COMPSCI 311: Introduction to Algorithms **Fall 2022**

Challenge Problems 1

due 9/21/2022 at 11:59pm in Gradescope

Instructions. Limited collaboration is allowed while solving problems, but you must write solutions yourself. List collaborators on your submission.

You can choose which problems to complete, but must submit at least one problem per assignment. See the course page for information about how challenge problems are graded and contribute to your homework grade. Since you don't need to complete every problem, you are encouraged to focus your efforts on producing high-quality solutions to the problems you feel confident about. There is no benefit to guessing or writing vague answers.

If you are asked to design an algorithm, please (a) give a precise description of your algorithm using either pseudocode or language, (b) explain the intuition of the algorithm, (c) justify the correctness of the algorithm; give a proof if needed, (d) state the running time of your algorithm, (e) justify the running-time analysis.

Submissions. Please submit a PDF file. You may submit a scanned handwritten document, but a typed submission is preferred. Please assign pages to questions in Gradescope.

Problem 1. Stable Matchings Consider a version of the stable matching problem where there are n students and n colleges as before. Assume each student ranks the colleges (and vice versa), but now we allow ties in the ranking. In other words, we could have a school that is indifferent two students s_1 and s_2, but prefers either of them over some other student s_3 (and vice versa). We say a student s *prefers* college c_1 to c_2 if c_1 is ranked higher on the s's preference list and c_1 and c_2 are not tied.

1. **Strong Instability.** A strong instability in a matching is a student-college pair, each of which prefer each other to their current pairing. In other words, neither is indifferent about the switch. Does there always exist a matching with no strong instability? Either give an example instance for which all matchings have a strong instability (and prove it), or give and analyze an algorithm that is guaranteed to find a matching with no strong instabilities.

2. **Weak Instability.** A weak instability in a matching is a student-college pair where one party prefers the other, and the other may be indifferent. Formally, a student s and a college c with pairs c' and s' form a weak instability if either

   - s prefers c to c' and c either prefers s to s' or is indifferent between s and s'.
   - c prefers s to s' and s either prefers c to c' or is indifferent between c and c'.

   Does there always exist a perfect matching with no weak instability? Either give an example instance for which all matchings have a weak instability (and prove it), or give and analyze an algorithm that is guaranteed to find a matching with no weak instabilities.

Problem 2. Stable Matching Running Time. In class, we saw that the Propose-and-reject algorithm terminates in at most n^2 iterations, when there are n students and n colleges.

1. It seems possible that the algorithm may complete in fewer rounds if the preference lists have a certain structure. Describe how to construct preference lists for any number n of colleges and students, such that the propose-and-reject algorithm will complete in only $O(n)$ rounds when run on these preference lists.

2. Could it be the case that the running time is actually $O(n)$ for all preference lists? Show that this is not true by designing preference lists so that the number of rounds of the algorithm is $\Omega(n^2)$.

Problem 3. Given an array S of n characters, you want to compute the number of occurrences of the character 'x' among the consecutive entries $S[i], S[i+1], \ldots, S[j]$, for all possible starting and ending positions i and j, that is, all i and j such that $1 \le i < j \le n$. You'll store the resulting numbers in an output array B. Consider the following algorithm for this problem:

for $i = 1, 2, \ldots, n$
    for $j = i + 1, \ldots, n$
        Scan the entries $S[i], S[i+1], \ldots, S[j]$ to count the number of 'x' characters
        Store the result in $B[i, j]$.

1. Find a function f such that the running time of the algorithm is $O(f(n))$, and clearly explain why.
2. For the same function f argue that the running time of the algorithm is also $\Omega(f(n))$. (This establishes an asymptotically tight bound $\Theta(f(n))$.)
3. Design and analyze a faster algorithm for this problem. You should give an algorithm with running $O(g(n))$, where $g(n)$ grows strictly more slowly than $f(n)$ (a precise definition of "strictly slower than" would be that $\lim_{n\to\infty} g(n)/f(n) = 0$).

Problem 4. Testing board strength. You have a supply of identical boards and n one-pound weights. You want to find the "bending strength" of a board, which is the largest weight (in pounds) it can support without breaking.

Your goal is to determine the bending strength using the smallest number of tests of different weights. However, you have a limited supply of boards, and need to determine the bending strength before breaking all of them.

For example, if you only had one board, you could test weights of 1, 2, 3, 4, 5, and so on, until the board breaks, and find the bending strength with at most n tests. With any other strategy, you would risk breaking the board before finding the bending strength.

Now, suppose you have two (identical) boards, so you can break one and still find the bending strength. Describe a strategy for finding the bending strength that requires you to test at most $f(n)$ different weights, for some function $f(n)$ that grows slower than linearly. (In other words, it should be the case that $\lim_{n\to\infty} f(n)/n = 0$.)