5.3 Schranken fürs Sortieren

5.3.1 Schranken

Wir haben gesehen:

- Sortieren durch Auswählen ("Selection Sort") : $O(n^2)$
- Sortieren mit Mergesort : $O(n \log n)$

Geht das noch schneller?

Allgemeiner interessiert man sich für obere und untere Laufzeitschranken für Algorithmen!

Beispiele:

	Best Case (min Laufzeit)	Worst Case (max Laufzeit)
DFS	$\Omega(n)$	$O(n)$
BFS	$\Omega(n)$	$O(n)$
Insert Bin. Suchbaum	$\Omega(1)$	$O(n)$
Insert AVL-Baum	$\Omega(\log n)$	$O(\log n)$
Binäre Suche	$O(1)$	$O(\log n)$

- Asymptotische Abschätzungen für Laufzeit
- Auch für Speicherplatz einsetzbar

Beachte: Oft gibt es noch „Schlupf" zwischen der Worst-Case-Laufzeit, die man beweisen kann und der, die man wirklich hat! (→ Pessimistische Abschätzungen)

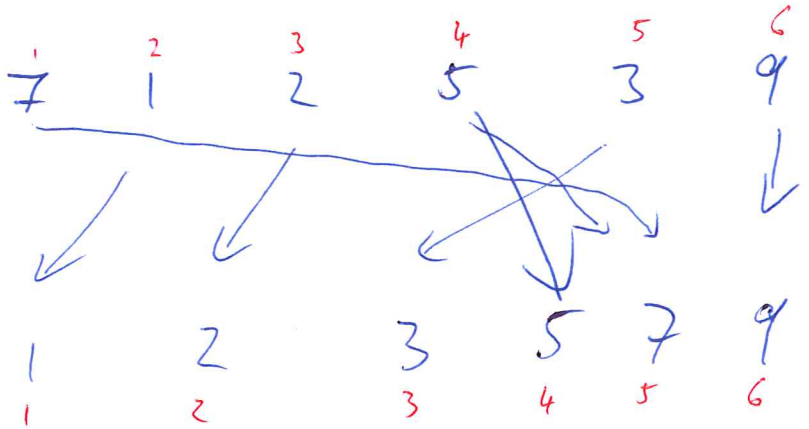
Allgemein:

- Obere Schranke:
  für Wort-Case-Laufzeit von Algorithmus, der Problem korrekt löst

- Untere Schranke:
  für Worst-Case-Laufzeit von Algorithmus, der Problem korrekt löst

5.3.2 Permutationen

Was bedeutet Sortieren?



Nicht notwendig: Objekte verschieben! (bzw. umkopieren)
Notwendig: Relative Anordnung identifizieren!

Also:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 1 & 2 & 4 & 3 & 6 \end{pmatrix}$$

Wie muss man verschieben?

Oder

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 5 & 4 & 1 & 6 \end{pmatrix}$$

Wo steht was?

Definition 5.4 (Permutation)

Eine Permutation π ist eine Umsortierung von n Objekten, d.h. eine bijektive Abbildung

$$\pi : \{1, \dots, n\} \to \{1, \dots, n\}$$
$$i \mapsto \pi(i)$$

Man schreibt π auch auf folgende Weise:

$$\begin{pmatrix} 1 & \dots & n \\ \pi(1) & \dots & \pi(n) \end{pmatrix}$$

- d.h. als Wertetabelle.

Beobachtung 5.5

Es gibt genau $n!$ verschiedene Permutationen von n Objekten.

5.3.3 Eine untere Laufzeitschranke

Satz 5.6

Für n Objekte $x_1, \dots, x_n$ benötigt man zum Sortieren mindestens $\Omega(n \log n)$, wenn man die Objekte nur paarweise vergleichen kann.

Beweis:

(1) Zunächst hat man $n! = n(n-1) \cdots 2 \cdot 1$ viele mögliche Permutationen.

(2) Jeder Vergleich von zwei Objekten x_i und x_j liefert zwei mögliche Ergebnisse:

$$x_i \leq x_j \quad \text{oder} \quad x_i > x_j$$

← kein "=" bei Totalordnung

(3) Entsprechend teilt jeder Vergleich die Menge der noch möglichen Permutation in zwei Teilmengen:

die mit $x_i \leq x_j$

die mit $x_i > x_j$

(4) Im Worst Case bleibt jeweils die größere Teilmenge übrig - das liegt am Input, auf den der Algorithmus keinen Einfluss hat.

(5) Bestenfalls kann man also eine Halbierung erzwingen.

(6) Bis man eine eindeutige Permutation unter den $n!$ Möglichkeiten sicher identifiziert hat, braucht man also mindestens $\log_2(n!)$ Vergleiche.

(7)
$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i$$
$$\geq \sum_{i=\frac{n}{2}}^{n} \log_2 i$$
$$\geq \frac{n}{2} \cdot \log_2 \frac{n}{2}$$
$$= \frac{n}{2}\left(\log_2 n - 1\right) \in \Omega(n \log n)$$

□