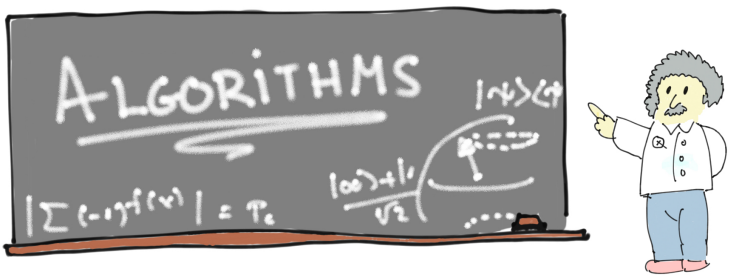


Algorithms: QFT adder [300 points]

Version: 1

Algorithms

The **Algorithms** category challenges will serve to give you a sense of commonly-used routines that are helpful when using real quantum computers. The quantum algorithms that you will see in this category are procedures that mathematically demonstrate some advantage over traditional/classical solutions such as the Deutsch-Jozsa algorithm, applications of the Quantum Fourier Transform (QFT), Grover's algorithm, and Quantum Phase Estimation (QPE).

Although these algorithms have very specific applications, they are commonly related to real-world problems. One of the clearest examples of this is the well-known Shor factorization algorithm, which revealed that finding prime factors of a number is equivalent to finding the period of certain functions, which can be achieved through QPE. It is for this reason that it is so important to know these basic ideas of quantum computation. Let's get to it! And good luck!

Problem statement [300 points]

In this challenge, you will generate a function capable of adding m to a number n encoded in binary. We encode the number n as a quantum state and perform some operations on it such that the result is $m + n$, also encoded as a quantum state. See Figure 1.

There are different techniques to perform addition with a quantum circuit, but one of the simplest ideas is to use the Quantum Fourier Transform (QFT). Let's talk first about the encoding procedure and the QFT.

How do we encode decimal integers as quantum states? For instance, the state $|011\rangle$ can be represented by $|3\rangle$ because $011 = 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 3$ (note that in this decimal representation, counting starts at zero). We want the inverse process: to encode an integer into a quantum state describing qubits (i.e., $|3\rangle \rightarrow |011\rangle$).

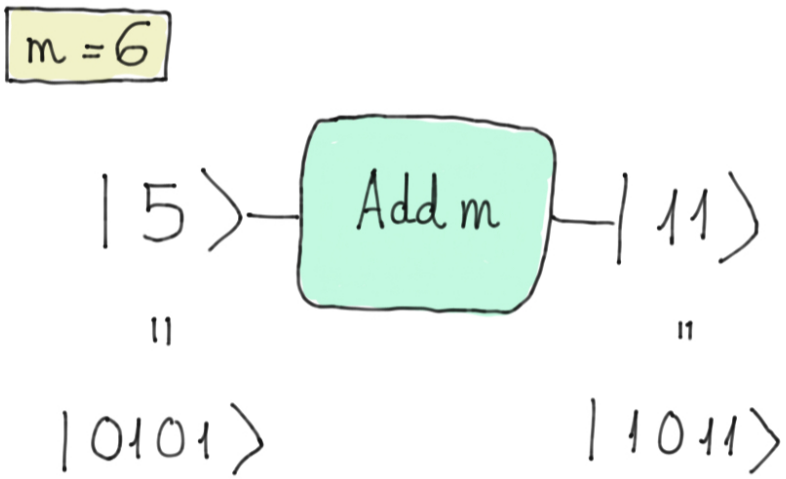
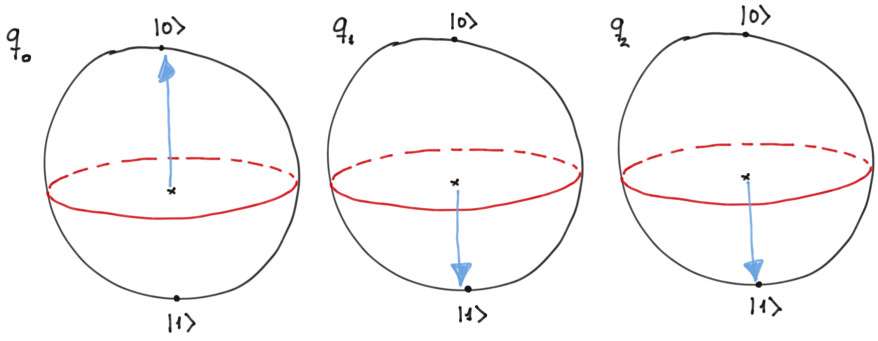


