KØBENHAVNS UNIVERSITET

Introduktion til diskret matematik og algoritmer: Problem Set 1

Due: Wednesday February 14 at 9:59 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. 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 (60 p) In the following snippet of code `A` is an array indexed from 1 to n containing elements that can be compared using the operator $=$.

```
for (i := 1 upto n)
    B[i] := 0
for (i := 1 upto n)
    for (j := 1 upto n)
        if (i != j and A[i] == A[j])
            B[i] := B[i] + 1
return B
```

1a Explain in plain language what the algorithm above does. What is the meaning of the entries in the array `B` that the algorithm returns?

1b 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.)

1c Suppose that we are guaranteed that all elements in the array `A` are integers between 1 and n. 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?

2 (70 p) Consider the snippet of code

```
check (A, lo, hi)
    mid := floor ((lo + hi) / 2)
    success := TRUE
    i := lo
    while (i < mid and success)
        if (A[i] > A[mid])
            success := FALSE
        i := i + 1
    i := mid + 1
    while (i <= hi and success)
        if (A[i] < A[mid])
            success := FALSE
        i := i + 1
    if (lo < mid - 1 and success)
        success := check (A, lo, mid - 1)
    if (mid + 1 < hi and success)
        success := check (A, mid + 1, hi)
    return success
```

where `A` is an array indexed from 1 to `A.size` that contains elements that can be compared, and the function `floor` rounds down to the nearest integer.

2a Explain in plain language what the result is of an algorithm call `check (A, 1, A.size)` (i.e., do not just rewrite the pseudocode word by word, so that your text is just a more verbose version of the pseudocode, but interpret what the code is doing and why). In particular, what holds when the algorithm returns `TRUE`?

2b Provide an asymptotic analysis of the running time as a function of the size of the array `A`.

2c 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?

3 (70+ p) In the following snippet of code `A` is an array indexed from 1 to n containing integers.

```
found := FALSE
i := 1
while (i <= n and not(found))
    j := 1
    while (j <= n and not(found))
        k := 1
        while (k <= n and not(found))
            if (i != j and j != k and i != k and A[i]+A[j]+A[k] == 0)
                found := TRUE
            else
                k := k+1
        if (not(found))
            j := j+1
    if (not(found))
        i := i+1
if (found)
    return (i, j, k)
else
    return "failed"
```

3a Explain in plain language what the algorithm above does. What is the triple of numbers returned when the algorithm does not "fail"?

3b Provide an asymptotic analysis of the running time as a function of the array size.

3c 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?

3d *Bonus problem (worth 40 p extra):* This is in fact a well-known research problem. What information can you find about this on the internet? What are the best known upper and lower bounds for algorithms solving this problem? Explain how you dug up the information, and give references to where it can be found. (Note that for this problem, in contrast to most other problems on the problem sets, you are expected and encouraged to search on the Internet for answers.)