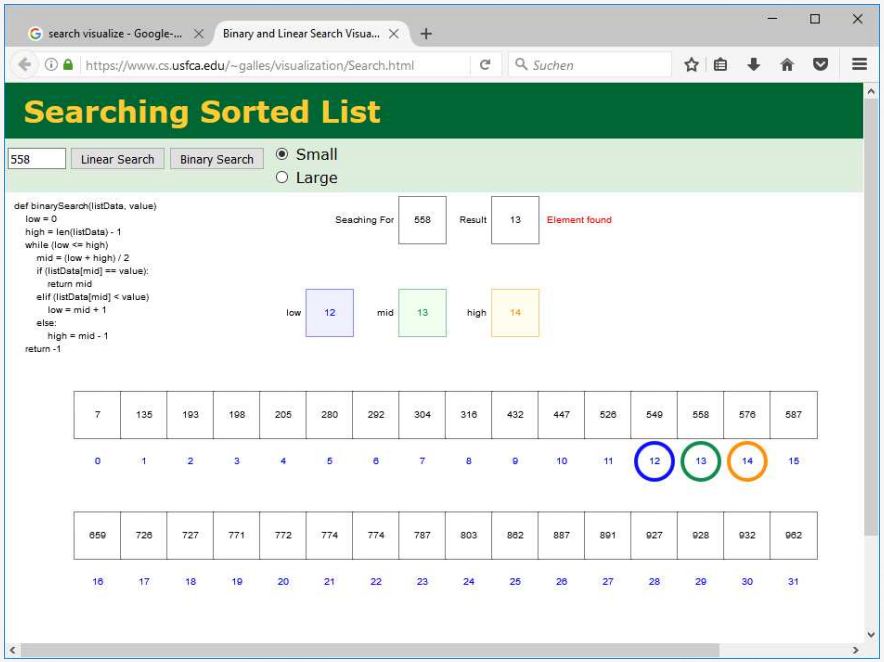
Suchen und Sortieren

Suchen: Visuelle Darstellung



https://www.cs.usfca.edu/~galles/visualization/Search.html

Big O: Vergleich von Algorithmen

Wann ist Algorithmus A besser als Algorithmus B?

- Algorithmus A verbraucht weniger Speicherplatz
  Algorithmus B ist schneller
- Oft verbrauchen schneller Algorithmen mehr Hauptspeicher!
- Folgerung: Optimierung der Laufzeit!

Wie schnell ist ein Algorithmus?

- Best Case
- Worst Case
- Average Case

Big O: Lineares Suchen

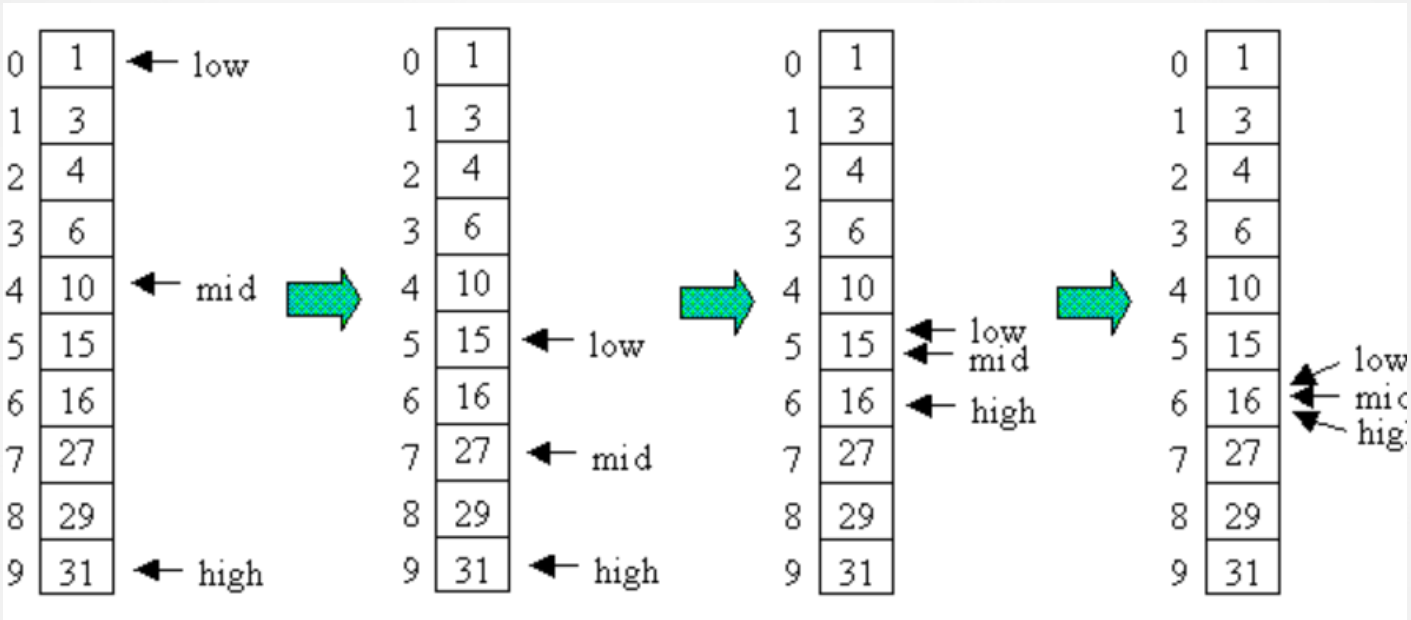
```
int linearSearch(int A[], int element) {
  for (i=0; i<A.length; i++) {
    if (A[i] == elem)
      return i;
  }
   return false;
}
```

