Sortierverfahren

Insertion Sort

Insertion Sort geht davon aus, dass alles links vom aktiven Index in der Elementfolge bereits sortiert ist. Insertion Sort erhöht den Index bei jedem Durchlauf und sortiert das aktive Element in die bereits sortierte Teilfolge links ein. (Abb1.2; Skriptum S. 15).

Laufzeiten: Best Case: $\Omega(n)$
Worst Case: $O(n^2)$
Avg. Case: $\Theta(n^2)$

Das Suchverfahren ist stabil, da die Werte von links nach rechts, d.h. mit steigendem Index bearbeitet werden und dabei bei der Suche für die Position des aktuellen Wertes in der schon sortierten Teilfolge der Index in der while-Schleife nur so lange runtergezählt wird, wie größere Elemente gefunden werden. Hierdurch wird der aktuelle Wert hinter ggf. vorhandenen Einträgen mit gleichem Wert gespeichert, d.h. die Reihenfolge dieser Elemente wird nicht verändert. Bei der Beispielsortierung zeigt sich dieser Sachverhalt, wenn bei der dritten Verschiebeoperation die 2 in Slot 3 zwischen der schon einsortierten 2 und der 3 eingefügt wird.

Selection Sort

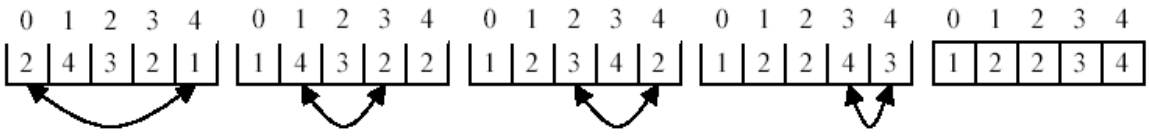
Geht wie Insertion Sort davon aus, dass alles links vom aktiven Element bereits sortiert ist. Jedoch sucht sich Selection Sort aus dem noch unsortierten Teil der Folge immer das Element mit dem niedrigsten Schlüssel um es dann mit dem 1. Element der noch unsortierten Teilfolge zu vertauschen und es somit in den sortierten Teil einfügt. (Abb. 2.1; Skriptum S. 25)

Laufzeiten: Best Case: $\Theta(n^2)$
Worst Case: $\Theta(n^2)$
Avg. Case: $\Theta(n^2)$

Schlüsselvergleiche: (Cmin, Cmax, Cavg) = $\Theta(n^2)$
Datenbewegungen: Mmin = 0; (Mmax, Mavg) = $\Theta(n)$

Selection Sort ist dann einzusetzen wenn Datenbewegungen teuer und Schlüsselvergleiche billig sind.

Wie an dem Beispiel ersichtlich, ist das Suchverfahren nicht stabil. Bei der ersten Iteration wird die 1 als minimales Element ausgewählt und mit der 2 getauscht. Dadurch ändert sich aber die Reihenfolge der beiden Zweien, was die Instabilität belegt.



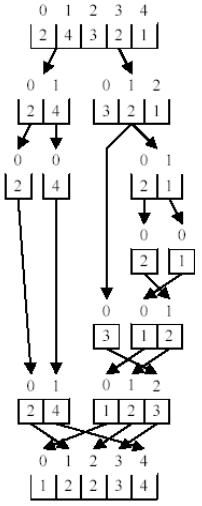
Merge Sort

Algorithmus funktioniert nach dem Teile und Erobere Prinzip: Teile das Problem in Teilprobleme auf, löse diese Probleme rekursiv, wenn sie klein genug sind löse sie direkt. Kombiniere anschließend die Lösungen der Teilprobleme zu einer lösung des Gesamtproblems.
Merge Sort teilt die unsortierte Folge immer in der Hälfte auf und macht dies so lange bis nur noch Einelementige Teilfolgen existieren. Dann werden diese Folgen mit Merge zu einer geordneten Folge rekombiniert. (Abb. 2.2; Skriptum S. 28)

Laufzeiten: Best, Worst, Avg Case = Θ (n*logn)

Schlüsselvergleiche: *C*max = Θ (n*logn); (*C*min, *C*avg) = Θ(nlogn)
Datenbewegungen: *M*max = Θ (n*logn); (*M*min, *M*avg) = Θ(nlogn)

Merge Sort braucht Θ(n) zusätzlichen Speicherplatz auf Grund des Platzhalterarrays B .



Das Suchverfahren ist stabil. Beim Splitten kann es per se zu keiner Reihenfolgeänderung kommen. Beim Mergen wird auf jeder Split-Ebene von den beiden beteiligten Teilfolgen jeweils die Linke bevorzugt, d.h. aus dieser Folge werden so lange Elemente in die Mergefolge eingefügt, wie sie kleiner oder gleich dem aktuellen Element der zweiten Menge sind. Somit kann es zu keiner Reihenfolgeänderung kommen.

Quick Sort

Ebenfalls ein Algorithmus vom Prinzip Teile und Erobere. Man bestimmt zuerst ein Pivot Element (z.B. das letzte Element der Folge) und bestimmt, wo das Pivot Element zum Schluss in der Folge stehen soll. Dazu bringt man alle kleineren Elemente in den linken Teil des Feldes und alle größeren Elemente in den rechten Teil des Feldes. Nach diesem Schleifendurchlauf sitzt das Pivot Element an seiner richtigen Endposition, alle kleineren Elemente liegen links, alle größeren rechts, allerdings noch unsortiert. Nun wir das Prinzip auf die beiden Teilfolgen rekursiv angewendet um auch sie zu sortieren.
(Abb. 2.4; Skriptum S. 35)

Laufzeiten : Best Case = Θ (n log n)
Worst Case = O (n^2)
Average Case = Θ (n*logn)
Schlüsselvergleiche: *C*min = Θ (nlogn)
Datenbewegungen: *M*min = Θ (n)

Quick Sort ist in der Praxis das beste Sortierverfahren.

