

# Sortieren in Java

## Standardverfahren

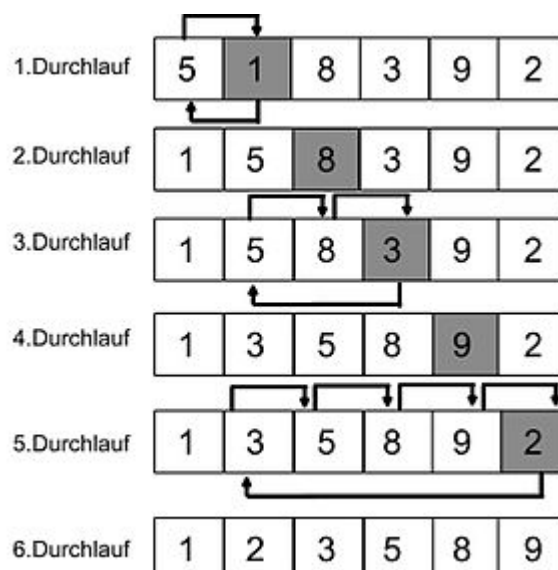


### Insertionsort

#### Quelltext

```
public static int[] Sort(int[] s) {  
    int zusortieren = 0;  
    for (int i=1; i<=s.length-1; i++) {  
        zusortieren = s[i]; //Zwischenspeichern des 1. Elements des unsortierten Teils  
        int j = i;  
        while (j > 0 && s[j-1] > zusortieren) {  
            s[j] = s[j-1]; //Verschieben von Elementen, die größer sind, um Einfügestelle frei zu machen  
            j--;           //Abfrage „> 0“ für den Fall, dass alle Elemente größer sind!  
        }  
        s[j] = zusortieren; //Einfügen des zu sortierenden Elements an die richtige Stelle  
    }  
    return s;  
}
```

Es gibt einen **sortierten** und einen **unsortierten** Teil im Array. Zu Beginn ist der **sortierte** Teil 1 Element groß (das 1. Array-Element) und mit jedem Durchlauf vergrößert sich dieser um 1 weiteres Element, indem jeweils das 1. Element des **unsortierten** Teils an die richtige Stelle im **sortierten** Teil eingefügt wird.



## Einzelsschrittdurchlauf am Beispiel „4,3,2,1“

Array s: **4**, 3, 2, 1 (in **Fettdruck** der bisher sortierte Bereich)

zusortieren = 0

for-Schleife (Start bei i=1, Ende nach i=3=Länge von s - 1)

i = 1

**zusortieren = s[i] = s[1] = 3**

j = i = 1

while-Schleife (solange j > 0 und s[j-1] > zusortieren)

j = 1 > 0 und s[j-1] = s[0] = 4 > **zusortieren = 3** (erfüllt!)

s[j] = s[j-1], also s[1] = s[0] = 4

Zwischenstand: 4, **4**, 2, 1

j--, also j = 0

j = 0 - nicht mehr > 0, also Ende while

s[j] = **zusortieren**, also s[0] = **3**

Zwischenstand: **3**, 4, 2, 1

i = 2

**zusortieren = s[i] = s[2] = 2**

j = i = 2

while-Schleife (solange j > 0 und s[j-1] > zusortieren)

j = 2 > 0 und s[j-1] = s[1] = 4 > **zusortieren = 2** (erfüllt!)

s[j] = s[j-1], also s[2] = s[1] = 4

Zwischenstand: 3, 4, **4**, 1

j--, also j = 1

j = 1 > 0 und s[j-1] = s[0] = 3 > **zusortieren = 2** (erfüllt!)

s[j] = s[j-1], also s[1] = s[0] = 3

Zwischenstand: 3, **3**, 4, 1

j--, also j = 0

j = 0 - nicht mehr > 0, also Ende while

s[j] = **zusortieren**, also s[0] = **2**

Zwischenstand: **2**, 3, 4, 1

i = 3

**zusortieren = s[i] = s[3] = 1**

j = i = 3

while-Schleife (solange j > 0 und s[j-1] > zusortieren)

j = 3 > 0 und s[j-1] = s[2] = 4 > **zusortieren = 1** (erfüllt!)

s[j] = s[j-1], also s[3] = s[2] = 4

Zwischenstand: 2, 3, 4, **4**

j--, also j = 2

j = 2 > 0 und s[j-1] = s[1] = 3 > **zusortieren = 1** (erfüllt!)

s[j] = s[j-1], also s[2] = s[1] = 3

Zwischenstand: 2, 3, **3**, 4

j--, also j = 1

j = 1 > 0 und s[j-1] = s[0] = 2 > **zusortieren = 1** (erfüllt!)

s[j] = s[j-1], also s[1] = s[0] = 2

Zwischenstand: 2, **2**, 3, 4

j--, also j = 0

j = 0 - nicht mehr > 0, also Ende while

s[j] = **zusortieren**, also s[0] = **1**

Zwischenstand: **1**, 2, 3, 4 (hier auch gleichzeitig Endstand!)