Wann ist Algorithmus A besser als Algorithmus B?

- Algorithmus A verbraucht weniger Speicherplatz
  Algorithmus B ist schneller
- Oft verbrauchen schneller Algorithmen mehr Hauptspeicher!
- Folgerung: Optimierung der Laufzeit!

Suchen: Binäres Suchen

Suche nach 16:



Suchen: Binäres vs sequentielles Suchen

Gegeben: 1000 Schachteln mit sortieren Zahlen. Es wird eine bestimmte Zahl gesucht z. B. 345

Lineare Suche

Im Mittel (1+1000)/2 Suchschritte

Im Mittel
=(1+n)/2 Suchschritte

Binäre Suche

Im Mittel $\log_2 1000$ Suchschritte

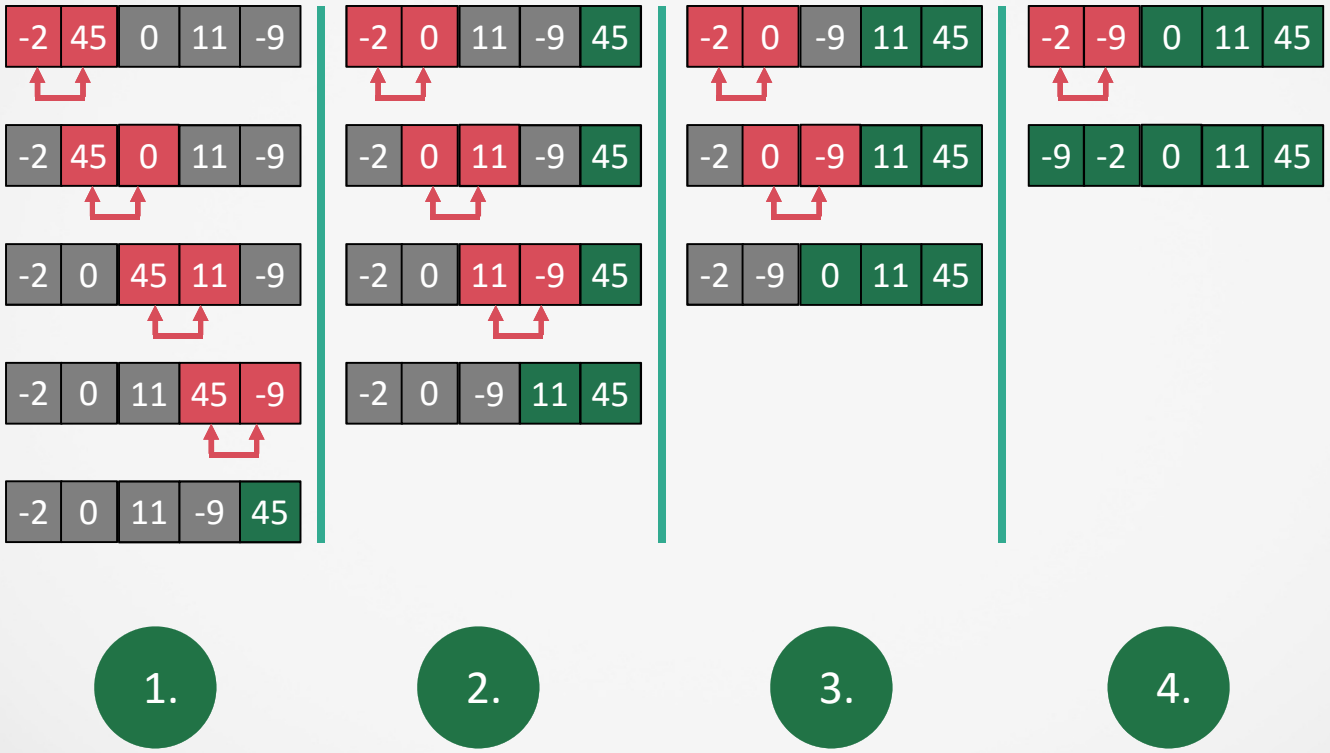
Im Mittel $\log_2 n$ Suchschritte

Sortieren: Bubble Sort I

Sortieren durch einfaches Vertauschen:

- Vergleiche das erste und zweite Element: Tausche es falls nötig!
- Vergleiche das zweite und dritte Element: Tausche es falls nötig!
- Vergleiche das dritte und vierte Element: Tausche es falls nötig!
- Mache die gleichen Schritte ab dem 2. Element!

