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

Algorithms and Datastructures
Winter Term 2024
Sample Solution Exercise Sheet 0

Due: Wednesday, October 30th, 12 pm

Exercise 1: Registration *(5 Points)*

Register for Zulip using the invitation-link given on the website. Note that we use Zulip as **forum** for questions regarding the *lecture* and the *exercises* as well as the platform to hand in submissions to tutors.

Exercise 2: Comparing the efficency of an algorithm *(5 Points)*

Suppose that we have two different algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ for solving our problem. For problem inputs of size n, $\mathcal{A}_1$ takes time:

$$t_1(n) = 10000 + 20 \cdot (\log(n))^2$$

The second algorithm $\mathcal{A}_2$ takes time:

$$t_2(n) = 2 \cdot \sqrt{n}$$

Which algorithm is faster for inputs of size $n = 1024 = 2^{10}$? For which n do the two algorithms run at the same speed (use a software like wolframalpha)? What happens if n becomes larger? Which algorithm would you choose to implement?

Sample Solution

Which algorithm is faster for $n = 1024$?

For $n = 1024$, we can calculate the running times of both algorithms.

$$t_1(1024) = 10000 + 20 \cdot (\log_2(1024))^2 = 10000 + 20 \cdot (10)^2 = 10000 + 2000 = 12000$$

$$t_2(1024) = 2 \cdot \sqrt{1024} = 2 \cdot 32 = 64$$

Thus, for $n = 1024$, we have:

$$t_1(1024) = 12000, \quad t_2(1024) = 64$$

So, $\mathcal{A}_2$ is much faster than $\mathcal{A}_1$ for $n = 1024$.

When do the two algorithms run at the same speed?

We want to find n such that $t_1(n) = t_2(n)$:

$$10000 + 20 \cdot (\log(n))^2 = 2 \cdot \sqrt{n}$$

This is a transcendental equation, so we solve it numerically. Using a trial-and-error approach or a numerical solver, we find that the two algorithms have approximately the same running time when $n \approx 1.536 \cdot 10^8$.

What happens when n becomes larger?

As n becomes larger, we notice the following trends:
- For large n, the term $20 \cdot (\log(n))^2$ grows logarithmically, while $2 \cdot \sqrt{n}$ grows polynomially.
- This means that as n increases, $\mathcal{A}_2$ becomes slower than $\mathcal{A}_1$. In other words, $\mathcal{A}_2$ is faster for small n, but $\mathcal{A}_1$ will eventually become faster for very large n.

Which algorithm should you implement?

- If the input size n is relatively small (up to a few thousand), $\mathcal{A}_2$ is preferable because it runs significantly faster for smaller values of n.
- However, if n can grow very large (e.g., $n > 10^6$), $\mathcal{A}_1$ will eventually become faster, making it the better choice for very large input sizes.

Exercise 3: Small Riddles *(10 Points)*

These small riddles can illustrate the types of thought processes one has to think about when designing algorithms. Often finding the best algorithm requires some nice idea or some out of the box thinking. The following two riddles are trying to illustrate what kind of thought processes or argumentations we are looking for in this course.

- Suppose that you have 27 metal balls that all look perfectly identical, but you know that exactly one of them is slightly heavier. The only thing at your disposal is a classical weighing scale. Everytime you weigh some balls against some other balls, either the scale tips left, right or stays balanced. The goal is to find the heaviest ball with the minimum number of uses of the scale. Can you find a way to figure out the heavier ball when weighing just 3 times?

- Suppose a standard 8×8 chessboard (or checkerboard) has two diagonally opposite corners removed, leaving 62 squares. Is it possible to place 31 dominoes of size 2×1 so as to cover all of these squares? Compare Figure 1 for an illustration.

Sample Solution

Weighing Riddle

The key insight here is to use a divide-and-conquer approach. You divide the set of 27 balls into 3 groups of 9 balls each. Weigh two of these groups against each other:

- If one group is heavier, the heavier ball must be in that group.
- If they balance, the heavier ball must be in the third group.

Now, you're left with 9 balls. Divide these into 3 groups of 3 balls each and repeat the same weighing procedure:

- If one group is heavier, the heavier ball is in that group.
- If they balance, the heavier ball is in the third group.

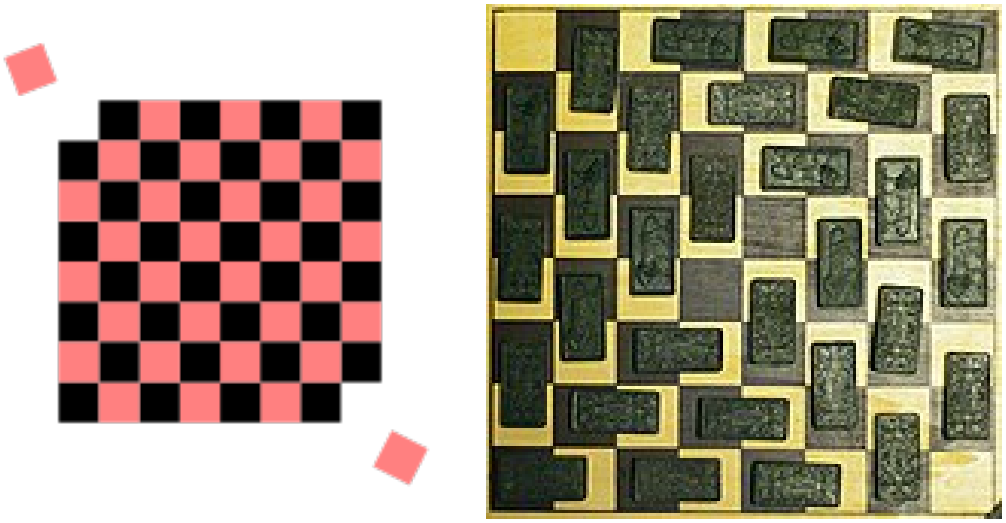

Figure 1: The left side shows a chessboard without the two diagonally opposite corners. The right side shows an (unsuccessful) attempt at a solution.

Finally, you have 3 balls left. Weigh two of them:

- If one is heavier, you've found the heaviest ball.
- If they balance, the heaviest ball is the third ball.

Thus, you can find the heavier ball in exactly 3 weighings.

Mutiliated Chessboard

This Problem is known as the Mutilated Chessboard and there is no solution. It is not possible to cover the mutilated chessboard with 31 dominoes. Here's why:
A domino always covers two adjacent squares. On a standard chessboard, the squares alternate in color (black and white). Thus, each domino must cover one black square and one white square.
However, if you remove two diagonally opposite corners from the chessboard, you remove two squares of the same color (both are either black or white). This means there will be 30 squares of one color and 32 squares of the other color.
Since each domino must cover one square of each color, it is impossible to cover all the squares with 31 dominoes because there is an imbalance in the number of black and white squares. Therefore, it's not possible to cover the mutilated chessboard with 31 dominoes.

Exercise 4: Submission *(5 Points)*

Send your answers to your tutor, on Zulip to receive feedback. Also if you have any questions regarding this exercise sheets, like a specific question about the riddles, then ask them in a public stream on Zulip.