This is project work due at the end of the term. (In fact, this is the last batch of project work. Congratulations!) In my estimation, each problem here is short or medium-length.

A. In qGates.py, implement the following function. Accomplish the task by wrapping the given gate U in bunch of SWAP gates. (This is the easiest way to accomplish the task. It is also the way that is truest to the idea that we're implementing large gates using small gates.)

```
def distant(gate):
    '''Given an (n + 1)-qbit gate U (such as a controlled-V gate, where V is
    n-qbit), performs swaps to insert one extra wire between the first qbit and
    the other n qbits. Returns an (n + 2)-qbit gate.'''
```

You might want to add a test function. I leave that up to you. I don't need to see your test.

B. In qGates.py, implement the following function. You are going to need cV, cZ, and cU gates, for specific one-qbit gates V, Z, and U, as discussed in class. Ideally, these would be engineered using our general construction of controlled one-qbit gates in terms of one-qbit gates and cNOTs. However, that is tedious, so let's not do it. Let's instead say that you can manually enter any two-qbit gate as a 4×4 matrix. You'll also need `distant`.

```
def ccNot():
    '''Returns the three-qbit ccNOT (i.e., Toffoli) gate. The gate is
    implemented using five specific two-qbit gates and some SWAPs.'''
```

Based on Day 10 Problem E, you should be able to write out the 8×8 matrix for the ccNOT gate explicitly. This gives you a check that your `ccNot` function is correct. I don't need to see your test.

C. In qGates.py, implement the following function. If you're not using Problem B, then you're doing it incorrectly.

```
def groverR3():
    '''Assumes that n = 3. Returns -R, where R is Grover's n-qbit gate for
    reflection across |rho>. Builds the gate from one- and two-qbit gates,
    rather than manually constructing the matrix.'''
```

You might want to write a test. I don't need to see your test.

D. (This problem is optional.) In qGates.py, implement the following function.

```
def distantLast(gate):
    '''Given an (n + 1)-qbit gate U (such as a V-controlled gate, where the
    last qbit controls a gate V on the first n qbits), performs swaps to
    insert one extra wire between the last qbit and the other n qbits. Returns
    an (n + 2)-qbit gate.'''
```

E. (This problem is optional.) In gGates.py, implement the following function. You'll need some two-qbit gates entered manually as 4×4 matrices, and you'll need `distantLast`.

```
def notCC():
    '''Returns the three-qbit NOTcc gate, in which the bottom two wires
    jointly control an X gate on the top wire. The gate is implemented using
    five specific two-qbit gates and some SWAPs.'''
```

F. (This problem is optional.) In gGates.py, implement the following function.

```
def iGroverR4():
    '''Assumes that n = 4. Returns the five-qbit gate (I tensor -R), where R is
    Grover's four-qbit gate for reflection across |rho>. Builds the gate from
    one- and two-qbit gates, rather than manually constructing the matrix.'''
```