EECS 2011 A, Fall 2022 – Assignment 1

Remember to write your **full name** and **student number** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of the course materials, and the course staff, that you consulted**. You must submit a PDF file and Java files (if any) with the required filename. The PDF file could be generated using LaTeX(recommended), Microsoft Word (saved as PDF, **not** the `.docx` file), or other editor/tools. The submission must be **typed**. Handwritten submissions are **not** accepted.

Due September 30, 2022, 10:00 PM (on eClass); required file(s): a1sol.pdf, A1Q2.java

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. Answers that are technically correct that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets]. Your submitted file does **not** need to include/repeat the questions — just write your solutions.

The assignment must be completed individually.

1. **[10]** Consider the following data structures for implementing the **List** ADT.

   A. array (like the Java array)

   B. singly-linked list, with "next" only, and with "head" only.

   C. singly-linked list, with "next" only, and with "head" and "tail".

   D. reverse singly-linked list, with "prev" only, and with "tail" only.

   E. doubly-linked list with "next" and "prev", and with "head" and "tail".

   F. circular singly-linked list, with "next"only, and with "tail" only.

   G. circular doubly-linked list, with "next" and "prev", and with "tail" only.

   Note: **None** of the above data structures, **except for A (array)**, keeps a "size" attribute.

   For the following operations, think about the efficiency of implementations based each of the above data structures. Assume that the operations always receive inputs that don't cause any error.

   Operations:

   (a) Search for an element.

   (b) Get the middle element, i.e., with the list size being n, return the $\lfloor n/2 \rfloor$-th element in the list.

   (c) Insert a new element at the beginning of the list.

   (d) Insert a new element at the end of the list.

   (e) Delete an element from the beginning of the list.

   (f) Delete an element from the end of the list.

   What to include in your submission (the PDF): for each of the above operations, divide the given data structures (A, B, C, D, E, F, and G) into two categories: "fast" and "slow". "Fast" means that the operation can always be completed within a constant number of steps, and "slow" means that the number of steps taken by the operation is proportional to the size of the list. Note that it is possible that all data structures fall into the same category.

   Also, provide a **concise justification** for each of your categorization.

Programming Question

The best way to learn a data structure or an algorithm is to code it up. In this assignment, we have a programming exercise for which you will be asked to write some code and submit it. You may also be asked to include a write-up about your code in the PDF file that you submit. Make sure to **maintain your academic integrity** carefully, and protect your own work. All code submissions will be checked for plagiarism at the end of the term. It is much better to take the hit on a lower mark than risking much worse consequences by committing an academic offence.

2. [**10**] Consider the list `[5, 4, 3, 1, 2]`. Using a stack s and a single iteration through the list, we can output the elements of the list in sorted order, as follows:

   - Push 5 onto s
   - Push 4 onto s
   - Push 3 onto s
   - Output 1
   - Output 2
   - Pop from s and output the 3
   - Pop from s and output the 4
   - Pop from s and output the 5

   We would say here that we can output all 5 elements of the list in sorted order.

   Now consider another example, this time for the list `[4, 5, 2, 1, 3]`. We cannot output the entire list in sorted order this time using a stack, but we **can** output the **smallest** 3 elements as follows:

   - Push 4 onto s
   - Push 5 onto s
   - Push 2 onto s
   - Output 1
   - Pop from s and output the 2
   - Output 3

   ... and that's all we can do. If we were to pop from s now, we'd get 5, when what we really want to output next is 4. We would say here that 3 elements of the list can be output in sorted order.

   Download the starter file `A1Q2.java`, and complete the `solve` function. It takes an array, and returns the maximum number of elements of the array (starting from `1`) that can be output in sorted order by using one auxiliary stack as in the examples above. If the array is of length n, then its elements are guaranteed to be a permutation of the n integers from 1 to n. The stack class is provided in the starter code and you **must NOT modify it**. Read all instructions in the starter code carefully.

   What to include in your submission: You will submit the completed code `A1Q2.java`. Also, in the PDF file (`a1sol.pdf`), provide a clear explanation of how your algorithm works. Be concise: one or two short paragraphs should be enough.