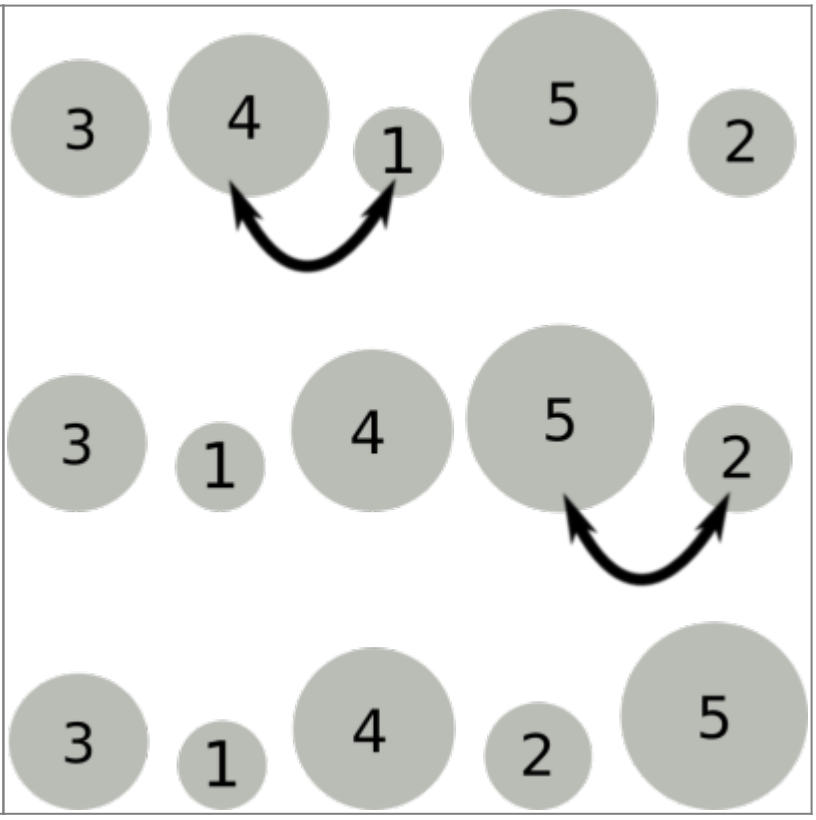
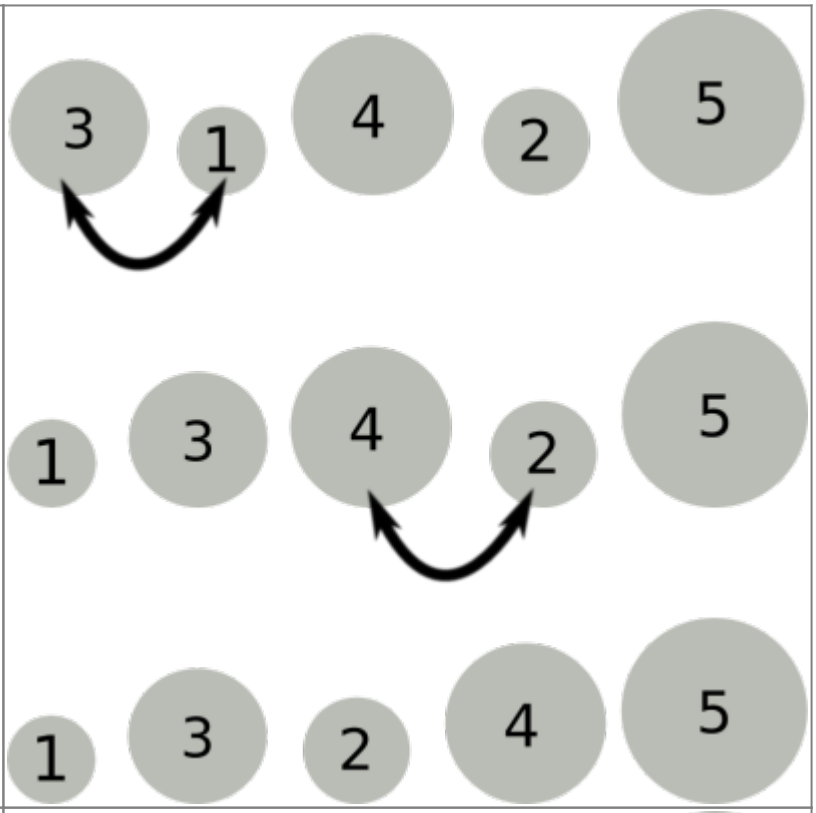
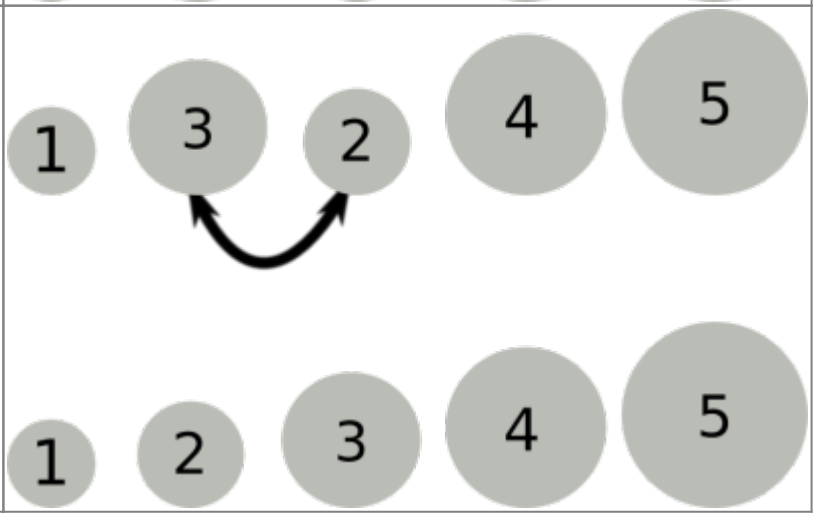
# Bubble Sort

