192.036	VU Introduction to Quantum Computing	SS 2024
	Programming Exercise 3	Uwe Egly

Exercise 1

We want to implement a gate $\mathbf{mod[q]}$ taking input s of width n and whose action is defined as follows:

$$\mathbf{mod[q]} = \begin{cases} 0 & \text{if } \mathrm{hw}(s) \equiv 0 \bmod q; \\ 1 & \text{otherwise.} \end{cases}$$

The function $\mathrm{hw}(\cdot)$ is the Hamming weight and counts the number of 1s in the input. Such a gate is helpful in an implementation of the recursive Fourier sampling problem (RFS), also called recursive Bernstein-Vazirani problem.

We are given an operator $\mathbf{M}_q$ on $k = \lceil \log_2 q \rceil$ qubits which acts as follows:

$$\mathbf{M}_q|x\rangle = \begin{cases} |x + 1 \bmod q\rangle & \text{if } x < q; \\ |x\rangle & \text{if } x \geq q. \end{cases}$$

We are interested in the case $q = 3$. We claim that $\mathbf{M}_3$ with its matrix given below can be used to implement $\mathbf{mod[q]}$ for $q = 3$.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This matrix looks quite similar to the matrix of the **SWAP** gate. Simply exchange in $\mathbf{M}_3$ row 0 with row 2 in order to obtain **SWAP**.

1.1 An important condition $\mathbf{M}_q$ has to satisfy is $(\mathbf{M}_q)^q = \mathbf{I}$. Show that $(\mathbf{M}_3)^3$ results in the identity.

1.2 Implement a gate $\mathbf{M}_3$ using one $\mathbf{CNOT}^0$ gate and one **SWAP** gate. Recall that $\mathbf{CNOT}^0$ acts when the control qubit is in state $|0\rangle$. Use Qiskit and employ the unitary simulator to obtain the matrix form of your implementation. Is this matrix equal to $\mathbf{M}_3$?

1.3 Implement a single-controlled version $\mathbf{cM}_3$ of gate $\mathbf{M}_3$ in Qiskit. Test your implementation and document the test cases.

1.4 Implement the $\mathbf{mod[q]}$ gate for $q = 3$ and $n = 4$ input qubits as a gate array in Qiskit.

Start with $k = \lceil \log_2 q \rceil$ ancillary qubits and apply four $\mathbf{cM}_3$ gates each controlled by the negation of another single input qubit. The target qubits are the ancillary qubits. Apply then an OR gate with target r exploiting that $a \vee b$ is equivalent to $\neg(\neg a \wedge \neg b)$. Measure the negated r and uncompute.

Test your implementation with all possible 16 values for the input.