KØBENHAVNS UNIVERSITET

# Introduktion til diskret matematik og algoritmer: Problem Set 1

**Due:** Monday February 16 at 12:59 CET.
**Submission:** Please submit your solutions via *Absalon* as a PDF file. State your name and e-mail address close to the top of the first page. Solutions should be written in LaTeX or some other math-aware typesetting system with reasonable margins on all sides (at least 2.5 cm). Please try to be precise and to the point in your solutions and refrain from vague statements. Never, ever just state the answer, but always make sure to explain your reasoning. *Write so that a fellow student of yours can read, understand, and verify your solutions.* In addition to what is stated below, the general rules for problem sets stated on *Absalon* always apply.
**Collaboration:** Discussions of ideas in groups of two to three people are allowed—and indeed, encouraged—but you should always write up your solutions completely on your own, from start to finish, and you should understand all aspects of them fully. It is not allowed to compose draft solutions together and then continue editing individually, or to share any text, formulas, or pseudocode. Also, no such material may be downloaded from or generated via the internet to be used in draft or final solutions. Submitted solutions will be checked for plagiarism.
**Grading:** A score of 120 points is guaranteed to be enough to pass this problem set.
**Questions:** Please do not hesitate to ask the instructor or TAs if any problem statement is unclear, but please make sure to send private messages—sometimes specific enough questions could give away the solution to your fellow students, and we want all of you to benefit from working on, and learning from, the problems. Good luck!

**1** (50 p) This problem is about different representations of integers.

**1a** Write the binary number $(110)_2$ in decimal notation.

**1b** Write the decimal number 110 in binary notation.

**1c** Write the octal number $(2025)_8$ in decimal notation.

**2** (50 p) Use the algorithm we have learned for determining $d = \gcd(m, n)$ for the numbers below, showing details of all function calls made, and then express $d$ as a linear combination of $m$ and $n$.

**2a** $m = 38$ and $n = 14$.

**2b** $m = 117$ and $n = 69$.

**3** (60 p) In the following snippet of code `A` is an array indexed from 1 to $n$ that contains numbers.

```
j := 1
while (j <= n) {
    A[j] := 0
    for i := j downto 1 {
        A[j] := A[j] + i * i
    }
    j := j + 1
}
```

**3a** Explain in plain language what the algorithm above does. In particular, what are the numbers that are computed and stored in the array `A`?

**3b** Provide an asymptotic analysis of the running time as a function of the array size $n$. (That is, state how the worst-case running time scales with $n$, focusing only on the highest-order term, and ignoring the constant factor in front of this term.)

**3c** Can you improve the code to run faster while retaining the same functionality? How much faster can you get the algorithm to run? Analyse the time complexity of your new algorithm. Can you prove that it is asymptotically optimal? (That is, that no algorithm solving this problem can run faster except possibly for a constant factor in the highest-order term or except for improvements in lower-order terms.)

**4** (70 p) In the following snippet of code `A` and `B` are arrays indexed from 1 to $n$ that contain numbers.

```
j    := n
good := TRUE
while (j >= 1 and good) {
    s := A[j]
    for i := j - 1 downto 1 {
        s := s + A[i]
    }
    s := s / j
    if (s > B[j]) {
        good := FALSE
    }
    else
    {
        j := j - 1
    }
}
return good
```

**4a** Explain in plain language what the algorithm above does. In particular, when does the algorithm return `TRUE` or `FALSE` and why?

**4b** Provide an asymptotic analysis of the running time as a function of the array size $n$. (That is, state how the worst-case running time scales with $n$, focusing only on the highest-order term, and ignoring the constant factor in front of this term.)

**4c** Can you improve the code to run faster while retaining the same functionality? How much faster can you get the algorithm to run? Analyse the time complexity of your new algorithm. Can you prove that it is asymptotically optimal? (That is, that no algorithm solving this problem can run faster except possibly for a constant factor in the highest-order term or except for improvements in lower-order terms.)