Oje, Willi hat seine Mistkugeln auf offenem Feld aufgereiht. Ein heftiger Windstoss und die Kugeln rollen davon und liegen schon wieder in einem Durcheinander. Also nochmals sortieren. Willi ist noch ganz erschöpft von seiner ersten Sortier-Aktion. Vor allem das Suchen der "falschen" Paare war sehr anstrengend! Er musste dabei immer die ganze Kugelreihe überblicken.

Gut, ich machs nochmals, sagt er sich, aber diesmal ein bisschen systematischer. Er sucht die "falschen" Paare jetzt nicht mehr nach dem Zufallsprinzip, sondern geht von links nach rechts durch die Reihe. Dabei vergleicht er immer zwei benachbarte Kugeln. Sobald er zwei Kugeln mit falscher Reihenfolge gefunden hat, vertauscht er diese. Nachdem er die ganze Reihe so abgearbeitet hat, beginnt er wieder vorn vorne, d.h. von links. Wenn es schon mit Zufall geklappt hat, denkt sich Willi, dann muss es doch mit System erst recht klappen. Vielleicht sogar noch schneller als vorher? Tatsächlich findet er nach einigen Durchgängen kein "falsches" Paar mehr…

## Schritt für Schritt…

| | |
|---|---|
| **Erster Durchlauf:**<br>Es wird von links nach rechts jeweils die Mistkugel mit ihrem rechten Nachbarn verglichen.<br>Ist die rechte größer als die Linke, passiert nichts, ist die Linke größer als die rechte werden die beiden vertauscht.<br>→ 3 ist kleiner als vier, **O.K.**, weiter zum 2. Element<br>→ 4 ist größer als 1, **Tauschen**. Weiter zum nächsten Element, dem dritten<br>- nach dem Tausch ist das wieder die 4!<br>→ 4 ist kleiner als 5, **O.K.**, weiter zun nächsten Element<br>→ 5 ist größer als 2, **Tauschen**. Weitergehen zu ELement vier.<br>Durchlauf zu Ende. |  |

| | |
|---|---|
| **Zweiter Durchlauf:**<br>→ 3 > 1, **Tauschen**<br>→ 3 < 4, **O.K.**<br>→ 4 > 2, **Tauschen**<br>→ 4 < 5, **O.K.** |  |
| **Dritter Durchlauf:**<br>→ 1 < 3, **O.K.**<br>→ 3 > 2, **Tauschen**<br>→ 3 < 4, **O.K.**<br>→ 4 < 5, **O.K.** |  |

Bei einem vierten Durchlauf würde man keine Vertauschung mehr benötigen - die Mistkugeln sind sortiert.

Wenn man sich anstelle der Mistkugeln Luftblasen in einem Aquarium vorstellt und das Ganze um 90° gegen den Uhrzeigersinn dreht, sieht man in Gedanken, wie die Luftblasen im Wasser aufsteigen. Zuerst die grösste ganz nach oben, dann die zweitgrösste usw. Die Vorstellung von den steigenden Luftblasen hat diesem Sortierverfahren seinen Namen gegeben: **BubbleSort** (bubble (engl.) = Blase).

**Aufgabe:** Sortiere auf einem Blatt Papier mit dem BubbleSort Verfahren die folgende Mistkugelreihe. Fertige eine Tabelle an, in der jeder Durchlauf des Sortierverfahrens zu erkennen ist, markiere die Vertauschungen.[1)]

Beobachte die Postion der jeweils größten Kugeln, die noch nicht an ihrem endgültigen Platz sind - was fällt dir auf?

