The goal of these problem sets is to get you to explore concepts from class. This should help prepare you for the exams and, more importantly, for understanding the big ideas from ICS 46. These are not "repeat the steps from lecture" or "answer questions from the reading" problems. The homework will help you develop *new skills*. This means you will likely get stuck at some point: that's okay! Review the reading and your notes and ask questions in office hours or on the course message board.

This is due Wednesday, January 18 at 7:30 AM. You will need to submit this via GradeScope.

For instructions on how to get access to the course GradeScope, consult the syllabus. Be sure to also read carefully the section about artifact submission policies and the section about academic honesty; you are responsible for following those.

1. Suppose we have a Stack that can grow indefinitely. We want to create a new type of Stack data structure. The contents are templated and will always contain only comparable items (e.g., integers or strings, although you **may not** assume anything about the contents of the stack other than that any two are comparable: this means, given two elements $a$ and $b$, we can determine if $a < b$ and can also determine if $a$ and $b$ are equal). This new structure will have the usual Stack public operations and also a new function `findMin`, which will return the smallest element currently in the stack. We could implement this by searching the contents, but that takes time linear in the number of elements in the Stack.

   Explain how you would change the new Stack data structure to allow for this function to run in $\mathcal{O}(1)$ time. If you are storing additional private member data, state what else you are storing. If you are changing existing functions `push`, `pop`, or `top` (or the constructor/size functions), explain briefly how you are changing them. Their running times must still be $\mathcal{O}(1)$; for example, you cannot search the full stack for the newest minimum value at every `push` and `pop`. Explain in a few sentences how each change works and how, after any valid sequence of `push` and `pop` operations, we can always find the minimum element in the Stack in $\mathcal{O}(1)$ time.

   Your explanation should be sufficient that if, six months from now, you had to write the necessary modifications to a Stack data structure written in your favorite programming language, using *only* your written description, you could do so.

   You may assume that there will never be a duplicate item pushed to the Stack.

2. Suppose I want to implement the public member functions of a Stack (`push`, `pop`, `top`, `size`, `isEmpty`). However, instead of the private member data we had in the reading, I have *only* a single Queue. The Queue has unbounded capacity.

   Explain how you would implement each of those functions using just that Queue. You may use $\mathcal{O}(1)$ additional space within each function. The only Queue functions you may call are the constructor, enqueue, dequeue, front, size, and isEmpty.

   Your explanation should be sufficient that if, six months from now, you had to write the necessary modifications to a Stack data structure written in your favorite programming language, using *only* your written description, you could do so.

   For each function in the newly-created Stack's public interface, give the running time in $\mathcal{O}$ notation in terms of $n$, the number of elements currently in the stack.