CpSc 8400: Design and Analysis of Algorithms

Instructor: Dr. Brian Dean — Spring 2021
Webpage: `http://www.cs.clemson.edu/~bcdean/` — TTh 12:30-1:45
Handout 2: Homework #1, Due Thursday 2/4/21 — Online Class

Some of the questions in this assignment ask you to "describe" an algorithm. You should generally do this in a clear, concise fashion in English, although you can also use pseudocode *if this adds clarity to your presentation*. Diagrams are also recommended if they help to clarify the operation of your algorithm. Any time you describe an algorithm, you should also say a few words about why it is correct (especially if this involves subtle or non-obvious observations) and also analyze its running time. Your write-ups generally do not need to be too lengthy as long as all the important details are present. Don't forget that typesetting is required, and do not forget to list your collaborators.

1-1. Enumerating Permutations. Consider a length-n array $A = (1, 2, 3, \ldots, n)$. We would like to step through all $n!$ permutations of A, updating the array as we go to represent each subsequent permutation. Permutations should be generated in lexicographic order; for example, with $n = 3$ we start with $A = (1, 2, 3)$, then move to $A = (1, 3, 2)$, $A = (2, 1, 3)$, $A = (2, 3, 1)$, $A = (3, 1, 2)$, and finally $A = (3, 2, 1)$. Please describe how to implement an operation *next-perm* with $O(1)$ amortized running time that modifies A to produce the next permutation in lexicographic order. Use a potential function in your analysis.

1-2. A Priority Queue Built from Sorted Arrays. Please do problem 92 (in chapter 5 of the book draft, on priority queues).

1-3. In-Place Matrix Transposition. Suppose we store an $m \times n$ matrix in *row-major* form — as an array of length $N = mn$ in which we list the elements of the matrix one row at a time from left to right. By taking the *transpose* of such a matrix, we convert it into *column-major* form — an array of length N listing the columns of the matrix one at a time, each one from top to bottom. Suppose the matrix is sufficiently large that we would like to permute its memory representation from row-major to column-major order *in place* (using only $O(1)$ extra storage beyond the memory that holds the matrix). Please describe and analyze an $O(N \log N)$ algorithm for solving this problem. As a hint, try to design an approach modeled on the "domination radius" algorithm from lecture, noting that for each element in the matrix, you can easily determine the location to which it needs to move, as well as the location of the element that will replace it.