KØBENHAVNS UNIVERSITET

Diskret Matematik og Formelle Sprog: Problem Set 1

Due: Monday February 14 at 12:59 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 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 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 from, the problems. Good luck!

1 In the following snippet of code A and B are arrays indexed from 1 to n that contain numbers.

```
for i := 1 upto n {
    B[i] := 1
    for j := 1 upto i {
        B[i] := B[i] * A[j]
    }
}
```

1a (20 p) Explain in plain language what the algorithm above does. In particular, what is the meaning of the entries B[i] after the algorithm has terminated?

1b (10 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) 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 improvements in lower-order terms.)

2 In the following snippet of code `A` is an array indexed from 1 to n that contains integers, and `B` is an auxiliary array, also indexed from 1 to n, that is meant to contain Boolean values.

```
for i := 1 upto n {
    if (A[i] < 1 or A[i] > n)
        return "failure"
}
i     := 1
found := -1
while (i <= n and found < 0) {
    for j := 1 upto n {
        B[j] := false
    }
    j := i
    while (B[j] == false) {
        B[j] := true
        j := A[j]
    }
    if (A[A[j]] == j)
        found := j
    i := i + 1
}
return found
```

2a (30 p) Explain in plain language what the algorithm above does. In particular, when does it return a positive value, and, if it does, what is the meaning of this value?

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) 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 $n \geq 2$ containing integers.

```
search (A, lo, hi)
    if (A[lo] >= A[hi])
        return "failure"
    else if (lo + 1 == hi)
        return lo
    else
        mid = floor ((lo + hi) / 2))
        if (A[mid] > A[lo])
            search (A, lo, mid)
        else
            search (A, mid, hi)
```

The first call to the algorithm is `search (A, 1, n)`, where n is whatever size (at least 2) the array has.

3a (30 p) Explain in plain language what the algorithm above does. If the algorithm returns something other than `"failure"`, then what is the meaning of the value returned?

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

3c (20 p) Could it be the case that recursive calls of the algorithm also return `"failure"`, or would it be sufficient to check just once before making the first recursive call? If we get the additional guarantee that all elements in the array are distinct, could we remove the `"failure"` check completely, since we would be guaranteed to never have this answer returned anyway? What about if we get the additional guarantee that the array is sorted in increasing order? What if both of these extra guarantees apply?