Sortieren: Bubble Sort II



Sortieren: Bubble Sort III

Sortieren durch einfaches Vertauschen:

```
for(step=0;step<n-1;++step)
  for(i=0;i<n-step-1;++i) {
    if (data[i]>data[i+1])  {
      temp=data[i];
      data[i]=data[i+1];
      data[i+1]=temp;
    }
  }
```

Durchlauf durch die Arrayelemente

Vergleich zweier Elemente

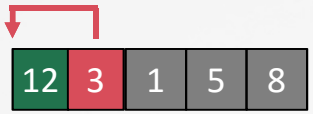
Vertauschen zweier Elemente

Sortieren: Insertion Sort I

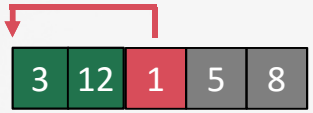
Sortieren durch Einfügen:
- Füge das zweite Element vor/hinter dem ersten an die richtige Stelle ein
- Füge das dritte Element an die richtige Stelle ein von den ersten zwei richtig sortierten Elementen ein
- Füge das vierte Element an die richtige Stelle ein von den ersten drei richtig sortierten Elementen ein

Sortieren: Insertion Sort II

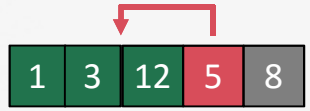
1.



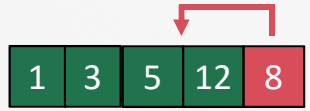
2.



3.