## Der Algorithmus

Wir sehen im Beispiel oben sofort, dass die Reihe nach dem dritten Durchgang sortiert ist und wir damit fertig sind. Wie aber kann ein Programm herausfinden, ob eine Reihe sortiert ist? Um zu entscheiden, ob die Reihe sortiert ist, kann unser Programm also einen weiteren Durchgang ausführen. Wenn in einem Durchgang kein "falsches" Paar mehr gefunden wird, sind wir fertig.

Diese Überlegungen führen uns auf folgenden Algorithmus:

| Schritt | Was ist zu tun? |
|---|---|
| (1) | Wähle die ersten beiden Elemente von links. |
| (2) | Ist beim gewählten Paar das linke Element grösser als das rechte? Wenn ja, vertausche die beiden. |
| (3) | Hat es rechts des soeben untersuchten Paares noch weitere Kugeln? Wenn ja, wähle das nächste Paar benachbarter Elemente und gehe zu (2). |
| (4) | Wir haben einen Durchgang beendet. Wurde in diesem Durchgang mindestens ein "falsches" Paar gefunden und vertauscht? Wenn ja, gehe zu (1), d.h. starte einen weiteren Durchgang. Wenn nein, sind wir fertig. |

## Anwendung des Algorithmus auf das Eingangsbeispiel

**Aufgabe:** In der Datei sortieren_bubble.pdf (ODS) ist der erste Durchgang gemäß des Algorithmus notiert. Fülle die Tabelle vollständig aus:

| Ausgangslage | | 3 | 4 | 1 | 5 | 2 |
|---|---|---|---|---|---|---|
| **1. Durchgang** | | | | | | |
| | Schritt & Was zu tun? | | | | | |
| (1) | Wähle das erste Paar: 3 und 4 | 3 | 4 | 1 | 5 | 2 |
| (2) | 3 ist nicht größer als 4, also nichts tun | 3 | 4 | 1 | 5 | 2 |
| (3) | Wähle nächstes Paar: 4 und 1 | 3 | 4 | 1 | 5 | 2 |
| (2) | 4 ist größer als 1, also vertauschen | 3 | 1 | 4 | 5 | 2 |
| (3) | Wähle nächstes Paar: 4 und 5 | 3 | 1 | 4 | 5 | 2 |
| (2) | 4 ist nicht größer als 5, also nichts tun | 3 | 1 | 4 | 5 | 2 |
| (3) | Wähle nächstes Paar: 5 und 2 | 3 | 1 | 4 | 5 | 2 |
| (2) | 5 ist größer als 2, also vertauschen | 3 | 1 | 4 | 2 | 5 |
| (3) | Es hat weiter rechts kein Paar mehr. | 3 | 1 | 4 | 2 | 5 |
| (4) | In diesem Durchgang 2 Vertauschungen gemacht, also noch ein Durchgang. | 3 | 1 | 4 | 2 | 5 |

- Schreibe Stichwortartig in die Felder, welchen Schritt des Algorithmus man ausführt und was genau dabei gemacht wird.
- Notiere die Zahlenreihe und markiere bei jedem Schritt, welche Zahlen dort betrachtet werden.

Warum sind vier Durchläufe nötig? Erkläre!

# Implementation des Algorithmus

**Aufgabe 1:** In der Datei bubblesort.zip findest du vier verschiendene Vorlagen für den Bubblesort-Algorithmus (easy - superhard).

- Die einfachste Variante ist dabei voll lauffähig, du kannst also auf jeden Fall damit beginnen, den Algorithmus mit dem Programm bubblesort_easy.php im Browser zu testen.
- Versuche dann, die Vorlage in bubblesort_medium.php lauffähig zu machen. Wenn das nicht klappt, bearbeite die Aufgaben in der Datei bubblesort_easy.php, und versuche anschließend bubblesort_medium.php zu bearbeiten.

**Aufgabe 2:** Was passiert, wenn man im Schritt 2 des BubbleSort-Algorithmus den Begriff "grösser als" durch

1. "kleiner als"
2. "grösser als oder gleich"
3. "kleiner als oder gleich"

ersetzt? Ändere das Programm jeweils entsprechend ab und überprüfe das Ergebnis deiner Überlegungen.

# Optimierungen

## Erste Optimierung

**Überprüfe mit Hilfe des Programms** `bubblesort_easy.php`, dass im ersten Durchgang die grösste Kugel immer sofort an ihren richtigen Platz (nämlich ganz nach rechts) transportiert wird, im zweiten Durchgang die zweitgrösste (nämlich an die zweite Position von rechts) usw.

