Sortieralgorithmen im Vergleich

	Schlüsselvergleiche			Datenbewegungen			Laufzeit		
	C_{best}	C_{avg}	C_{worst}	M_{best}	M_{avg}	M_{worst}	T_{best}	T_{avg}	T_{worst}
Insertion-Sort	n	-	n^2	n	-	n^2	n	-	n^2
Selection-Sort	n^2	n^2	n^2	n	n	n	n^2	n^2	n^2
Merge-Sort	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$
Quicksort	$n \log n$	-	n^2	n	-	$n \log n$	$n \log n$	$n \log n$	n^2
Heapsort	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$	$n \log n$
Bucket-Sort	0	0	0	n	n	n	n	n	n
Radix-Sort	0	0	0	n	n	n	n	n	n

Insertion-Sort

Stabil: Ja

Beschreibung: *Insertion-Sort* folgt dem Prinzip des Sortierens durch Einfügen. Dieser Algorithmus geht davon aus, dass links der aktuellen Position j bereits alles sortiert ist. Das j-te Element wird in die jeweils sortierte Reihenfolge eingeordnet, indem alle Elemente, die größer sind, der Reihe nach einen Schritt nach rechts rücken und dadurch schließlich Platz an der richtigen Stelle der Teilfolge schaffen.

Diskussion: *Insertion-Sort* erreicht den Best-Case falls das Feld bereits sortiert ist, und den Worst-Case falls das Feld umgekehrt sortiert ist.

Selection-Sort

Stabil: Nein

Beschreibung: *Selection-Sort* folgt dem Prinzip des Sortierens durch Auswahl. Im Unterschied zu Insertion-Sort wählt dieser Algorithmus das kleinste Element zwischen aktueller Position j und letzter Position n aus. Dieses wird dann in nur einer Datenbewegung an die Position j getauscht.

Diskussion: Einsatz von Selection-Sort falls

- die Bewegungen von Datensätzen teuer sind, und die
- Vergleiche zwischen den Schlüsseln billig.

Merge-Sort

Stabil: Ja

Beschreibung: Merge-Sort folgt dem Entwurfsprinzip *„Teile und Erobere" („divide and conquer")*:

- *Teile:* die n-elementige Folge in der Mitte in zwei Teilfolgen.
- *Erobere:* Sortiere beide Teilfolgen rekursiv mit *Merge-Sort*.
- *Kombiniere:* Verschmelze die beiden Teilfolgen zu einer sortiereten Gesamtfolge.

Diskussion:

- Merge-Sort benötigt $\Theta(n)$ zusätzlichen Speicherplatz.
- Als alternative Datenstruktur eignen sich verkettete Listen, da alle Teilfolgen nur sequentiell verarbeitet werden. Dann werden keine Daten bewegt, nur Zeiger verändert. $\Rightarrow$ Merge-Sort ist deshalb auch gut als externes Sortierverfahren geeignet (klassisch: Sortieren auf Bändern).

Quicksort

Stabil: Nein

Beschreibung: Quicksort folgt ebenfalls dem Entwurfsprinzip *„Teile und Erobere"*:

- *Teile:* Wähle „Pivotelement" k aus A. Dann teile A ohne k in zwei Teilfolgen A_1und A_2, so dass gilt:
  - A_1 enthält nur Elemente $\leq k$
  - A_2enthält nur Element $\geq k$
- *Erobere:* Sortiere A_1 und A_2:
  - Rekursiver Aufruf: $Quicksort(A_1)$;
  - Rekursiver Aufruf: $Quicksort(A_2)$;
- *Kombiniere:* Bilde A durch Hintereinanderfügen in der Reihenfolge A_1, k, A_2.