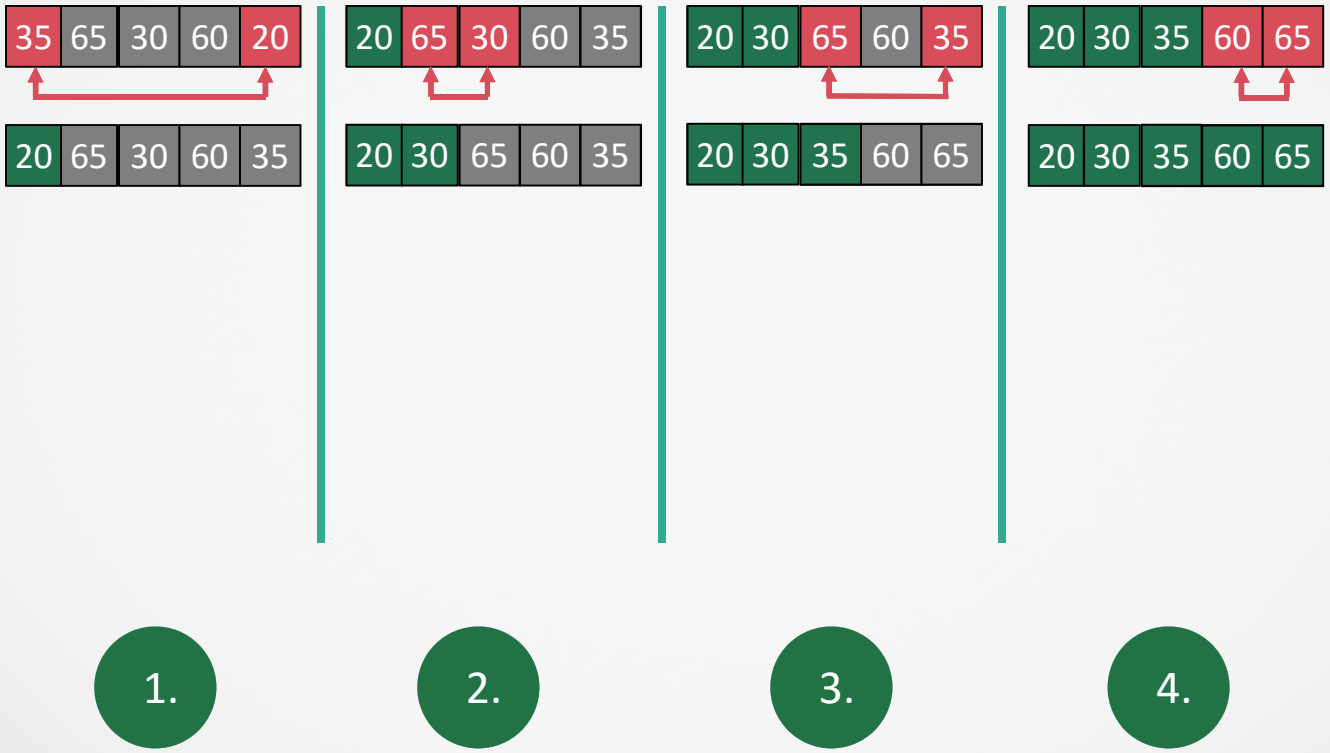


4.



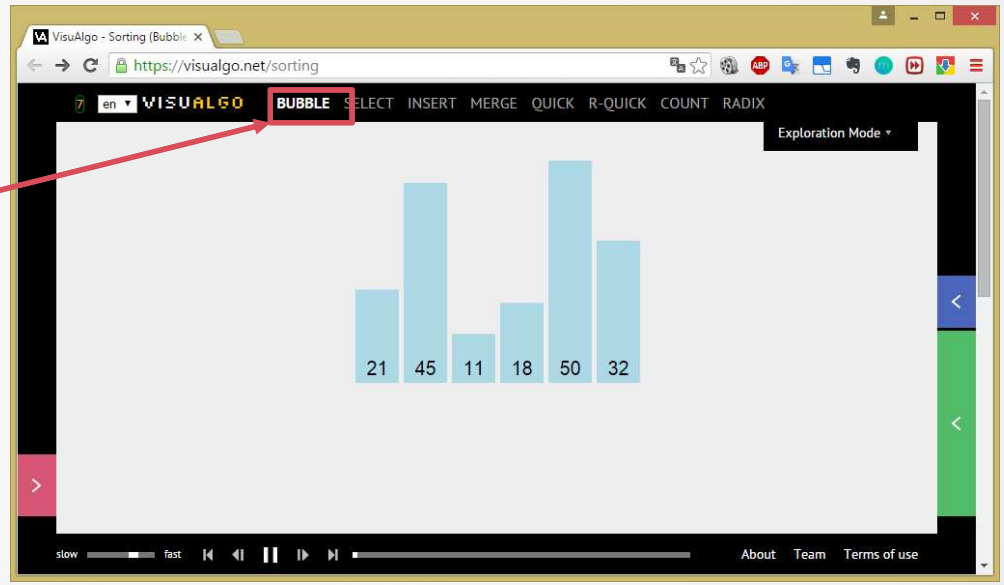
Ergebnis:

