

# Vergleich der einzelnen Methoden:

## Nach Eigenschaften:

z.B. Selection Sort: Wenn in der äußeren Schleife ein bestimmtes Element bearbeitet wird, sucht die Methode rechts das kleinste Element und holt es an die passende Stelle nach vor. Links stehen die Werte dann fix an ihren endgültigen Positionen. Man kann also jeden gefundenen Wert **sofort** als richtig eingeordnet weiterverwenden. (links statisch, rechts dynamisch)

z.B. Insertion Sort: Er transportiert Werte nach links. Dort ist ihr Platz aber nicht endgültig. Die rechten Elemente bleiben unverändert, bis sie drankommen. Man könnte also während des Programmlaufs noch weitere Elemente rechts anfügen, ohne die Sortierung zu stören. (rechts statisch, links dynamisch)

## Nach Laufzeit:

Wir interessieren uns hier nur für das Verhalten im Großen. Zeitverhalten  $n \log(n)$  bedeutet also nur die Proportionalität für den signifikanten Teil. Mathematiker würden  $O(n \log n)$  schreiben.

## Nach Stabilität:

Angenommen wir haben eine Adressliste mit Nachname, Vorname, Bundesland und Straße bereits richtig nach Namen sortiert. Jetzt wollen wir eine Sortierung nach Bundesländern. Nun hat etwa ein Neuntel der Daten den gleichen Vergleichsschlüssel. Was passiert nun mit den Namen – bleiben sie alphabetisch richtig (eine derartige Routine heißt **stabil**) oder werden sie vermengt? Einfacher: sortiere ich [2,3,4,5,3,1] mit einer stabilen Methode, so bleibt der rechte Dreier garantiert auch in der sortierten Liste rechts vom linken.

| Algorithmus | Ordnung              | stabil |   |
|-------------|----------------------|--------|---|
| Selection   | $n^2$                | ja     | im Mittel $n^2/2$ Vergleiche aber nur $n$ Austauschoperationen. Gut geeignet für große Datensätze, wo der Tausch zeitaufwändig ist  |
| Insertion   | $n^2$                | ja     | im Mittel $n^2/4$ Vergleiche und $n^2/8$ Tausche. Im schlechtesten Fall doppelt so viele. Wenn an eine sortierte Liste wenige neue Elemente angehängt werden, ist diese Methode ideal. (für ein einziges neues Element ist nur ein einziger Durchlauf nötig!) |
| Bubble      | $n^2$                | ja     | im Mittel $n^2/2$ Vergleiche und $n^2/2$ Austauschoperationen. Vergessen wir ihn.   |
| Shell       | $<n^{3/2}$           | nein   | die beste Methode wenn man nur einige 1000 Elemente hat. Aber nicht stabil!   |
| Quick       | $n \log n$ bis $n^2$ | nein   | bei zufälligen Listen besonders schnell   |
| Heap        | $n \log n$           | nein   | immer schnell, aber nicht stabil  |
| Merge       | $n \log n$           | ja     | doppelter Speicherbedarf, aber immer schnell und stabil!  |

## Lernziel:

Es gibt nicht 'den besten' Sortieralgorithmus. Aber eine Faustregel:

Der Umsichtige verwendet immer **ShellSort**,  
 der Unwissende **immer** Quicksort,  
 der Könner den **geeigneten** Algorithmus.