Sortieralgorithmen

**Insertion Sort**

Zuerst teilen wir das Array gedanklich in einen linken, sortierten Teil und einen rechten, unsortierten Teil. Der sortierte Bereich enthält zu Beginn bereits das erste Element, da ein Array mit einem einzigen Element immer als sortiert angesehen werden kann.

Dann betrachten wir das erste Element des unsortierten Bereichs und prüfen, an welcher Stelle im sortierten Bereich dieses eingefügt werden muss, in dem wir es mit seinem linken Nachbarn vergleichen. Im Beispiel ist die 2 kleiner als die 6, gehört damit also links neben die 6. Um dort Platz zu machen, schieben wir die 6 um eine Position nach rechts und setzen die 2 dann auf das freigewordene Feld. Dann schieben wir die Grenze zwischen sortiertem und unsortiertem Bereich um eine Position nach rechts.

Wir betrachten wieder das erste Element des unsortierten Bereichs, die 4. Diese ist kleiner als die 6, aber nicht kleiner als die 2 und gehört somit zwischen die 2 und die 6. Wir schieben also die 6 erneut um ein Feld nach rechts und setzen die 4 auf das freigewordene Feld.

Die Grundidee hinter dem Insertion Sort ist, dass der sortierte Teil der Liste mit jedem eingefügten Element wächst, während der unsortierte Teil schrumpft. Der Algorithmus ist einfach zu implementieren und eignet sich gut für kleine Listen oder bereits teilweise sortierte Listen. Allerdings ist die Laufzeit des Insertion Sorts im Durchschnitt etwas langsamer als bei effizienteren Sortieralgorithmen wie beispielsweise dem Quicksort oder Mergesort.

**Selection Sort**

Selection Sort ist ein einfacher Sortieralgorithmus, der eine Liste schrittweise sortiert, indem er das kleinste Element auswählt und an die richtige Position platziert.

Der Algorithmus durchläuft die Liste und sucht das kleinste Element. Sobald das kleinste Element gefunden wurde, wird es mit dem ersten Element der Liste vertauscht. Anschließend wird der Algorithmus auf den verbleibenden Teil der Liste angewendet, wobei das kleinste Element erneut gesucht und mit dem zweiten Element der Liste getauscht wird. Dieser Vorgang wird solange wiederholt, bis die gesamte Liste sortiert ist.

Bei Insertion Sort hatten wir die jeweils nächste unsortierte Karte genommen und dann in den sortierten Karten an der richtigen Stelle *eingefügt* ("inserted").

Selection Sort funktioniert gewissermaßen anders herum: Wir *wählen* ("select") die jeweils kleinste Karte aus den unsortierten Karten, um diese dann – eine nach der anderen – an die bereits sortierten Karten anzuhängen.

Die Grundidee hinter dem Selection Sort ist, dass bei jedem Durchlauf das kleinste Element an die richtige Position im sortierten Teil der Liste platziert wird. Der Algorithmus hat eine feste Anzahl von Vergleichen und Vertauschungen pro Durchlauf, unabhängig von der ursprünglichen Anordnung der Elemente. Allerdings ist die Laufzeit des Selection Sorts im Durchschnitt langsamer als bei effizienteren Sortieralgorithmen wie dem Quicksort oder Mergesort, insbesondere für größere Listen.

**Bubble Sort**

Bei Bubble Sort (auch: "Bubblesort") werden jeweils zwei aufeinanderfolgende Elemente miteinander verglichen und – wenn das linke Element größer ist als das rechte – werden diese vertauscht.

Der Algorithmus durchläuft die Liste mehrmals und vergleicht dabei jeweils zwei aufeinanderfolgende Elemente. Wenn das zweite Element kleiner als das erste Element ist, werden sie miteinander vertauscht. Dadurch "blubbeln" die größten Elemente nach oben, ähnlich wie Luftblasen in Wasser.

Dieser Vorgang wird solange wiederholt, bis keine Vertauschungen mehr erforderlich sind und die Liste vollständig sortiert ist. Die größten Elemente steigen dabei nach oben und finden nach und nach ihre endgültige Position.

Bubble Sort ist einfach zu verstehen und zu implementieren, jedoch nicht sehr effizient für große Listen. In der durchschnittlichen und schlechtesten Fallkomplexität benötigt Bubble Sort eine quadratische Laufzeit, was bedeutet, dass die Ausführungszeit mit der Anzahl der Elemente exponentiell steigt. Daher wird Bubble Sort normalerweise für kleinere Listen oder als Lehrbeispiel für Sortieralgorithmen verwendet.

**Quicksort**

Quicksort funktioniert nach dem "Teile-und-herrsche"-Prinzip ("divide and conquer"):

Als erstes teilen wir die zu sortierenden Elemente auf zwei Bereiche auf – einen mit kleinen Elementen (im folgenen Beispiel "A") und einen mit großen Elementen (im Beispiel "B").

Welche Elemente klein sind und welche groß, entscheidet dabei das sogenannte Pivot-Element. Das Pivot-Element kann ein beliebiges Element aus dem Eingabe-Array sein. (Welches man wählt, bestimmt die Pivot-Strategie, dazu später mehr.)