Sortieren: Selection Sort



Sortieren: Visuelle Darstellung I

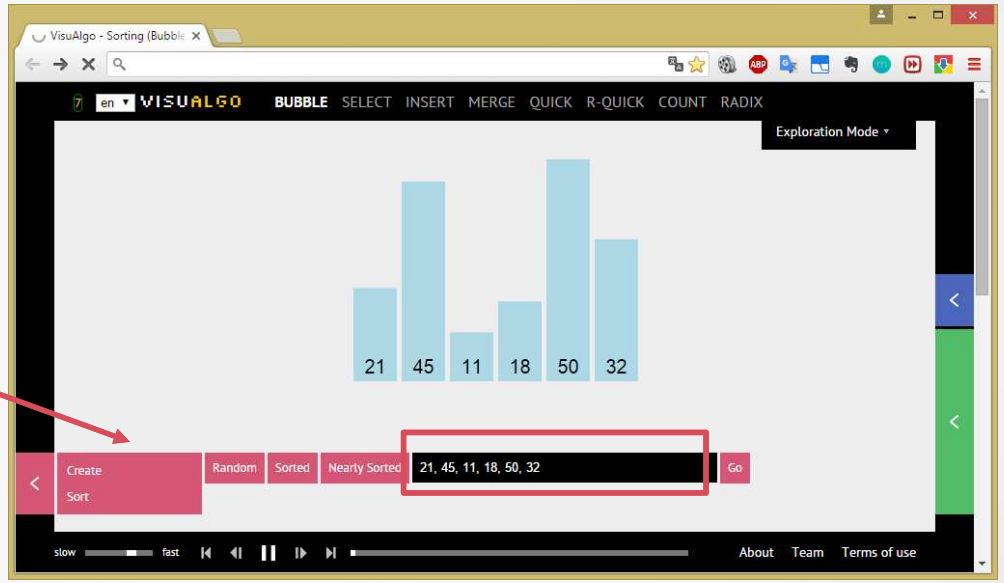
1. Sortieralgorithmus wählen



https://visualgo.net/sorting

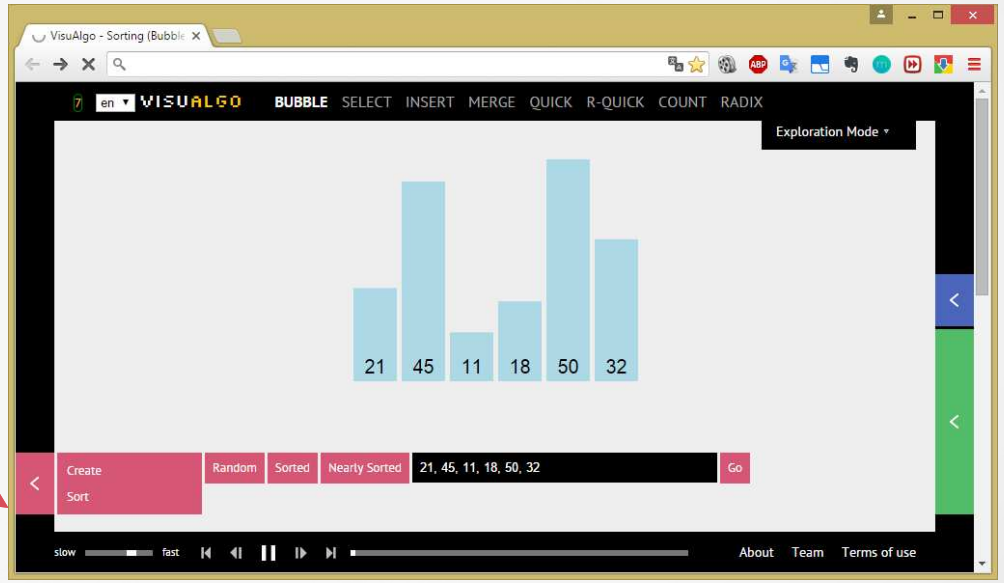
Sortieren: Visuelle Darstellung II

2. Daten eingeben



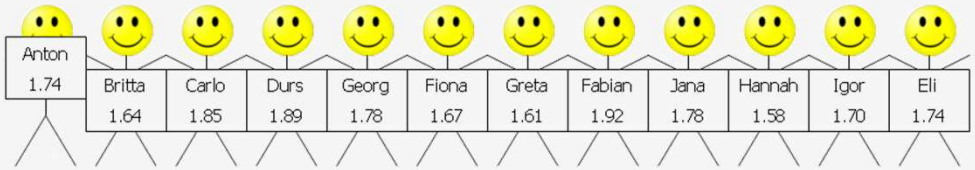
Sortieren: Visuelle Darstellung III

3. Sortieren

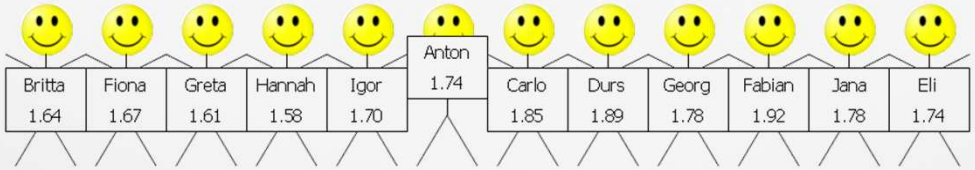


Sortieren: Quick Sort – Prinzip I

Anton, Britta, Carlo, ... wollen sich der Größe nach in einer Reihe aufstellen.
Zuerst werden alle genau vermessen.
Eine Person (z.B. die erste in der Reihe) wird als Vergleichsperson ausgewählt. Im vorliegenden Beispiel ist das Anton.

