CSE 5211: The Design and Analysis of Algorithms Fall 2025
Assignment 01: Fun with Fundamental Data Structures
Total Points: 18

Group Composition: Academic research is normally done collaboratively in groups. To give you experience with such collaboration, you are to work in groups of either two or three. Moreover, inter-collaboration between different groups is also encouraged. However, each group must write up their solutions independently without any discussion with members from other groups. Please see course syllabus for policies governing use of web resources, and generative-AI technologies.

Group collaboration is supposed to be productive, with all group members contributing equally to a discussion. Students need to take ownership of their products. This means that each member of the group has worked on all problems equally, and is responsible for everything written and produced in the assignments. A violation of academic integrity policies by one student affects all members in the group.

Please clearly indicate your group members as well as a list of collaborators. Failure to provide this list may also constitute to plagiarism and violates academic integrity. Each group is required to submit only once. Please note that you are only allowed to collaborate with other students registered currently for this course, as well as your course instructor.

Preamble: On questions that ask for description of an algorithm or a data structure, please do so clearly, precisely and with no ambiguity in plain English. Pseudocode is not necessary, but may be used for clarity. Actual implementation code in a specific language is not required. Several topics discussed in class may be used as "black boxes" without further elaboration. But new algorithms and data structures must be described and analyzed rigorously. Solutions must be neatly typeset in Latex or Word. Diagrams may be included to help with articulation. Specific examples may also be used to articulate your algorithm, however they cannot be used as a substitute solution to your algorithm. Technical communication is a very important skill, and highly convoluted and obtuse descriptions and analysis will be penalized, and in some cases not graded.

1. **Loopy Linked Lists (3 points):** Linked lists are fundamental data structures that store elements haphazardly in memory by linking them sequentially via pointers. Suppose you are given a pointer to the head element of a linked list. The last element in the linked list points back to any previous element in the list. The list therefore consists of two parts – a non-looping part and a looping part. Please describe an algorithm to find the number of elements stored in the list. To make things interesting, you are only allowed $O(1)$ additional space for storage. As a hint, suppose you didn't know about the looping part, how can you determine if the list ends with a `null` or whether in a loop.

2. **Moving in Sequence (3 points):** Consider the input sequence of elements $a_1, a_2, \ldots, a_n$. We would like to rearrange these to get another sequence $b_1, b_2, \ldots, b_n$. A single operation allows us to choose and move a single element a_j certain number of positions to the left. We might move the same element multiple times. Please determine the minimum number of operations needed to transform the input array a into b. For example, if $a = [5, 1, 3, 2, 4]$, and $b = [4, 5, 2, 1, 3]$, then there are two operations needed. The first operation moves 4 four places to its left, and the second operation moves 2 two places to its left.

3. **Domination Radius, Take 2 (3 points):** Consider the problem of computing the domination radius for each element a_i in an array $a_1, \ldots, a_n$ of comparable elements. Specifically, The *left-domination* $d_L(i)$ for an individual i is the number of consecutive elements leading upto i from the left that are all smaller than or equal to a_i. That is, we find the largest index j for which elements $a_{i-1}, a_{i-2}, \ldots a_{j+1}$ are all smaller than or equal to a_i, and that $a_j > a_i$, $1 \leq j < i \leq n$. The *left-domination* of i is then $i - j$. Similarly, we find the smallest index j, $i < j \leq n$, for which elements $a_{i+1}, a_{i+2}, \ldots a_{j-1}$ are all smaller than or equal to a_i, and that $a_j > a_i$. The *right-domination* $d_R(i)$ for an individual i is then $j - i$. The domination radius $d(i)$ for individual i is the minimum of its left and right dominations, i.e., $d(i) = \min\{d_L(i), d_R(i)\}$. Without loss of generality, you may assume that the ends contain people at infinite height, i.e., $a_0 = a_{n+1} = \infty$.

   Consider only computing the left-domination $d_L(i)$ for every element i. Suppose that we can take $O(n)$ extra space. Please show how we can can only make one pass over the entire input to compute thee left-domination for every element i, in only $O(n)$ total time. This suggests an $O(n)$ algorithm for computing the domination radius for every individual as we can scan twice, once computing the left-domination, and once computing the right-domination for every element i.

4. **Virtual Initialization (4 points):** One problem with arrays is that they must typically be initialized prior to use. On most computing environments, when we allocate an array of $O(n)$ words of memory they start out filled with "garbage" values (whatever data last occupied that block of memory), and we must spend $O(n)$ time setting the words in the block to some initial value. In this problem, we wish to design a data structure that behaves like an array (i.e., allowing us to retrieve the i^{th} value and modify the i^{th} value both in $O(1)$ time), but which allows for initialization to a specified value v in $O(1)$ time as well. That is, if we ask for the value of an element we have not modified since the last initialization, the result should be v. The data structure should occupy $O(n)$ space in memory (note that this could be twice or three times as large as the actual space we need to store the elements of the array), and the data structure should function properly regardless of whatever garbage is initially present in this memory. As a hint, try to combine the best features of an array and a linked list.

5. **In-place Matrix Transposition (5 points):** 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.

Submission: Please submit your solution as a single **pdf** file named `h01.pdf` on Canvas. Your solution must be written in a 12pt font size, and double-spaced. Please limit your answers to 2 pages per problem.