5.3 Mergesort

Wir wollen folgende Zahlen sortieren (aufsteigend nach Größe):

A: 8 3 9 6 3 11 7 12

↳ Idee: ① Finde Minimum in A.

② Kopiere Minimum nach B (Array)

③ Lösche Minimum in A

Durchlauf

1 B: 3

2 B: 3 3

3 B: 3 3 6

⋮

8 B: 3 3 6 7 8 9 11 12

Laufzeit ?

Speicherplatz ?

Verwaltet man einen Pointer auf das Ende von B und hat man einen Pointer auf das aktuelle Minimum können ② und ③ in $O(1)$ ausgeführt werden.

Für ① braucht man im schlechtesten Fall im

1. Durchlauf	$n-1$	Schritte/Vergleiche
2. "	$n-2$	"
:		
i. "	$n-i$	"

$\Rightarrow$ Insgesamt $\sum_{i=1}^{n} n-i = n^2 - \frac{n}{2}\cdot(n+1) = n^2 - \frac{n^2}{2} - \frac{n}{2}$

$= \frac{n^2}{2} - \frac{n}{2} \in O(n^2)$

Vergleiche

Satz 5.1 gibt eine untere Schranke für die Anzahl der Vergleiche von $\Omega(n \log n)$.

Speicherplatz ? $2 \cdot A$

↳ Auch das geht besser: „In-Place-Sorting"

Zurück zur Laufzeit:

Der Algorithmus „Mergesort" kommt mit $O(n \log n)$ Vergleichen aus: Dabei wird das algorithmische Prinzip „Divide and Conquer" (Teile und herrsche) verwendet.

Zunächst am obigen Beispiel:

8 3 9 6 | 3 11 7 12

Teile: 8 3 9 6 | 3 11 7 12

Teile: 8 3 | 9 6 | 3 11 | 7 12

Teile: 8 | 3 | 9 | 6 | 3 | 11 | 7 | 12

Nun kann jedes dieser 8 Teilprobleme in $O(1)$ gelöst werden.

Die „Teillösungen" müssen nur noch zusammengefügt werden.

z.B. Teillsg. 1. Prob. 8
" 2. " 3 $\Big\} \rightarrow$ 3 | 8

3 8 | 6 9 ‖ 3 11 | 7 12

3 6 8 9 | 3 11 7 12

3 3 6 7 8 9 11 12

$\log n + 1$

Divide-and-Conquer-Algorithmen bestehen aus 3 wesentlichen Schritten:

Divide: Problem in Teilprobleme aufteilen

Conquer: Rekursiv die Teilprobleme lösen, wenn sie „klein genug" (z.B. in $O(1)$ lösbar) sind.

Combine: Teillösungen zur Gesamtlösung vereinigen

<u>Algorithmus 5.2</u>: <u>Mergesort(A, p, r)</u>

<u>Input</u>: Subarray von $A = [1, \dots n]$ der bei p beginnt und bei r endet ($A[p \dots r]$)

<u>Output</u>: Den sortierten Subarray

```
IF p < r THEN
    q ← ⌊(p+r)/2⌋                  ← Divide
    Mergesort(A, p, q)          }
    Mergesort(A, p+1, r)        }  ← Conquer
    Merge(A, p, q, r)              ← Combine
```

- Sortieren von $A = (A[1], \dots, A[n])$: Mergesort$(A, 1, n)$
- Solange $p < r$ wird geteilt/halbiert.
- Falls $p \geq r$ besteht der Subarray nur noch aus einem Element und ist damit sortiert.

Noch zu klären:

- $\lfloor x \rfloor$ Gauß-Klammer

  $\lfloor x \rfloor = \max \{ k \in \mathbb{Z} : k \leq x \}$

  z.B. $\lfloor 4{,}3 \rfloor = 4$, $\lfloor -2{,}3 \rfloor = -3$, $\lfloor 5 \rfloor = 5$

- Merge(A, p, q, r)

  ↳ Folie

  ↳ ausführlich am Freitag in der gr. Übg

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
for i ← 1 to n1
     do L[i] ← A[p + i − 1]
for j ← 1 to n2
     do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
     do if L[i] ≤ R[j]
           then A[k] ← L[i]
                i ← i + 1
           else A[k] ← R[j]
                j ← j + 1
```

Wie sieht die Laufzeit aus?

Satz 5.2:

Mergesort hat eine Laufzeit von $O(n \log n)$ (für einen n-elementigen Array A).

Beweis:

Zunächst runden wir n auf die nächste 2er-Potenz.
(in O-Notation ändert sich dadurch nichts, $2^{k-1} < n \leq 2^k$, Faktor 2)
Wir erhalten in jedem Schritt also einen Subarray der Größe $\frac{n}{2}$.

$T(n)$ bezeichne die Laufzeit von Mergesort für einen n-elementigen Array A.

Divide: $O(1)$

Conquer: $2 \cdot T(\frac{n}{2})$

Combine: $O(n)$

Damit erhalten wir folgende Beziehung:

$$T(n) = \begin{cases} O(1), & n = 1 \\ O(1) + 2 \cdot T(\frac{n}{2}) + O(n) = 2 \cdot T(\frac{n}{2}) + O(n), & \text{für } n \geq 2 \end{cases}$$

T(n) ist über eine sog. „Rekursionsgleichung" definiert (mehr im nächsten Kapitel).

Für geeignete Konstanten d, g können wir T(n) auch folgendermaßen schreiben.

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \underbrace{d \cdot n}_{\text{statt } O(n)}, \quad n \geq 2, \quad d \in \mathbb{R}$$

$$T(1) = O(1) = g, \quad g \in \mathbb{R}$$

Nun müssen wir zeigen, dass $T(n) \leq c \cdot n \log n$ für ein geeignetes c.

Induktions-Anfang:

$$n = 2: \quad T(2) = 2 \cdot T(1) + d \cdot 2 = 2 \cdot g + 2d = 2(g + d)$$