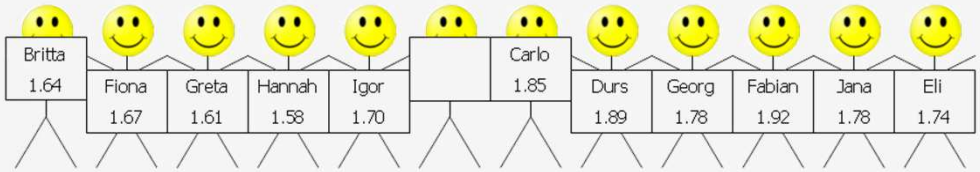


Alle anderen ordnen sich links bzw. rechts von der Vergleichsperson ein, je nachdem, ob sie kleiner oder größer gleich als die Vergleichsperson sind.



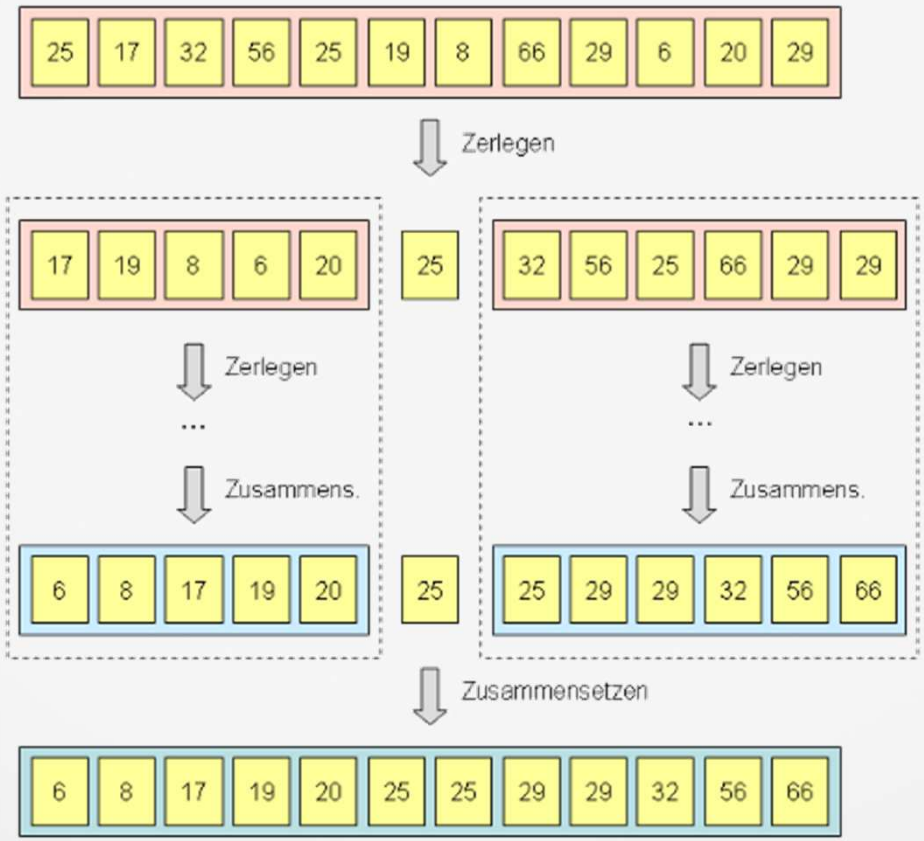
Sortieren: Quick Sort – Prinzip II

Anton bleibt jetzt auf seiner Position. Er nimmt nicht mehr an der Sortierung teil. Dasselbe Verfahren wird jetzt im linken und im rechten Bereich durchgeführt: Eine Person (z.B. die erste in der Reihe) wird wiederum als Vergleichsperson ausgewählt.



Und so weiter ...

Sortieren: Quick Sort – Beispiel



Sortieren: Quick Sort

Ebenso gut ist:

http://www.w3resource.com/javascript-exercises/searching-and-sorting-algorithm/index.php

Informationen zu Groß Oh:

https://www.cs.usfca.edu/~galles/cs245S08/lecture/lecture11.pdf

Graphische Darstellung von Quicksort:

http://me.dt.in.th/page/Quicksort/