Diskussion: Abschließende Bemerkungen zu *Quicksort:*

- Quicksort ist das beste Verfahren in der Praxis. (Anm.: umstritten)
- Es gibt viele Varianten, auch solche, die den Worst-Case einer sortierten Folge verhindern:
  - zufällige Wahl des Pivoelements
  - mittleres (Median) von zufällig gewählten Elementen
- Bei jedem rekursiven Aufruf werden die aktuellen Werte der Parameter und der lokalen Variablen auf den Stack gelegt (Operation „pusch"), und nach Beendigung des rekursiven Aufrufs wieder vom Stack geholt (Operation „pop"). Wir haben gesehen, dass der Stack für *Quicksort* im Worst-Case $\Theta(n)$ Platz benötigt.
- Man kann *Quicksort* mit Hilfe von Stapeln ohne rekursive Aufrufe implementieren. Dadurch kann man den zusätzlich benötigten Speicherplatz auf $\Theta(\log n)$ drücken.
- Quicksort erreicht z.B. den Worst-Case, falls man das Pivot-Element „ganz rechts" wählt und das das Feld aber *aufsteigend* sortiert ist, oder falls man das Pivot-Element „ganz links" wählt und das Feld aber *absteigend* sortiert ist. Allgemein: wenn man das Pivo-Element so wählt, dass bei jedem Schritt die zu untersuchende Liste um eins kleiner wird.
- Im Best-Case wird das Pivotelement stets so gewählt, dass die entstehenden Teillisten beide gleich groß sind.

Heapsort

Stabil: Nein

Beschreibung: *Heapsort* folgt wie *Selection-Sort* dem Prinzip des Sortierens durch Auswahl, wobei diese Auswahl jedoch geschickt organisiert ist. Dazu wird eine spezielle Datenstruktur, der *Heap*, verwendet.

Diskussion:

- Quicksort ist im Durchschnitt schneller als Heapsort, aber Heapsort benötigt nur konstant viel zusätzlichen Speicherplatz.
- Im Durchschnitt ist Heapsort nur schneller als Quicksort, falls Vergleiche auf den zu sortierenden Daten sehr aufwendig sind.

Einfaches Bucket-Sort

Stabil: Ja

Beschreibung: *Einfaches Bucket-Sort* gehört zu den linearen Sortierverfahren. Voraussetzung für *einfaches Bucket-Sort* ist ein kleiner endlicher Werte-Bereich der Schlüsselwerte, z.B. ganze Zahlen im Bereich 1 bis m.

Funktionsweise: Man führt für jeden möglichen Schlüssel Buckets (Kübel) ein und verteilt zunächst die Schlüssel nacheinander in ihre jeweiligen Buckets. In einem zweiten Schritt sammelt man die Schlüssel der Reihe nach aus den Buckets wieder auf.

Radix-Sort

Stabil: Ja

Beschreibung: *Radix-Sort* (aka *Sortieren durch Fachverteilung*) gehört ebenfalls zu den linearen Sortierverfahren. Voraussetzungen für *Radix-Sort* sind:

1. Schlüssel müssen aus Zeichen eines endlichen Alphabets bestehen.
2. Zwischen den Zeichen muss eine Totalordnung bestehen.
3. Die Länge der Schlüssel muss auf eine maximale Länge beschränkt sein.

Funktionsweise:

1. Der Algorithmus wechselt zwischen Verteilungsphase und Sammelphase.
2. Die Verteilungs- bzw. Sammelphase wird für jede Position d. Schlüssels, von der niederwertigsten bis zur höchstwertigen Position, durchlaufen.
3. Für jedes Zeichen aus dem Alphabet gibt es ein Fach.
4. In der Verteilungsphase bestimmt das an der aktuellen Stelle betrachtete Zeichen in welches Fach die Daten abgelegt werden.
5. In der Sammelphase werden die Datensätze aus den Fächern $F_0, ..., F_{l-1}$ so eingesammelt, dass die Datensätze in Fach F_{i+1} hinter F_i kommen.

Radix-Sort muss ein stabiler Algorithmus sein, sonst könnte er nicht funktionieren.