University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
G. Schmid

Algorithms and Data Structures – Conditional Course Winter Term 2026/27

Sample Solution Exercise Sheet 1

Due: Wednesday, 28 October 2026, 12:00

Exercise 1: Registration *(5 Points)*

Register for Zulip using the invitation link on the course website once it is available. We use Zulip as a forum for questions about the lectures, exercises, and course organization. Prefer asking questions in the public course channel so that others can benefit from the answers. Follow the current course website for submission instructions.

Exercise 2: Sorting and Measuring Running Times *(5 Points)*

You have already encountered several sorting algorithms in the lectures. In this exercise, you will implement such algorithms and measure and compare their running times. Implement both *Selection Sort* and *Merge Sort*. Use the template `Sort.py`.

- Measure the running time of both implementations. Use the supplied code to generate random arrays of size n. Plot running time against n, using lengths from 100 to 10000, for example in increments of 100.[1] Briefly discuss the results in `experience.txt`; see Exercise 4.

Sample Solution

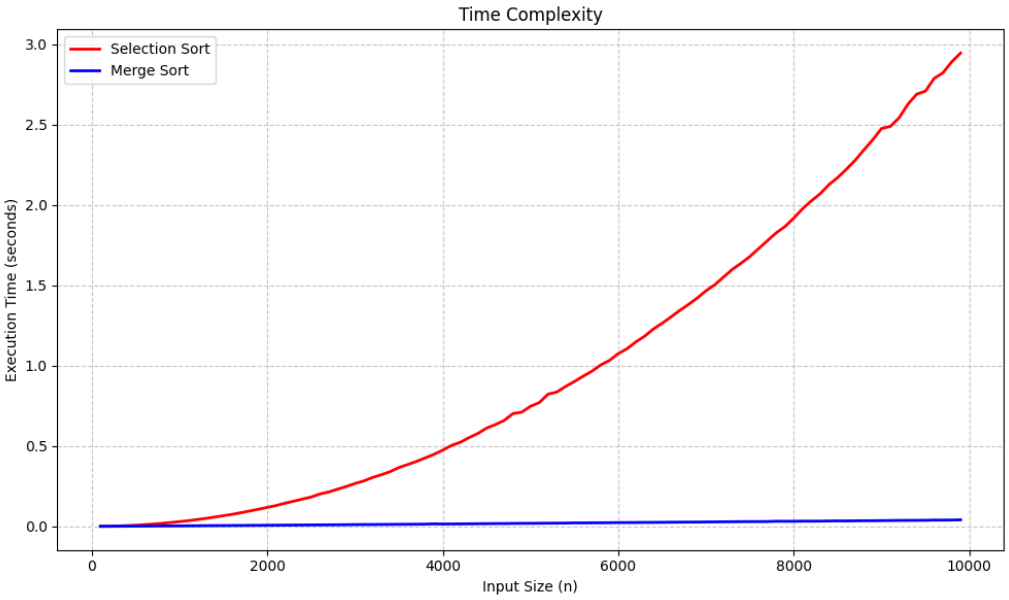
The file `Sort.py` accompanying this solution contains implementations of both algorithms.

Selection Sort repeatedly looks through the remaining part of the array to find its smallest element. It puts that element in the next free position at the front. This means that it looks at many elements again and again. When the array gets larger, there are both more positions to fill and more elements to search through. As in the lecture, doubling the array size makes the sorting time roughly four times as large.

Merge Sort splits the array into two halves, sorts each half in the same way, and then merges the sorted halves. To merge them, we only need to compare the next unused element from each half and take the smaller one. We do not have to search through the whole remaining array each time. This is why Merge Sort handles larger arrays much faster in our measurements.

The plot below shows example measurements from the reference implementation. Your exact times will depend on your computer and the generated arrays, but you should see the same general difference: the curve for Selection Sort bends upwards much more strongly than the curve for Merge Sort. Briefly describe what you see in your own plot in `experience.txt`.

[1] The comparison is clearest when both algorithms appear in the same plot, with n on the horizontal axis and running time on the vertical axis.



In this example, Selection Sort takes about 3 seconds for 10,000 elements, while Merge Sort stays close to the horizontal axis on this scale. The red curve gets steeper as the input grows: sorting twice as many elements takes roughly four times as long. Merge Sort's running time increases much more slowly.

Exercise 3: Puzzle: The Corrupt Police Officers *(5 Points)*

Solve the following puzzle and write your solution in a PDF, as you should for all subsequent theoretical exercises. We recommend LaTeX, but other formats are acceptable if the result is clear and legible.
Imagine that you are investigating a police station with 64 officers. Some are corrupt, but a strict majority (at least 33) are honest. Every officer knows exactly which officers are corrupt.
Your task: Determine which of the 64 officers are honest and which are corrupt. You may ask only one *type* of question: "Officer A, is your colleague B corrupt?" You may question any officer as often as you wish, but each question must concern exactly two officers in this manner.
Honest officers always tell the truth. Corrupt officers answer arbitrarily: they may deliberately lie or tell the truth to protect themselves.
Describe a strategy that is guaranteed to identify all corrupt officers. How many questions does your strategy need at most?

Sample Solution

The idea is to find just one officer who we can be sure is honest. We can then ask that officer about everyone else.

Imagine a chain of officers where each officer says that the next one is honest:

$$p_1 \longrightarrow p_2 \longrightarrow p_3 \longrightarrow \cdots \longrightarrow p_L.$$

An arrow here means "says the next officer is honest". If one officer in the chain is honest, the next one must be honest too, and so must everyone after that. So any corrupt officers must form a *prefix*: they can only appear together at the beginning of the chain, before the honest officers.

There are at most 31 corrupt officers. A chain of 33 must therefore contain an honest officer, so its last officer is honest.

Start with any officer. Ask the last officer in the chain about someone new. If the answer is "honest", add that person to the end and continue.

But what if the last officer A says that the next person B is corrupt? At least one of the two must be corrupt: either A is honest and tells the truth about B, or A is corrupt. We can therefore set both aside while we look for an honest officer. Removing them cannot destroy the honest majority, because we remove at least one corrupt officer and at most one honest officer.

After setting A and B aside, continue from the previous end of the chain. If the chain is now empty, start again with someone who has not yet been considered. We still have an honest majority among the officers we have not set aside. So we can stop as soon as the chain contains **more than half of those remaining officers**. Its last officer must then be honest, by the same reasoning as before.

Each question brings in someone new. If nobody new is left, all remaining officers are in the chain, so it must be long enough to stop.

Finally, ask the honest officer about all the other officers, **including those we set aside**. Their answers tell us exactly who is corrupt.

How many questions do we use? During the search, each question concerns a new person, and the first person was chosen without a question. So finding an honest officer takes at most 63 questions. Asking that officer about the other 63 people takes at most another 63. Altogether, our strategy uses at most 126 questions. We do not claim that this bound is optimal.

Exercise 4: Submission *(5 Points)*

Prepare your code from Exercise 2, including plots, together with the PDF containing your solution to Exercise 3. Submit the files according to the instructions on the current course website.
Also include a text file named `experience.txt`. In a few sentences, describe your experience with this sheet and the related lectures. In particular, how much time did you spend? Where did you encounter difficulties, and how much time did they cost you?