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

Ludwig-Maximilians-Universität München
Institut für Informatik
Discussion on 22.10.2025
Homework due on 29.10.2025 at 16:00

Possible solution for Exercise Sheet 2 in

Scientific and Technical English for Computer Scientists

The exercise sheets consist of in-class exercises and homework. The in-class exercises take place in 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 this would defeat the purpose of the exercises.

In-class exercise 2-1 ***Curse of Knowledge*** Start by reading the following definition adapted from Jonathan Katz and Yehuda Lindell, *Introduction to Modern Cryptography*, Second Edition, CRC Press, 2014:

> A *message authentication code* (or *MAC*) consists of three probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrfy})$ such that:
>
> - The *key-generation algorithm* Gen takes as input the security parameter 1^n and outputs a key k with $|k| \geq n$.
> - The *tag-generation algorithm* Mac takes as input a key k and a message $m \in \{0,1\}^*$, and outputs a tag t. Since this algorithm may be randomized, we write this as $t \leftarrow \mathsf{Mac}_k(m)$.
> - The deterministic *verification algorithm* Vrfy takes as input a key k, a message m, and a tag t. It outputs a bit b, with $b = 1$ meaning *valid* and $b = 0$ meaning *invalid*. We write this as $b := \mathsf{Vrfy}_k(m, t)$.
>
> It is required that for every n, every key k output by $\mathsf{Gen}(1^n)$, and every $m \in \{0,1\}^*$, it holds that $\mathsf{Vrfy}_k(m, \mathsf{Mac}_k(m)) = 1$.
>
> If there is a function ℓ such that for every k output by $\mathsf{Gen}(1^n)$, algorithm Mac_k is only defined for messages $m \in \{0,1\}^{\ell(n)}$, then we call the scheme a *fixed-length MAC for messages of length* $\ell(n)$.

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

Identify the background knowledge required to understand this definition for a novice to the field of cryptography.

POSSIBLE SOLUTION:

The following suggestions are in order of appearance.

- What is a probabilistic algorithm?
- What is a polynomial-time algorithm?
- What is a key?
- What is a security parameter?
- What is a tag?
- What does the notation 1^n mean?
- What does the notation $|k|$ mean?
- What does the notation $\{0,1\}^*$ mean?
- What is a randomized algorithm?
- Why is the input k not inside the parentheses in $\mathsf{Mac}_k(m)$?
- What is a deterministic algorithm?
- What does it mean for an algorithm to be defined?
- What does the notation $\{0,1\}^{\ell(n)}$ mean?

In-class exercise 2-2 ***From Title to Ideas*** Together with your colleagues, you invented, implemented, and evaluated a new sorting algorithm. You now want to write and submit a 15-page paper to a renowned international conference in the field of algorithms.

a) Choose a title for your paper. It may be catchy, informative, or double-barreled.

b) Write your paper's table of contents, and allocate a page budget to every section and subsection.

c) In each section and subsection, write down at least two bullet points standing in for the future content.

POSSIBLE SOLUTION:

A Linear-Time Tasseography-Based Sorting Algorithm

1. Introduction (1.5 pages)

- From Wikipedia: Tasseography (also known as tasseomancy, tassology, or tasseology) is a divination or fortune-telling method that interprets patterns in tea leaves, coffee grounds, or wine sediments
- From Wikipedia: While tea leaf reading originated in China, likely soon after the creation of tea, various regions practice it with slight variations
- There are many sorting algorithms in the literature, but none based on tasseography
- We present the first such algorithm
- We prove it correct, study its complexity, and evaluate it

2. Background (2 pages)

2.1. Tasseography (0.5 pages of 2)

- Instructions:
  a) Pour the tea (do not use a filter)
  b) Drink the tea (or pour it away if the divination takes place after 17:00; otherwise, you will not be able to sleep at night)
  c) Shake the cup
  d) Look at the pattern formed by the tea leaves, and interpret it

2.2. Sorting (0.75 pages of 2)

- partial and total orderings
- the sorting problem:

  Input A sequence of n numbers

  Output A permutation $x_1, x_2, \ldots, x_n$ of the input sequence such that $x_1 < x_2 < \cdots < x_n$

- stability of sorting algorithms

2.3. Complexity (0.75 pages of 2)

- big O notation

- time complexity
- space complexity

3. Tasseo-Sort (8 pages)

3.1. Definition (2 pages of 8)

- divination procedure
- Tasseo-Sort procedure

3.2. Correctness (2 pages of 8)

- Tasseo-Sort produces a sorted permutation + proof
- Tasseo-Sort is stable + proof

3.3. Complexity (4 pages of 8)

- Tasseo-Sort runs in $O(n)$ time + proof
- Tasseo-Sort runs in O(1) space + proof

4. Evaluation (2 pages)

- We implemented Tasseo-Sort in Rust
- We generated 1000 sequences of length $10^3, 10^4, \ldots, 10^9$ (7000 sequences in total)
- The elements are pseudorandomly generated 64-bit integers
- We compared the following algorithms: Tasseo-Sort, heapsort, quicksort, mergesort
- Tasseo-Sort is substantially faster than the other algorithms

5. Related Work (1 page)

- Coffee-grounds reading is an alternative to tea-leaf reading
- Prayer-Sort is an alternative to Tasseo-Sort
- Meyer et al. produced a comparative evaluation of sorting algorithms in 2021

6. Conclusion (0.5 pages)

- Tasseo-Sort is based on tea-leaf reading and runs in linear time
- Tasseo-Sort is substantially faster than heapsort, quicksort, and mergesort
- Future work includes a comparison of tea-leaf and coffee-grounds reading in the context of sorting algorithms