Bei einem Array der Grösse n werden also in den ersten n-1 Durchgängen die n-1 grössten Elemente (das sind alle ausser das kleinste) an ihren endgültigen Platz gebracht. Wenn aber alle Elemente bis auf das kleinste am richtigen Platz sind, dann muss zwangsläufig das kleinste auch schon am richtigen Platz sein, nämlich ganz links. Eine andere Möglichkeit gibt es nicht!

Mit anderen Worten: Bei einem Array der Grösse n wird im n-ten Durchgang sicher nichts mehr vertauscht. Es sind also höchstens n-1 Durchgänge nötig. Man kann auch direkt eine feste Zahl von n-1 Durchgängen ausführen. Dadurch erspart man sich die Abfrage, ob in einem Durchgang etwas vertauscht wurde.

**Aufgabe:** Kopiere eine funktionierende Version des Bubbsort-Programms in eine Datei mit dem Namen `bubblesort_opt.php`. Verändere das Programm so, dass die erläuterte Optimierung umgesetzt ist.

Was ist ein möglicher Nachteil, wenn wir eine fixe Anzahl (nämlich n-1) Durchgänge ausführen?

### Zweite Optimierung

Nach dem ersten Durchgang steht das grösste Element ganz rechts. Nach dem zweiten Durchgang steht das zweitgrösste Element an zweiter Stelle von rechts. Nach i Durchgängen steht das i-t-grösste Element an der i-ten Stelle von rechts, und die i Elemente rechts sind sortiert. Es muss also nicht jedes Mal der ganze Array durchlaufen werden, sondern im i-ten Durchgang nur die ersten (n-i+1) Elemente.

**Aufgabe:** Streiche in der Tabelle, die du in der Aufgabe oben erstellt hast, alle Zeilen durch, welche gemäss dieser zweiten Optimierung unnötig sind.

Bearbeite anschließend die funktionierende Version des optimierten Bubbsort-Programms aus der vorigen Aufgabe (`bubblesort_opt.php`) so dass dort beide Optimierungen implementiert sind.

## Aufwandsmessungen

Um den Zeitaufwand bzw. die "Geschwindigkeit" eines Sortierverfahrens zu messen, sind verschiedene Möglichkeiten denkbar: Man könnte zum Beispiel die Laufzeit in Sekunden messen. Dies wäre allerdings abhängig von der verwendeten Maschine und möglicherweise auch von anderen, gleichzeitig laufenden Programmen.

Wir wollen den Aufwand unserer Verfahren messen, indem wir zwei Grössen bestimmen: Die **Anzahl Vergleiche** und die **Anzahl Vertauschungen** von Array-Elementen. Sämtliche hier behandelten Verfahren basieren nämlich ausschliesslich auf diesen beiden Operationen. Den übrigen Aufwand, z.B. für die Ablaufsteuerung von Schleifen, ignorieren wir grosszügig.

Verwende als Basis für die folgenden Aufgaben bitte die Datei bubblesort_zufallsarray.php.zip. Dabei handelt es sich um ein nicht optimiertes Bubblesortverfahren, welches zunächst ein Zufallsarray erzeugt, dessen Länge im Formular angegeben werden kann. Außerdem verfügt das Programm über zwei Zähler, die folgendes tun:

- `$vergleiche_gesamt` → Zählt die bis zur Ausgabe des sortierten Arrays nötigen Vergleichoperationen.
- `$vertauschungen_gesamt` → Zählt die bis zur Ausgabe des sortierten Arrays nötigen Vertauschungsoperationen.

**Aufgabe:** Teste das Programm mit verschieden langen Arrays. Recherchiere im Internet (PHP Doku), was genau jede Anweisung der folgenden im Programm enthaltenen Schleife macht. Füge entsprechende Kommentare ein.

```
// Erzeugt ein Array der Länge $laenge mit
// Zufallszahlen zwischen 1 und 1000
for($i=0;$i<$laenge;$i++) {
  $zufallszahl = rand(1,1000);
  $das_array[] = $zufallszahl;
}
```

**Aufgabe:** Fülle die Tabelle in der Datei sortieren_aufwand_bubble.ods aus. Kopiere und bearbeite dazu das Programm bubblesort_zufallsarray.php.zip so, dass es die Optimierungen erhält und

zusätzlich die Aufwandszählung vornimmt.

Kannst du einen Zusammenhang zwischen der Arraygröße **n** und den erforderlichen Operationen erkennen?

Weiter zu SelectionSort.

1)

Für die Tabelle kann man auch nur die Zahlen nehmen und die Mistkugeln weglassen…

From:
https://www.info-bw.de/ -

Permanent link:
**https://www.info-bw.de/faecher:informatik:oberstufe:algorithmen:sortieren:bubblesort?rev=1582211134**

Last update: **20.02.2020 15:05**