Ende for (Array ist sortiert!)

Rückgabe des Arrays

## Bubblesort („Sortieren durch Aufsteigen“, „Austauschsortieren“)

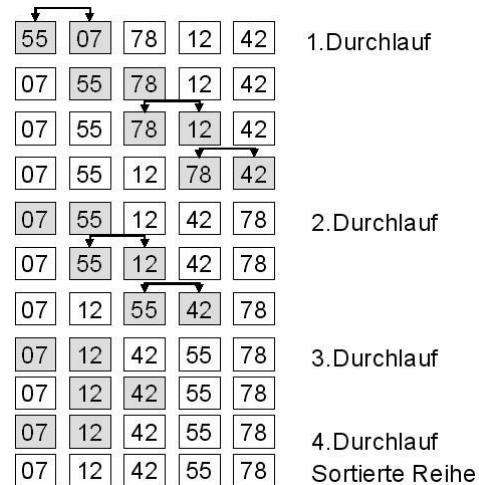
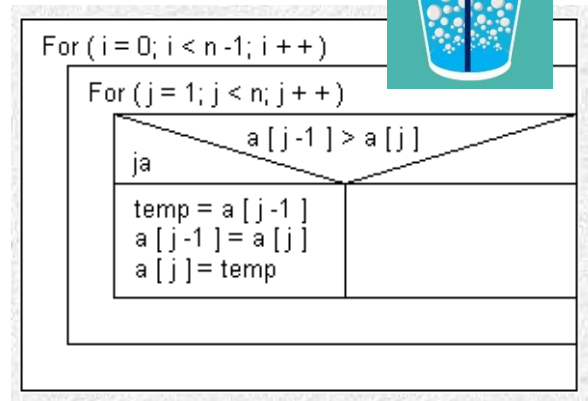
### Quelltext

```
public int[] sort(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        for (int j = 1; j < a.length; j++) {
            if (a[j - 1] > a[j]) {
                int temp = a[j - 1];
                a[j - 1] = a[j];
                a[j] = temp;
            }
        }
    }
    return a;
}
```

**Benachbarte Zahlen werden von vorne nach hinten verglichen und – bei Bedarf – getauscht – dies wird |Anzahl der Elemente im Array - 1| Mal wiederholt.**

In der Grafik rechts ist ein optimierter Algorithmus sichtbar - in jedem Durchlauf wird der zu betrachtende Teil um ein Element (von rechts nach links) verringert.

### Struktogramm



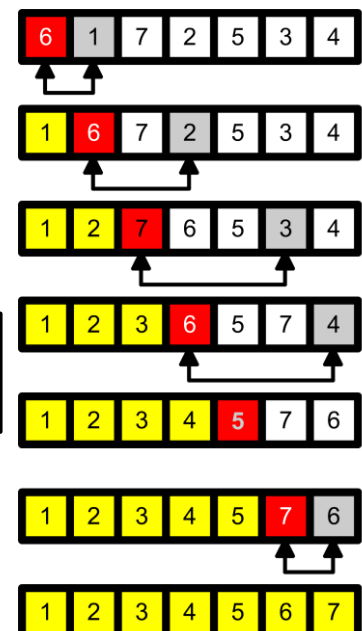
## Selectionsort (hier exemplarisch Minsort)

```
private void selectionSort() {
    for (int i=0; i<a.length-1; i++) {
        int minpos=minimumPosition(i);
        swap(minpos, i);
    }
}

private int minimumPosition(int from) {
    int minpos=from;
    for (int i=from+1; i<a.length; i++) if (a[i]<a[minpos]) minpos=i;
    return minpos;
}

private void swap(int i, int j) {
    int temp=a[i];
    a[i]=a[j];
    a[j]=temp;
}
```

Es gibt einen **sortierten** und einen **unsortierten** Teil im Array. Zu Beginn ist der **sortierte** Teil 0 Elemente groß und mit jedem Durchlauf vergrößert sich dieser um 1 weiteres Element, indem jeweils das kleinste Element des **unsortierten** Teils mit dem 1. Element des **unsortierten** Teils getauscht wird.



**Kleinstes Element in Restliste**  
**Sortierter Bereich**  
**Aktuelles Element**