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.

Due date: **October 3**, 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.

For this assignment, your response to each question must be contained within a single piece of paper (or digital equivalent). When you submit to GradeScope, you will need to inform the system which page of your PDF contains the answer. *Do this even if your submission is a single page.*

If you choose to hand-write this, you will need to *scan your work* to a PDF. Taking a picture and embedding in a document is *not okay* for this. If the grader cannot reasonably read your work, you might receive diminished or zero credit. If your uploaded document is not a PDF, you might receive diminished or zero credit, regardless of other considerations.

Please review the syllabus and course reference for the expectations of assignments in this class. Remember that problem sets are not online treasure hunts and that copying from external sources, including a previous quarter's posted solutions, is unacceptable. You are welcome to discuss matters with classmates, but remember the Kenny Loggins rule. Remember that you may not seek help from any source where not all respondents are subject to UC Irvine's academic honesty policy.

1. In the reading, we saw how to make a Stack data structure. Suppose we have a Stack that can grow indefinitely (for example, the push method has been fixed to double the size of the array when at capacity instead of throwing a StackFullException). We want to create a second Stack data structure, which I promise will always contain only comparable items (e.g., integers, strings, although you **may not** assume anything about the data other than that it is a comparable type). We also want to add a function `findMin` to the Stack interface that will return the smallest element currently in the stack. We could implement this by searching the array, but that takes time linear in the number of elements in the Stack.

   Explain how you would change the 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. You should also explain in a few sentences how the change works and how it achieves the goal.

   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 class, 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 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, give the running time in $\mathcal{O}$ notation in terms of n, the number of elements currently in the stack.

3. Give the asymptotic complexity for the worst case run-time of the following function `foo()` with respect to its input N. Function `bar()` runs in constant time with respect to N. For full credit you should show how you arrived at your answer. An answer that solely provides a value in Big O notation without reason will receive zero points. Answers that include variables other than N will receive fewer points.

```
1  void foo(unsigned int N) {
2      unsigned int S = 0;
3
4      for (unsigned int i = 1; i <= N; i++) {
5          S += i;
6      }
7
8      for (unsigned int j = 1; j <= S; j++) {
9          bar();
10     }
11 }
```

Not Collected Questions

These questions will not be collected. Please do not submit your solutions for them. However, these are meant to help you to study and understand the course material better. You are encouraged to solve these as if they were normal homework problems.

The textbook of Goodrich and Tamassia has excellent practice problems available. This homework (approximately) covers chapters 5.1 and 5.2, plus portions of 3.1 and 3.2 (including not-collected questions).

If you need help deciding which problems to do, consider trying these problems from the second edition of the textbook: R-5.3, R-5.4, R-5.5, R-5.6, R-5.9, R-3.4, R-3.7, R-3.10, C-3.6, C-3.13, C-3.17, C-3.19, C-3.20

If you want a larger project related to this, try P-5.2 and/or P-5.12.