**IDMA 2025**

**– Ugeseddel 1 –**

# Overview

Welcome to the *Introduction to Discrete Mathematics and Algorithms* course! This first week we will talk about what discrete math and algorithms and then get started on algorithms for **sorting** and **searching** in data. We will also discuss some simple data structures such as **stacks**, **queues**, and **linked lists**. Later in the course we will often assume that we have access to efficient implementations of these kind of objects, but it is good to understand how they can be constructed from first principles. All of this will also give us an opportunity to discuss what it means for an algorithm to be *efficient*, and how we can compare the efficiency of different algorithms using **asymptotic analysis**.

*Please note that the first problem set to be handed in will be posted in the coming days and will be due on* ***Wednesday Feb 12 at 12:59 CET****. Please make sure to familiarize yourself with the problem set rules posted on Absalon. In particular, you are welcome (and encouraged) to discuss the problems on the problem sets in groups of two or three persons, but your solutions should be written up individually, and you are not allowed to share any text, formulas, or code with others (or to copy such material from the internet or generate it via chatbots) when writing up your solution.* ***Passing all problem sets is a requirement to be eligible for the exam, so start working on the problems as soon as you can!***

# Reading Material

Cormen, Leiserson, Rivest, and Stein (CLRS): Chapters 1–3 and 10.

Please note that — as a general rule — *the lectures will not cover everything in the textbooks*, so just attending the lecture without reading the assigned material in the textbook is not sufficient.

# Plan for Lectures

## Monday Feb 3 at 13:15-15:00

- Introduction

- RAM model (how do we think about what a computer is)
- Pseudocode

### Wednesday Feb 5 at 10:15-12:00 and 13:15-15:00

- Searching
- Sorting
- Time complexity of algorithms
- Stacks, queues, and lists

## Software for Calculations and Visualisation

The math part of the course will actually not contain that many numerical calculations (just as the the algorithms part will not contain that much actual coding). Our main focus on this course is to make sure that you get a solid *theoretical, mathematical* understanding (after which coding things up will become a breeze). We will, however, occasionally encourage you to calculate some function values or draw a graph to visualize a function. For this purpose, you can use whatever tools you are most comfortable with. It does not matter whether this is an app on your phone or math software on your computer.

## Exercises

*Please note that the exercises below are* ***just suggestions for you.*** *At the end of the day, you have to read the material and work on it to make sure that you understand what is going on. This might involve solving fewer exercises than what is listed below, or might require solving more exercises. Only you can tell when you feel that you have mastered the material.*

You can of course also work on the problem set problems from previous years that are posted on Absalon, and are highly encouraged to do so. But it is probably a good idea to first do some of the exercises below to make sure that you have understood the material.

Exercises that might be slightly tricky are marked by stars [*].

## Exercises on Algorithm Analysis, Searching, and Sorting

1. In the following snippet of code `A` and `B` are arrays indexed from 1 to $n$ and containing numbers.

```
for (i := 1 upto n)
    B[i] := A[1]
    for (j := 2 upto i)
        B[i] := B[i] + A[j]
    B[i] := B[i] / i
```

   (a) Explain in plain language what the algorithm above does. In particular, what are the values computed and stored in the array `B`?

   (b) How does the running time of the function scale as a function of the array size $n$? (Focus on the highest-order term and ignore constant factors.)

   (c) Can you improve the algorithm (i.e., change the pseudocode) to run asymptotically faster while retaining the same functionality? In case you can design an asymptotically faster algorithm, analyse the time complexity of your new algorithm. For the best algorithm that you have—either your new one, or the old one—can you prove that the running time of this algorithm is asymptotically optimal?

2. [*] In the following snippet of code `A` is an array indexed from 1 to $n$ containing elements that can be compared using operators $>$, $=$, and $>$.

```
flip1 := FALSE
flip2 := FALSE
for (i := 1 upto n-1)
    if (A[i] > A[i+1] and not(flip1))
        flip1 := TRUE
    else if (A[i] < A[i+1] and flip1)
        flip2 := TRUE
if (flip1 and not(flip2))
    return "success"
else
    return "failure"
```

   (a) Explain in plain language what the algorithm above does. (In particular, what does it mean that it achieves "success"?)

(b) How does the running time of the function scale as a function of the array size $n$? (Focus on the highest-order term and ignore constant factors.)

(c) Can you modify the algorithm so that it "succeeds" when it encounters an odd number of "flips"? What does the pseudo-code look like? Does the time complexity change? If so, how?

3. Suppose we have the array $[1, 3, 4, 5, 8, 10, 11, 14, 15, 18, 20]$.

   (a) Illustrate how the linear search algorithm searches for element 16 in this array and for element 18.

   (b) Perform the same searches for elements 16 and 18 but using binary search.

4. Suppose we have the array elements as above, but permuted to yield the array $[1, 3, 20, 10, 8, 18, 15, 14, 11, 4, 5]$.

   (a) Run the selection sort algorithm on this array and illustrate (iteration by iteration) how the array gets sorted.

   (b) Run merge sort on the same array. Illustrate the recursive calls and how they are merged to form a sorted array.

   (c) [*] Which algorithm of selection sort or merge sort is fastest on this particular array?

5. CLRS 2.1-1

6. CLRS 2.2-2

7. CLRS 2.3-1

8. [*] CLRS 2-2

9. CLRS 3.2-1

## More Exercises on Asymptotic Analysis

Unless explicitly stated otherwise, you are free to use theorems from the notes on Absalon without a proof, as well as make use of any previously solved exercises.

(1) Two teams of computer science students compete in writing the most efficient algorithm to solve a task on an array with $N$ entries. Team 1 has constructed an algorithm that runs in

$$f_1(N) = 10 \cdot N^{10}$$

steps. Team 2 has constructed an algorithm that runs in

$$f_2(N) = 2^N$$

steps. One of the following statements must hold:

(A) Team 1's algorithm runs with fewer steps for all $N$.

(B) Team 2's algorithm runs with fewer steps for all $N$.

(C) Team 1's algorithm runs with fewer steps for some choices of $N$, while Team 2's algorithm runs with fewer steps for some other choices of $N$.

Draw the graphs of the functions. Which of the statements (A), (B), and (C) are true?

(2) Use the definition of big-$O$ to show that $x^3 + 12x - 3$ is $O(x^3)$.

(3) Use the definition of big-$O$ to show that $x^3 + 12x - 3$ is *not* $O(x^2)$.

(4) Use the rules *(B1)*–*(B6)*, *(L1)*–*(L6)*, *(M1)*–*(M4)* from the notes to determine which of the functions

$$x^2 + 2^x, (x+2)^2, (x + \log_{10} x)^2, x^3 - x, x^2 + \log_2 x$$

are $\Theta(x^2)$. Keep in mind that you also need to provide a justification whenever you claim that $f(x)$ is *not* $\Theta(x^2)$.

(5) Sort the sequences

$$\begin{aligned} a_n &= n2^n + n^7 \\ b_n &= 3^n + n^5 \\ c_n &= \sum_{k=1}^{n} 2^k \\ d_n &= \log_2 n + n^7 \end{aligned}$$

in increasing asymptotic order using only the results from this week's notes.

## Exercises on Stacks, Queues, and Lists

See the separate document with exercises on stacks, queues, and lists posted on Absalon.

## Some Additional Exercises (Possibly for Later)

Here are some extra exercises that might be useful if you want to repeat this material when preparing for the exam.

1. CLRS 1-1 (a bit tedious, but useful)
2. CLRS 2.1-2
3. [*] CLRS 2-3
4. CLRS 3.1-2
5. CLRS 3.2-2