KØBENHAVNS UNIVERSITET

Diskret Matematik og Formelle Sprog: Problem Set 2

Due: Monday February 27 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. 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 (70 p) Use mathematical induction to solve the following two problems.

1a (30 p) For which positive integers n does the inequality $3^n > n^3$ hold?

1b (40 p) For which positive integers n does the inequality

$$1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} + \ldots + \frac{1}{\sqrt{n}} = \sum_{i=1}^{n} \frac{1}{\sqrt{i}} > \sqrt{n}$$

hold?

2 (50 p) When Jakob started preparing this course last autumn, he ran the following interesting experiment several times. He picked a random set S consisting of 1012 distinct integers between 1 and 2022 (i.e., in mathematical notation he had $|S| = 1012$ and $1 \leq i \leq 2022$ for all $i \in S$), and then studied the numbers in this set S. Every time, he found that there was some number $c \in S$ that was the sum $c = a + b$ of two other numbers $a, b \in S$ (not necessarily distinct). Can you prove that this was not a coincidence, but that this must always happen?
(*Bonus problem:* What if we require $a, b, c \in S$ to be all different?)

3 (50 p) Another problem that Jakob got interested in when planning the DMFS course last autumn was to find all solutions to the equation

$$x^2 = 2022 + y^2 \tag{1}$$

for integer-valued x and y. Jakob actually worked on this pretty hard, but in the end had to give up because he was not able to solve this equation (and was planning to keep this secret, until he finally had to cobble together a second problem set for the DMFS course and realized that this might be an opportunity to get some help from the students).

Can you help Jakob by proving that in fact there are no solutions to Equation (1) if we insist that x and y should be integers?

Hint: Rewrite the equation and factor it.

4 (60 p) This problem is about the merge sort algorithm that we learned about during the first week of lectures.

4a (30 p) Run merge sort by hand on the array

$$A = [8, 4, 2, 9, 3, 1, 5, 10, 7, 6] \tag{2}$$

as in the handwritten notes for the lectures. Show in every step of the algorithm what recursive calls are made and how the results from these recursive calls are combined. Make sure to explain the merge steps carefully, in particular, the final (and most interesting) merge. (Any clear way of explaining is fine—you do not have to learn how to draw pictures in LaTeX if you do not want to.)

4b (30 p) The reason merge sort runs in time essentially $n \log_2 n$, as explained in class, is that the list of elements to be sorted is split into two lists of half the size in each recursive call, which means that after $\log_2 n$ recursive calls we get lists of size 1 that are already (vacuously) sorted. Merging all the lists takes a linear amount of work for each level of recursion, meaning that the total running time scales like $n \log_2 n$.

When Jakob thought more about this, he realized that if we would make a three-way split in each recursive call into three lists of a third of the size, then the recursion tree would terminate after $\log_3 n < \log_2 n$ levels, and the merging would still cost a linear amount of work per level, resulting in running time $n \log_3 n < n \log_2 n$. Or if in every recursive call we subdivided the list into four lists of equal size, then the recursion tree would only be of depth $\log_4 n$, and the total work of splitting and merging together the sorted lists would be $n \log_4 n < n \log_3 n$.

Pushing this idea to the limit, after a lot of calculations Jakob found that if in every level we do a $\sqrt{n}$-wise split of the list of size n into $\sqrt{n}$ lists of size $\sqrt{n}$, then the total running time of splitting and merging would be $n \log_{\sqrt{n}} n = 2 \cdot n = \mathrm{O}(n)$, meaning that we have a linear-time algorithm! Are you buying Jakob's arguments for this, or can you see any problems with his linear-time sorting algorithm? Please explain clearly what arguments you have to support or refute Jakob's claims.

5 (70 p) February–March is the worst time of the year for Jakob's pet frog. Jakob is busy teaching in Copenhagen and the frog is all alone at home in Lund and has to find other entertainment during long, dark afternoons and evenings.

One day, the frog came up with a new game with the following rules: for some positive integer n, make jumps on a straight line of lengths $i = 1, 2, \ldots, n$, each jump being either to the left or the right, and try to end up in the same position. The frog immediately got captivated by this fascinating game, and quickly figured out for $n = 3$ that it is possible to jump 1 step right, 2 more steps right, and then 3 steps left to get back to the starting position. For $n = 4$, one possible strategy is to jump 1 step right, 2 steps left, 3 more steps left, and then 4 steps right to get back to the starting position. For $n = 5$, however, things started to get a little bit too complicated for froggy...

At this point, the frog remembered what Jakob often says, namely that the proper way of approaching situations like this is to model the problem mathematically, design an algorithm for it, prove that the algorithm is correct, and finally analyse the complexity of the algorithm. But exhausted after a day full of intellectual hard labour, the frog was not able to make any further progress.

Can you help the frog to solve this problem? That is:

1. Design an algorithm that, given a positive integer n, prints out a sequence of left and right jumps of increasing length $1, 2, \ldots, n$ that will end up in the starting position. (If there is no such sequence for a given n, then the algorithm should of course report this.)

   Sample output for your algorithm could look something like

   ```
   Jumping sequence of length 3:
   Jump length 1 right
   Jump length 2 right
   Jump length 3 left
   ```

   or

   ```
   Jumping sequence of length 4:
   Jump length 1 right
   Jump length 2 left
   Jump length 3 left
   Jump length 4 right
   ```

   for the two examples above.

2. Prove that the algorithm that you have designed is correct.

3. Analyze the time complexity of your algorithm.

Partial results can also give points on this problem (as for any other problem). In particular, if you cannot quite design an algorithm, but still can say mathematically interesting things about this problem (like for which n it might or might not be possible to construct left-right jumping sequences that return to the starting point), then such insights will be rewarded generously if they are written down clearly and with proper explanations.

Hint: Continue where the frog left off and work out examples for some small values of n, and try to find some patterns.