

Sortierverfahren

Selection Sort

In einer N-fachen Schleife wird überprüft, welches Element das jeweils kleinste ist und mit dem Element an der N-ten Stelle getauscht. Es werden zwei Schleifen benötigt, von denen die erste von 1 bis N-1 läuft (bei jedem Durchlauf wird ja eins von N Elementen an seinen richtigen Platz gesetzt), die zweite läuft von der Position der ersten +1 bis zum Ende. Daher ergibt sich die Laufzeitordnung $O(n^2/2)$

Pseudocode für die Schleifen:

```
für i von 1 bis N-1
{
  für j von i + 1 bis N
  {
  }
}
```

Insertion Sort

Ähnlich dem Selection Sort, allerdings werden hier die Elemente nicht ausgetauscht, sondern das jeweils niedrigste Element eingefügt, der Rest wird nach hinten verschoben. Für das Verschieben wird eine weitere Schleife benötigt, die in unsere Laufzeitordnung eingreift. Best-case $O(n)$ für den Fall, dass alles sortiert ist, im ersten Schleifendurchlauf von "i" wird festgestellt, dass die Elemente[j] alle bereits sortiert sind und Schleife "i" wird nach dem ersten Durchlauf bereits abgebrochen
Average-case $O(n^2/4)$
Worst-case $O(n^2/2)$

Pseudocode:

```
für i von 2 bis N
{
  j = i
  t = Element[i]
  solange j > 0 und a[j-1] > t
  {
    rücke a[j-1] = a[j]
  }
  setze a[j] = t
}
```

Bubble Sort

Die Elemente werden an ihren Positionen in einer Schleife nacheinander miteinander verglichen und, je nach Größenunterschied, ausgetauscht. Werte, die in der Reihenfolge weiter nach vorne gehören, wechseln ihre Position in die vorderen Reihen bei jedem Schleifendurchlauf um eine Position, der bei jedem Durchlauf größte Wert erreicht bereits in einem Durchlauf seine Position. Deshalb werden bei jedem Schleifendurchlauf auch nur noch N-Schleifendurchlauf Elemente miteinander verglichen.
Laufzeitordnung $O(n^2/2)$

Pseudocode für die Schleifen:

```
für i von 1 bis N
{
  für j von 1 bis N-i
  {
  }
}
```

Quick Sort

Aus der ungeordneten Folge wird ein Element als Pivotelement ausgewählt. die übrigen Elemente werden so umgestellt, dass sich links vom Pivotelement nur minderwertigere, rechts davon nur höherwertige Elemente befinden. Die beiden Teile, die vom Pivotelement getrennt wurden, werden erneut geteilt und die Sortierung rekursiv durchgeführt.

Wegen der ständigen Teilung ist der Best-case "log n", hinzu kommt der Durchlauf aller Elemente (deshalb n-mal). Im Worst-case wird bei jedem Durchlauf jedes Element nochmal sortiert, das kommt allerdings so gut wie nie vor.

Best-case $O(n \log n)$

Worst-case $O(n^2)$

Pseudocode:

```
für i von 1 bis Index_des_Pivots - 1
{
  für j von Index_des_Pivots + 1 bis N
  {
  }
}
```

Merge Sort

Die zusammengesetzten Elemente werden mit jedem Schritt halbiert, bis wir die einzelnen Elemente haben. Dann werden sie in jedem Schritt mit dem jeweiligen Nachbarn wieder sortiert zusammengesetzt. Laufzeit: $O(n \log n)$.