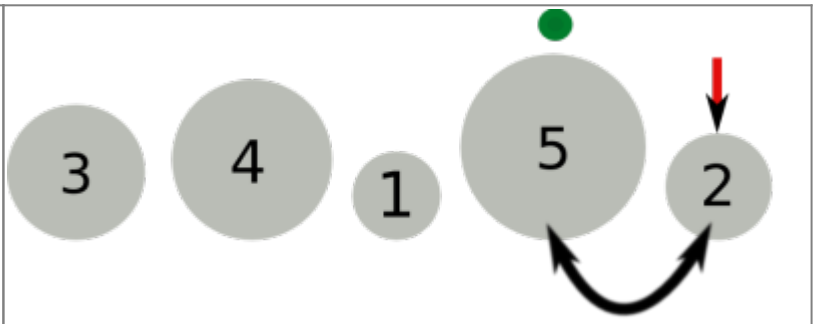
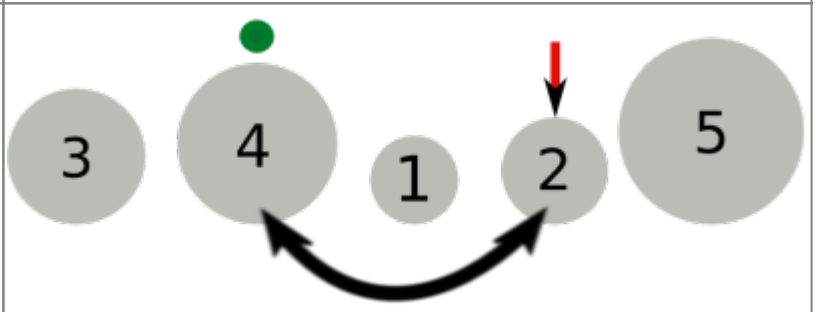
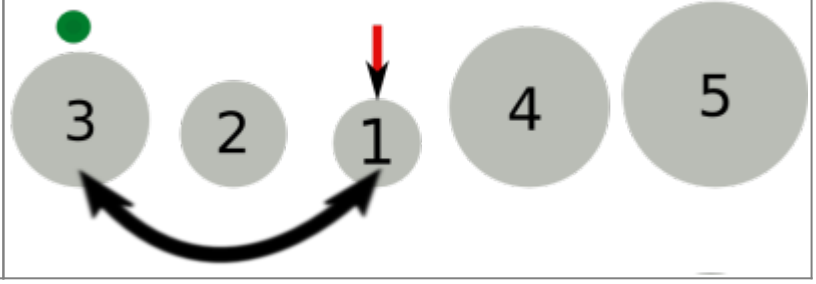
Selectionsort

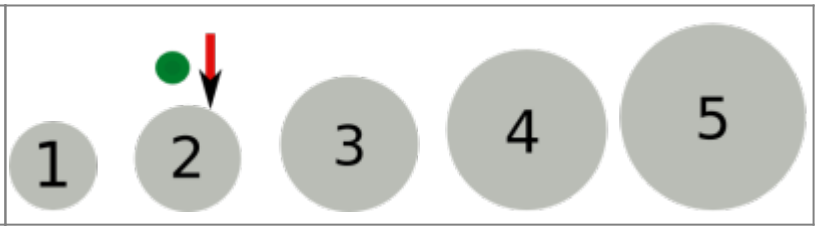
in furchtbares Gewitter ist durch das Land gezogen und hat Willis fein säuberlich sortierte Mistkugelsammlung total durcheinander gebracht. Diesmal ist Willi aber guten Mutes. Er weiss ja nun, wie er die Kugeln wieder in die richtige Reihenfolge bringen kann. Schon will er ans Werk gehen. Aber halt! Die Kugeln sind vom Regen noch ganz schmutzig. Muss er wirklich jede Kugel so oft anfassen wie beim ersten Mal? Nicht dass er sich vor seinen Mistkugeln ekeln würde, aber die Kugeln könnten dabei Schaden nehmen.

Er setzt sich hin und überlegt. Am Schluss muss doch die grösste Kugel ganz rechts liegen, das ist klar. Warum suche ich nicht einfach die grösste Kugel und lege sie ganz nach rechts?

Gedacht, getan. Das ist aber gar nicht so einfach, denn die beiden grössten Kugeln sind fast gleich gross. Willi nimmt sein Messband und misst die Umfänge der beiden Kandidaten. Das Resultat ist eindeutig. Er bringt also die grösste Kugel an die Position ganz rechts, indem er sie mit der Kugel vertauscht, welche vorher dort war. Die zweitgrösste Kugel bekommt den Platz direkt links neben der grössten, ebenfalls durch Vertauschen. So fährt Willi nun fort: Er bestimmt die drittgrösste Kugel und bringt sie an die dritte Position von rechts und so weiter. Es dauert gar nicht lange, bis er fertig ist. Zufrieden schaut er sein Werk an und überlegt sich, wieviel Mal er jede Kugel anfassen musste...

