfloor) + T(\lceil m/2 \rceil) + c_2m \\
 &\leq c_1 + 2T(2^k/2) + c_2m \\
 &\leq c_1 + 2T(2^{k-1}) + c_2m \\
 &< 2cm \log_2(2m) \quad (2^k < 2m, \text{ da } m > 2^{k-1}) \\
 &= 2cm(\log_2 2 + \log_2 m) \\
 &= 2cm(1 + \log_2 m) \leq 4cm \log_2 m \quad (1 \leq \log_2 m \text{ für } m \geq 2)
 \end{aligned}$$

Obere Schranke  $c'm \log_2 m$  gilt allgemein (für irgendein  $c'$ )

Untere Schranke?

$$T(m) = c_1 \sum_{i=0}^{k-1} 2^i + c_2m \log_2 m + mc_0 \geq c_2m \log_2 m$$

Untere Schranke  $cm \log_2 m$  (für irgendein  $c$ )

## Top-Down-Mergesort: Analyse IV

sort?

- ▶ Aufruf von `sort_aux` mit  $m = n =$  Länge der Eingabe
- ▶ Anlegen/Kopieren von Array geht in linearer Zeit  
→ kann durch Anpassung der Konstanten abgedeckt werden.

**Theorem**

*Top-Down-Mergesort hat leicht überlineare Laufzeit, d.h. es gibt Konstanten  $c, c', n_0 > 0$ , so dass für alle  $n \geq n_0$ ,  $cn \log_2 n \leq T(n) \leq c'n \log_2 n$ .*

## A8.2 Zusammenfassung

# Zusammenfassung

- ▶ **Mergesort** hat auch in der Top-Down-Variante leicht überlineare Laufzeit.