Sortieralgorithmen

Weiterführende Literatur:

- *Wikipedia-Artikel „Sortierverfahren“*

Klassifizierung der Sortieralgorithmen

Interne vs. externe Verfahren[1]

Bei *internen Sortierverfahren* ist stets ein *direkter Zugriff* auf *alle* zu sortierenden Elemente notwendig. Alle Elemente müssen *gleichzeitig* im Hauptspeicher liegen.

internen Sortierverfahren
direkter Zugriff
alle

Bei *externen Sortierverfahren* ist der Zugriff auf einen *Teil* der zu sortierenden Elemente beschränkt. Nur ein Teil der Daten muss gleichzeitig im *Hauptspeicher* liegen. Dieses Verfahren eignet sich für Sortierung von *Massendaten* auf *externen Speichermedien*.[2]

externen Sortierverfahren
Teil
Massendaten
externen Speichermedien

Vergleichsbasierte vs. Nicht-Vergleichsbasierte Verfahren[3]

Beim vergleichsbasierten Sortieren *vergleicht* der Algorithmus mehrfach jeweils *zwei Elemente* miteinander. Die Elementen werden aufgrund aufgrund ihrer *relativen Position* vertauscht. Beispiele: `QuickSort`, `MergeSort`

vergleicht
zwei Elemente

Beim nicht-vergleichsbasiertes Sortieren benötigt der Algorithmus *keinen direkten Vergleich* zwischen zwei Elementen, er *zählt* stattdessen die Werte oder betrachtet *„einzelne Stellen“* Beispiele: `CountingSort`, `RadixSort`

keinen direkten Vergleich
zählt
„einzelne Stellen“

Stabil vs. Instabil[4]

- stabiles Sortierverfahren:
  → Sortierverfahren, welches die Eingabereihenfolge von Elementen mit *gleichem Wert* beim Sortieren *bewahrt*

  Insbesondere dann wichtig, wenn hintereinander nach *mehreren Kriterien* sortiert wird.

In-Place vs. Out-Of-Place

- in-place (in situ)
  - Speicherverbrauch unabhängig von Eingabegröße
    → braucht nur eine konstante Menge an zusätzlichem Speicher
    → überschreibt im Allgemeinen die Eingabe- mit den Ausgabedaten
- out-of-place (ex situ)

[1] *Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19*, Seite 34.
[2] Saake und Sattler, *Algorithmen und Datenstrukturen*, Seite 124.
[3] *Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19*, Seite 35.
[4] *Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19*, Seite 36.

- Speicherverbrauch abhängig von Eingabegröße
  → Speicherverbrauch steigt mit Anzahl der zu sortierenden Elemente

Achtung: Aufrufstapel *Rekursive Algorithmen*, deren Aufruftiefe von der Eingabegröße abhängt, arbeiten genaugenommen *out-of-place*, denn für die Funktionsschachteln auf dem Aufrufestapel wird Speicherplatz benötigt. Manchmal bezeichnet man aber auch solche Algorithmen mit einem Speicherverbrauch von $\mathcal{O}(log(n))$ als in-place.

Laufzeitkomplexität

Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19, Seite 38

- für die Laufzeitkomplexität unterscheidet man verschiedene Fälle:
  - Best-Case
  - Average-Case
  - Worst-Case
- adaptive Sortierverfahren:
  - Laufzeit abhängig vom *Grad der Vorsortierung*
    → *schneller*, wenn Eingabe schon *„einigermaßen" sortiert* ist
    → Laufzeit in *Best-Case* und *Worst-Case unterschiedlich*
- *untere Schranken* für die Laufzeit (n: Anzahl an Elementen):
  - vergleichsbasiertes Sortieren: nicht besser möglich als $\mathcal{O}(log(n))$
  - nicht-vergleichsbasiertes Sortieren: lineare Laufzeit möglich

Vergleich der Sortieralgorithmen

Laufzeit[5]

	Best	Average	Worst
Binary Tree Sort	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n^2)$
Bubblesort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Vergleiche	$n-1$	$\sim\frac{n^2}{2}$	$\sim\frac{n^2}{2}$
Kopieraktionen	0	$\sim\frac{n^2}{4}$	$\sim\frac{n^2}{2}$
Heapsort	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$
Insertionsort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Vergleiche	$n-1$	$\sim\frac{n^2}{4}$	$\sim\frac{n^2}{2}$
Kopieraktionen	0	$\sim\frac{n^2}{4}$	$\sim\frac{n^2}{2}$
Mergesort	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$
Quicksort	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n \cdot \log(n))$	$\mathcal{O}(n^2)$
Selectionsort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Vergleiche	$\sim\frac{n^2}{2}$	$\sim\frac{n^2}{2}$	$\sim\frac{n^2}{2}$
Kopieraktionen	0	$3n$	$3n$

Implementation

	Kontrollstrukturen	Hilfsvariablen	Hilfsmethoden
Bubblesort	do while, for (bis vorletztes), if	t = getauscht	tausche
Insertionsort	for (ab zweitem), while	m = merker	
Selectionsort	while, for (ab zweitem)	m = markierung	tausche

Literatur

[1] *Algorithmen und Datenstrukturen: Tafelübung 11, WS 2018/19.* `https://www.studon.fau.de/file2567217_download.html`. FAU: Lehrstuhl für Informatik 2 (Programmiersysteme).

[2] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2. Sortieren, Suchen, Komplexität.* `https://www.studon.fau.de/file2566441_download.html`.

[3] Gunter Saake und Kai-Uwe Sattler. *Algorithmen und Datenstrukturen. Eine Einführung in Java.* 2014.

[5] *Qualifizierungsmaßnahme Informatik: Algorithmen und Datenstrukturen 2*, Seite 35.

[4] *Wikipedia-Artikel „Sortierverfahren“*. https://de.wikipedia.org/wiki/Sortierverfahren.