Schritt für Schritt

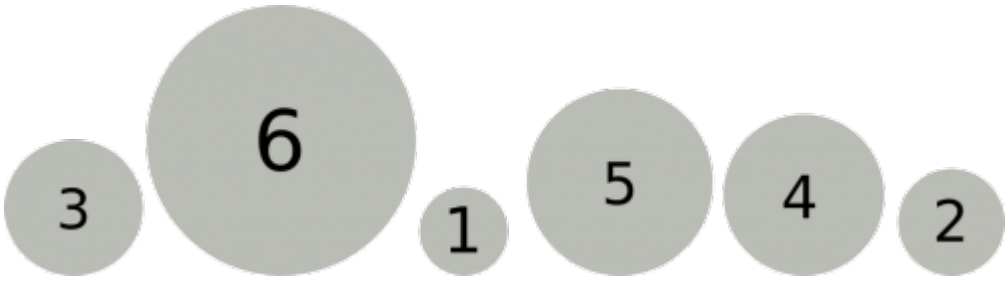
Es wird das gesamte Array nach dem größten Element durchsucht (5) Dieses wird mit dem Element ganz rechts vertauscht (2)	
Das Array wird bis auf das Element ganz rechts nach dem größten Element durchsucht (4) Dieses wird mit dem vorletzten Element rechts vertauscht (2)	
Das Array wird bis zum drittletzten Element nach dem größten Element durchsucht (3) Dieses wird mit dem drittletzten Element rechts vertauscht (1)	

Das Array wird bis zum viertletzten Element nach dem größten Element durchsucht (2) Dieses wird mit dem viertletzten Element rechts vertauscht (2) Dasselbe Element? Fertig!	

In diesem Beispiel sind wir mit drei Vertauschungen ausgekommen. Die letzten beiden Kugeln waren schon am richtigen Ort. War das Zufall? Bei der vorletzten schon, aber bei der letzten nicht. Wenn nämlich alle Kugeln ausser die kleinste an ihren richtigen Platz verschoben wurden, dann muss die kleinste zwangsläufig auch schon an ihrem richtigen Platz stehen, nämlich ganz links.

Dieses Verfahren trägt den Namen **SelectionSort**, weil in jedem Durchgang das nächstgrösste Element des Arrays ausgewählt und an seinen endgültigen Platz gesetzt wird (selection (engl.) = Auswahl).

Aufgabe: Sortiere auf einem Blatt Papier mit dem SelectionSort Verfahren die folgende Mistkugelreihe.



Der Algorithmus

Den Algorithmus für einen Array mit n Elementen könnte man so formulieren: Algorithmus

Schritt	**Was tun**
(1)	Suche das grösste Element des ganzen Arrays und vertausche es mit dem Element ganz rechts.
(2)	Suche das zweitgrösste Element des ganzen Arrays und vertausche es mit dem zweiten Element von rechts.
... und so weiter...	
(n-1)	Suche das zweitkleinste Element und vertausche es mit dem zweiten Element von links.

Wir wollen uns aber mit dieser saloppen "und so weiter"-Formulierung nicht zufrieden geben. Präziser ist die folgende Darstellung:

Schritt	**Was tun**
(1)	Setze einen Zähler i auf 1
(2)	Suche das i-t-grösste Element (*)
(3)	Vertausche es mit dem i-ten Element von rechts
(4)	Erhöhe die den Zähler i um 1. Wenn i kleiner als die Arraylänge ist, gehe zu (2), sonst sind wir fertig

(*) mit dem "i-t-grössten Element" ist folgendes gemeint:

i	"i-t-grösstes Element"
1	grösstes
2	zweitgrösstes
3	drittgrösstes
...	

Bevor wir diesen Algorithmus programmieren, wollen wir noch ein wichtiges Detail klären: Im Schritt (2) müssen wir das i-t-grösste Element des Arrays bestimmen. Dieses Problem allgemein zu lösen ist gar nicht so einfach. Glücklicherweise ist es aber in unserem Fall nicht schwierig.

Schauen wir uns nochmals den obigen Ablauf an: Zu Beginn des zweiten Durchgangs ist das grösste Element ganz rechts, zu Beginn des dritten Durchgangs ist zusätzlich das zweitgrösste an zweiter Stelle von rechts. Allgemein: Zu Beginn des Durchgangs i befinden sich die (i-1) grössten Elemente auf der rechten Seite des Arrays, und zwar bereits in sortierter Reihenfolge und damit an ihrer endgültigen Position. An diesem Teil des Arrays müssen wir also gar nichts mehr ändern.

Wie finden wir nun das i-t-grösste Element? Ganz einfach: Es ist das grösste Element des linken, noch unsortierten Teils unseres Arrays.

Damit erhalten wir die **letzte Version des Algorithmus**, die wir dann auch programmieren können:

Schritt	Was tun
(1)	Setze den Zähler i auf 1.
(2)	Suche das grösste Element der (n-i+1) Elemente von links.
(3)	Vertausche es mit dem i-ten Element von rechts.
(4)	Erhöhe den Zähler i um 1. Wenn i kleiner als die Arraylänge ist, gehe zu (2), sonst sind wir fertig.

Implementation des Algorithmus

Aufgabe 1: In der Datei selectionsort.zip findest du drei verschiendene Vorlagen für den Selectionsort-Algorithmus (easy - hard) und eine Version zur Aufwandsmessung.

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

Aufwandsmessungen

Verwende als Basis für die folgenden Aufgaben bitte die Datei `selectionsort_zufallsarray.php` aus dem Archiv selectionsort.zip . Dabei handelt es sich um ein Selectionsortverfahren, 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. Fülle die Tabelle in der Datei sortieren_aufwand_selection.ods aus.

- Kannst du einen Zusammenhang zwischen der Arraygröße **n** und den erforderlichen Operationen erkennen?
- Vergleiche den Aufwand von Selectionsort mit dem Bubblesortverfahren.