KØBENHAVNS
UNIVERSITET

Diskret Matematik og Formelle Sprog: Problem Set 1

Due: Monday February 15 at 23:59 CET.
Submission: Please submit your solutions via *Absalon* as 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. *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 in the course information 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 pseudo-code. Also, no such material may be downloaded from the internet and/or used verbatim. 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 on the problems. Good luck!

1 In the following snippet of code `A` is an array indexed from 1 to n containing elements that can be compared using the operator $<$.

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

1a (30 p) Explain in plain language what the algorithm above does. What is the number that the algorithm returns when it does not return “failed”, and when does the algorithm declare failure?

1b (20 p) 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 (20 p) 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 improvements in lower-order terms.)

2 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
```

2a (30 p) Explain in plain language what the algorithm above does. What is the meaning of the entries in the array `B` that the algorithm returns?

2b (20 p) 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.)

2c (20 p) 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?

3 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 (20 p) Explain in plain language what the algorithm above does. What is the triple of numbers returned when the algorithm does not “fail”?

3b (20 p) Provide an asymptotic analysis of the running time as a function of the array size.

3c (40 p) 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.