A. The usual matrix multiplication algorithm uses $\mathcal{O}(mpq)$ arithmetic operations to multiply an $m \times p$ matrix and a $p \times q$ matrix. (Asymptotically faster algorithms are known to exist, but let's ignore them.)

This exercise is about the known $k = 1$ version of Grover's problem. In a classical simulation, such as `grover` in `qc.py`, most of the work lies in applying $((R \otimes I) \cdot F)^\ell$ to a certain state $|\psi\rangle$. It's natural to wonder: By tweaking the order of matrix multiplication, can we make our simulation faster or slower? Compare the asymptotic running times (in terms of n) of the following three strategies.

1. Starting with the state $|\psi\rangle$, apply F to get a new state $F \cdot |\psi\rangle$, then apply $R \otimes I$ to get a new state $(R \otimes I) \cdot F \cdot |\psi\rangle$, and then apply F, and then apply $R \otimes I$, and so on.

2. First compute $(R \otimes I) \cdot F$. Then compute $((R \otimes I) \cdot F)^\ell$ by repeated squaring. Then apply that single matrix to $|\psi\rangle$.

3. First compute $(R \otimes I) \cdot F$. Then compute $((R \otimes I) \cdot F)^\ell$ by the naive algorithm (not repeated squaring). Then apply that single matrix to $|\psi\rangle$.

B. This exercise is about the known $k \geq 1$ version of Grover's problem. Consider how the performance of Grover's algorithm is affected, as k changes from 1 to 2 to 3 and so on. That is, imagine that k is increasing but remains small.

1. As k increases, does Grover's algorithm perform better or worse? Discuss qualitatively.

2. To make it more quantitative, let's focus on one quantity: the expected number of invocations of F needed to obtain a single correct $\left|\delta^j\right\rangle$. As k increases, does this quantity increase or decrease?