# Exercise 07: Sorting

Sorting is essentially based on two operations: the comparison of two objects and, depending on the sorting method, the swapping of two objects or the removal and reinsertion of objects. These two operations are repeated in an appropriate manner until the sorting order is established. In the lecture, we learned about insertion sort, selection sort, merge sort and quicksort. In the exercise today, we want to implement bubble sort, where individual elements "bubble up" until they are in the right place. The following animation illustrates the process:
https://commons.wikimedia.org/w/index.php?curid=14953478

### Task 1: Swap: swapping elements

- Create a `Sorting` class
- Write a static generic `swap` method, which receives an array and two indexes and swaps the elements at those specific indexes.
- Implement a test case (e.g. in a `SortingTests` class) that verifies the function of the `swap` method. For this purpose, use the `@Test` annotation from JUnit 5.

### Task 2: Sorting: bubble sort

Implement bubble sort, a sorting method that takes an array and sorts it in-place, i.e. modifies the passed array.

- Make sure you understand the algorithm by manually sorting the number sequence 5, 2, 3, 6 using this method, see https://de.wikipedia.org/wiki/Bubblesort
- Implement bubble sort as a generic static method that takes an array.
- Set the bounds so that only arrays of classes that implement Comparable accordingly can be passed.
- Modify the Student class so that objects of this type are sorted by matriculation number in ascending order.
- Write a test case that sorts four students. Please note: for example, you can use the `Arrays.toString(...)` method to easily output arrays to the console, and `Assertions.assertArrayEquals(...)` to test two arrays for identical content (equality).

### Task 3: Own sorting orders

It is often necessary to sort arrays (or other data structures) according to different criteria, e.g. in an e-mail inbox: by sender, date, subject, size, etc.

- Now implement bubble sort again, only this time without bounds; also add a Comparator to the signature and adjust your implementation accordingly.
- Implement a Comparator that sorts students by name in ascending alphabetical order. Test it with a suitable test case.
- Write a Comparator that sorts students initially by name in ascending alphabetical order, and in case of identical names (equality), by matriculation number in ascending order. Test it with a suitable test case.