Quick Sort ist nicht stabil, wenn nämlich für ein Array aus gleichwertigen Elementen das äußere rechte Element der betrachteten Teilfolge zum Pivot-Element gewählt wird, würde

dies immer an den Anfang der Teilfolge gebracht werden und dort bleiben. Insgesamt würde QuickSort das Array „invertieren“.

Heap Sort

Sortieren durch Auswahl, hierzu ist eine spezielle Datenstruktur, der Heap nötig. Er kann als binärer Baum veranschaulicht werden (Abb. 2.6; Skriptum S. 39). Ein binärer Baum kann Heap genannt werden, falls der Schlüssel eines Knotens mindestens so groß ist wie der seiner beiden Kinder, falls diese existieren. In einem Heap steht das größte Element an der Spitze, am Index 1, die Spitze muss größer sein, als seine beiden Kinder, die wiederum größer als deren Kinder sein müssen, usw. usf.
Ein Heap wird folgendermaßen konstruiert: Man trägt die Elemente der Reihe nach wie sie in der Eingabefolge stehen in einen binären Baum ein, jeder Knoten muss 2 Kinder haben.
Nun muss der Heap noch so umkonstruiert werden, dass jeder Sohn kleiner oder gleich seinem Vater ist. Dies wird durch Versickerung bewerkstelligt. Man nimmt den höchsten Index (=letztes Blatt des binären Baums von links nach rechts betrachtet) dividiert ihn durch 2, rundet eventuell ab und hat den Startpunkt gefunden (z.B. Baum mit 8 Elementen, größter Index ist 8, Hälfte ist 4 oder Baum mit 9 Elementen, Hälfte ist 4,5 => abgerundet 4)
Der Startpunkt ist immer der Knoten mit dem größten Index, der noch mindestens 1 Kind hat. Der Knoten an diesem Startpunkt wird mit seinen Söhnen verglichen, wenn er kleiner als einer oder beide seiner Söhne ist wird er mit dem größeren Sohn ausgetauscht. Danach wird der Index des Starpunktes um 1 erniedrigt und es wird wieder geprüft ob der Vater größer als seine Söhne ist. Dies wird solange gemacht, bis man am Index 1, der Wurzel, angelangt ist. Dann ist jeder Vater größer als sein/e Sohn/Söhne. (Abb. 2.8; Skriptum S. 43)

Methoden zum sortieren mit Heaps

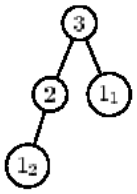
Absteigend: Zuerst wird aus der Eingabefolge ein Heap konstruiert. Dann wird die Wurzel (= größtes Element) aus dem Heap entfernt und das Element mit dem höchsten Index (= kleinstes Element) an die Wurzel gestellt. Noch ist die Heapeigenschaft nicht erfüllt, da das kleinste Element, das jetzt an der Wurzel steht, sicher nicht größer als seine beiden Kinder ist. Also wird die Wurzel „versickert“ und anschließend steht wieder das größte Element an der Wurzel, wird entfernt, usw. usf. Kurz gesagt: Wurzel entfernen, aus dem Rest wieder einen Heap machen, Wurzel entfernen.....

Aufsteigend: Hier wird das höchste Element, das in der Wurzel steht, mit dem kleinsten Element, das im Knoten mit dem höchsten Index steht vertauscht und nachher aus dem Heap entfernt. Das Element an der Wurzel wird wieder versickert und der Prozess beginnt von neuem, solange bis leer ist und die Elemente aufsteigend sortiert in z.B. einem Hilfsfeld stehen.

Laufzeiten: Best Case: Θ (n*logn)
Worst Case: Θ (n*logn)
Average Case: Θ (n*logn)

Schlüsselvergleiche: (Cmin, Cmax, Cavg) = Θ(nlogn)
Datenbewegungen: (Mmin, Mmax, Mavg) = Θ(nlogn)

Heap Sort ist nicht stabil.
Beispiel, gegeben sei die Folge $3,2,1_1,1_2$.
Nach dem Entfernen von 3 wandert die 1_2 von oben nach Links und überholt die 1_1. Die sortierte Folge ist danach $1_2 1_1 2\ 3$



Lineare Sortierverfahren

Diese Sortierverfahren nutzen die Eigenschaften der Schlüssel aus, wenn sie über einem Alphabet stehen z.B. Wörter über unserem Alphabet {a,b,....z,A,B,....Z} oder Dezimalzahlen.

Bucket Sort: Anzuwenden bei ganzzahligen Schlüsseln aus einem kleinen Wertebereich (0,1,....m)
Es werden m + 1 Buckets (Eimer) erstellt und die Elemente werden nacheinander in ihren passenden Bucket verteilt. Dies geschieht unter Berücksichtigung der Regeln des Alphabets. Nach der „Einkübelung“ sammelt man die Schlüssel der Reihe nach wieder aus den Küblen.

Sortieren durch Fachverteilung: Dieses Verfahren eignet sich für Schlüssel, welche alle aus dem selben Alphabet sind und gleiche Länge haben.
Die Schlüssel/Wörter werden zuerst nach ihrer niedrigwertigsten Position in Fächer einsortiert. Dann werden die Wörter der reihe nach den Fächern in der Sammelphase entnommen. Jetzt werden die Worte nicht mehr nach dem niedrigsten Index sortiert sondern nach dem 2. niedrigsten. Dies wird solange fortgesetzt bis man beim höchstwertigsten Index angelangt ist. Die Sammelphase gibt dann die Schlüssel in sortierter Folge aus. (Skriptum S.47)