Prof. Dr. Jasmin Blanchette
Dr. Martin Desharnais-Schäfer
Dr. Michael Kirsten

Ludwig-Maximilians-Universität München
Institut für Informatik
Discussion on 16.12.2026
Homework due on 23.12.2026 at 14:15

Exercise Sheet 8 in

Scientific and Technical English

WiSe 2026/27

The exercise sheets consist of in-class exercises and homework. The in-class exercises take place during the second half of the lecture time slots. The homework, which is optional and ungraded, can be submitted via the "Homework" section in Moodle. The homework is subject to peer review.

Unless indicated otherwise, generative artificial intelligence assistants such as ChatGPT may be used, as long as you acknowledge how you use them as specified by the Institute's policy on plagiarism.[1] However, you may not use such tools to generate peer reviews for you. In addition, we strongly recommend that you do not use them to generate entire solutions, since that would defeat the purpose of the exercises.

In-class exercise 8-1 ***Telephone Game*** The following two pages present the Java code for the bubble sort and selection sort algorithms.

a) Form a pair with another student.

b) Assign one of the algorithms to you and the other one to your teammate.

c) Read your algorithm and only that one.

d) Write a description of the inner workings of your algorithm. You can assume that the reader knows about the `swap` function.

e) Exchange the descriptions once you and your teammate have finished writing.

f) Write a Java program that exactly follows your teammate's description.

g) Once you and your teammate have finished programming, exchange your programs and compare them with those given on the following pages.

h) Discuss how effective the descriptions were at conveying the algorithms.

[1] `https://www.medien.ifi.lmu.de/lehre/Plagiate-IfI.pdf`

```
static public void bubbleSort(int[] a) {
    final int n = a.length;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1]) {
                swap(a, j, j + 1);
            }
        }
    }
}
```

```
static public void selectionSort(int[] a) {
    final int n = a.length;
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++) {
            if (a[j] < a[min]) {
                min = j;
            }
        }
        if (min > i) {
            swap(a, i, min);
        }
    }
}
```

Homework 8-2 ***Mathematical Prose*** Improve the presentation of the following excerpts. Follow the guidelines on mathematical prose and notation style from the lecture.

a) Consider the language $L = \{abc | a \in \Sigma\}$ over the alphabet $\Sigma = \{b, c\}$.

b) Consider an undirected graph $G = (V, E)$. If $\forall$ elements $x, y \in C$, there is a $z \in E$ so that $z_1 = x$ and $z_2 = y \Longrightarrow C$ is a clique.

Homework 8-3 ***Mathematical Proof*** The following is an excerpt from introductory material on algebraic structures intended for bachelor computer science students. Improve its presentation to make it easier to follow.

> **Lemma 3.1.** *In every group* $(G, \circ)$ *it holds that:* $\forall a \in G. \forall b \in G. (a \circ b)^{-1} = b^{-1} \circ a^{-1}$.
>
> *Proof.* $a, b \in G$. Then
>
> $$(b^{-1} \circ a^{-1}) \circ (a \circ b) = b^{-1} \circ (a^{-1} \circ a) \circ b = b^{-1} \circ e \circ b = b^{-1} \circ b = e,$$
>
> because of the laws of associativity and the inverse and neutral element, respectively. So $b^{-1} \circ a^{-1}$ is the left inverse of $a \circ b$, therefore $(a \circ b)^{-1} = b^{-1} \circ a^{-1}$. $\square$

Homework 8-4 ***Numbers*** Improve the formatting of numbers in the following paragraphs using the rules from the lecture:

> More generally, `./sort1 p q` reads $q \times p$ input bits, applies this paper's algorithm, and prints a test program that includes the resulting code. The program packs each matrix row into 4 `unsigned long long` variables, so it is limited to $p \in \{0, 1, 2, \ldots, 256\}$, but it allows any $q \in \{1, 2, 3, \ldots\}$ that fits into memory. The test program prints the algebraic complexity of the code, i.e., the number of xors. The test program also checks that the code computes the desired outputs; if this check fails, the program prints `999999999`.
>
> Table 3.1 [not shown here] shows the average algebraic complexity, divided by pq, of this code for 10000 random $q \times p$ matrices obtained from `/dev/urandom`, the Linux cryptographic random number generator. Table 3.2 shows the standard deviation of the algebraic complexity. For example, for $(p, q) = (64, 128)$, the two tables have entries 0,1922 and 0,0011 respectively; this algorithm evaluated 10000 random sixty-four-bit-to-one-hundred-twenty-eight-bit linear maps using approximately $0,1922 \cdot 128 \cdot 64 \approx 1,575$ xors on average, with standard deviation approximately $0,0011 \cdot 128 \cdot 64 \approx$ nine.