# Tutorial 1

## Exercise 1

- Suppose we have measured these runtimes:

| Input size | Algorithm 1 | Algorithm 2 |
|---|---|---|
| 100 | 5 | 9 |
| 200 | 21 | 13 |
| 400 | 83 | 16 |
| 800 | 417 | 20 |
| 1600 | 1666 | 24 |
| 3200 | 8330 | 27 |

- Using the empirical method described in the lecture, try to determine the complexity classes of the algorithms.

## Exercise 2

One important point in this week's lecture is how sorting the contents of an array makes finding values more efficient. Let us have a look at a simple example for this.

- First, come up with an algorithm to solve the following problem:
  - Given: an array (or vector, or ArrayList) of nonnegative (i.e. >=0) integers.
  - Wanted: The smallest nonnegative number which is missing.
  - For example, if the input is 2, 7, 4, 0, 3, 2, 5, 1, 0, 3 the result should be 6.
- Determine the complexity of your solution based on its loop structure.
- Measure its runtime on inputs of some suitable sizes (keep the doubling hypothesis in mind, and aim for ones which take at least 1 second) and see if they point at the same complexity. For creating the input, we are using the `createInput` function.
- Write another version which assumes that the input is sorted, and repeat the analysis. For the measurements, set the `sort` variable in `main()` to `true`.

  Note that we only measure the time taken by the actual search, not the sorting (you could change this by moving the lines which do the sorting between the time checks), so the program is going to run longer than the measurements would suggest.