Das Array wird nun so umsortiert, dass die Elemente, die kleiner als das Pivot-Element sind, im linken Bereich landen, die Elemente, die größer als das Pivot-Element sind, im rechten Bereich landen, und dass das Pivot-Element zwischen den zwei Bereichen positioniert wird – und damit automatisch an seiner endgültigen Position.

Der Algorithmus wählt ein Pivotelement aus der Liste und partitioniert die restlichen Elemente so, dass Elemente kleiner als das Pivotelement links davon liegen und Elemente größer rechts davon. Dieser Schritt wird als Partitionierung bezeichnet.

Anschließend wird der Algorithmus rekursiv auf die beiden Teillisten angewendet, die durch das Pivotelement getrennt sind. Dieser Vorgang wird wiederholt, bis die Teillisten eine Größe von eins haben, was bedeutet, dass die Liste vollständig sortiert ist.

Die Wahl des Pivotelements beeinflusst die Effizienz des Algorithmus. Ein häufig verwendetes Verfahren ist die Wahl des letzten Elements oder eines zufälligen Elements als Pivotelement.

Quicksort hat im Durchschnitt eine gute Laufzeit und wird oft als einer der schnellsten Sortieralgorithmen betrachtet. Die durchschnittliche Laufzeit beträgt O(n log n), wobei n die Anzahl der Elemente in der Liste ist. In einigen Fällen kann die schlechteste Laufzeit jedoch O(n^2) erreichen, wenn das Pivotelement in jedem Schritt das kleinste oder größte Element ist.

**Mergesort**

Mergesort funktioniert nach dem „Teile-und-herrsche“-Prinzip („divide and conquer“):

Zunächst werden die zu sortierenden Elemente in zwei Hälften geteilt. Die entstandenen Sub-Arrays werden dann wieder geteilt – und wieder, bis Sub-Arrays der Länge 1 entstanden sind.

Nun werden in umgekehrter Richtung jeweils zwei Teil-Arrays so zusammengeführt ("gemerged"), dass jeweils ein sortiertes Array entsteht. Im letzten Schritt werden die zwei Hälften des ursprünglichen Arrays gemerged, so dass dieses letztendlich sortiert ist.

Mergesort ist ein Divide-and-Conquer-Sortieralgorithmus.

Der Algorithmus teilt die Liste in zwei Hälften und sortiert jede Hälfte separat durch rekursives Anwenden von Mergesort. Dieser Schritt wird so lange wiederholt, bis die Teillisten eine Größe von eins haben.

Dann werden die sortierten Teillisten zusammengeführt (gemergt), indem jeweils das kleinste Element von beiden Listen ausgewählt und in die endgültige sortierte Liste eingefügt wird. Dieser Vorgang wird wiederholt, bis alle Elemente in die sortierte Liste eingefügt sind.

Mergesort hat eine Laufzeit von O(n log n), wobei n die Anzahl der Elemente in der Liste ist. Es ist ein stabiler Sortieralgorithmus, das heißt, er erhält die relative Reihenfolge von Elementen mit dem gleichen Wert.

Mergesort ist aufgrund seiner effizienten Laufzeit und seiner Stabilität weit verbreitet. Es wird häufig verwendet, wenn eine stabile Sortierung erforderlich ist oder wenn es sich um externe Sortierungen handelt, bei denen die Daten nicht vollständig in den Arbeitsspeicher passen.

**Heapsort**

Der Heapsort-Algorithmus besteht aus zwei Phasen: In der ersten Phase wird das zu sortierende Array in einen Max Heap umgewandelt. Und in der zweiten Phase wird jeweils das größte Element (also das an der Baumwurzel) entnommen und aus den restlichen Elementen erneut ein Max Heap hergestellt.

Heapsort ist ein Sortieralgorithmus, der auf der Datenstruktur des Binärheaps basiert.

Zunächst wird aus der Liste ein Max-Heap erstellt, wobei das größte Element an der Wurzel des Heaps liegt. Ein Max-Heap ist ein binärer Baum, bei dem der Wert eines Knotens immer größer oder gleich dem Wert seiner Kinder ist.

Das größte Element, das sich an der Wurzel des Heaps befindet, wird entfernt und an die richtige Position in der sortierten Liste platziert. Anschließend wird der Heap neu geordnet, um das nächstgrößte Element an die Wurzel zu bringen.

Dieser Vorgang wird solange wiederholt, bis alle Elemente in die sortierte Reihenfolge gebracht wurden.

Heapsort hat eine Laufzeit von O(n log n), wobei n die Anzahl der Elemente in der Liste ist. Es ist ein instabiler Sortieralgorithmus, da die relative Reihenfolge von Elementen mit dem gleichen Wert verändert werden kann.

Heapsort wird aufgrund seiner effizienten Laufzeit und seines geringen zusätzlichen Speicherbedarfs häufig verwendet. Es ist jedoch etwas komplexer zu implementieren als andere Sortieralgorithmen wie Quicksort oder Mergesort.