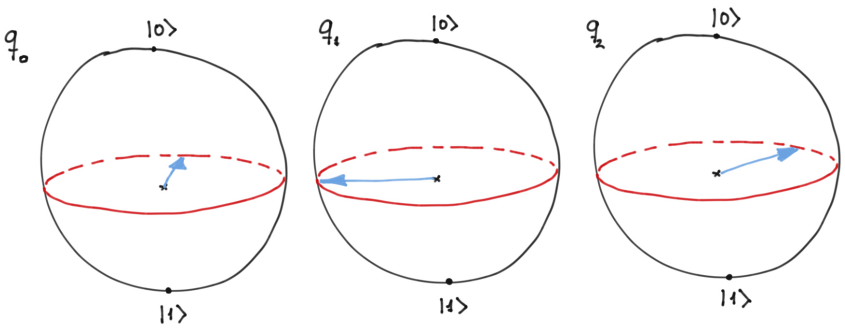
Figure 1: QFT Adder

In this representation, the QFT maps states as

$$QFT|x\rangle = \frac{1}{\sqrt{M}} \sum_{k=0}^{M-1} e^{\frac{2\pi i}{M} xk} |k\rangle,$$

where $M = 2^N$ is the number of possible N-qubit states and x is a positive integer. In PennyLane, the QFT operation is callable using `qml.QFT`. Essentially, the QFT maps a given state to a specific superposition of all 2^N basis states. Thinking of the Bloch sphere, each qubit gets individually rotated into the XY plane.





But, where exactly is each qubit pointing to when we apply a QFT to a state $|x\rangle$? Let's look at an example where $|x\rangle = |q_0, q_1, q_2\rangle$ ($q_i \in 0, 1$) and focus on the rightmost qubit q_2 (also the qubit associated to the 2^0 coefficient when counting in binary). Since we are working with three qubits, we will divide the circumference in $2^3 = 8$ equal parts, and as we move up in the computational basis, the coefficient $e^{i\theta}$ associated with the basis state grows by $2\pi\frac{1}{8}$. The rest of the qubits will progressively double the angle with respect to the previous qubit. We will say that if the phase of a qubit is, for example, $2\pi \times \frac{1}{4}$, then it is pointing to $\frac{1}{4}$.

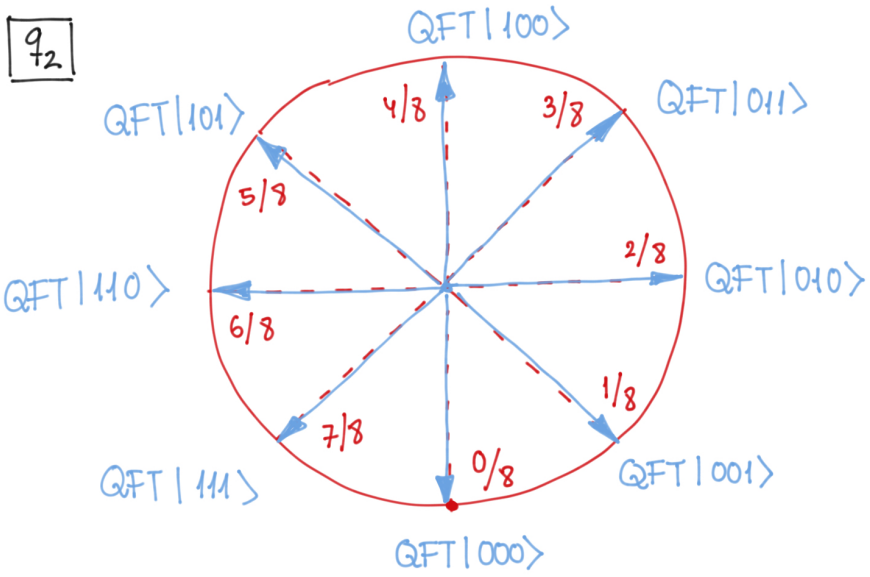


Figure 2: Rotations of qubits on the Bloch sphere with QFTs

Therefore, if the qubit q_2 is pointing to $\frac{3}{8}$, the next qubit q_1 will point to $\frac{6}{8}$, and finally q_0 points to $\frac{12}{8}$ which will be equivalent to $\frac{4}{8}$. See Figure 2.

One possible strategy to implement integer addition is to perform a QFT operation on our binary-encoded integer n (i.e., use `qml.QFT`). Then, we perform some operation on the state that modifies the angle of the individual $e^{\frac{2\pi i}{M}xk}$ phases. Finally, we reverse the QFT mapping via `qml.QFT(wires=wires).inv()`. You must implement this procedure.

The provided template `adder_QFT_template.py` file contains a function called `qfunc_adder` that you must complete. The `qml.QFT` command and its inverse are already written in the template. Between these commands, you must add code that implements the operations required to perform the addition of integers m and n in this manner. Here, n is fixed to 2^{N-1}. Then, the function `test_circuit` (*do not modify this function*) uses the operations in `qfunc_adder` to perform the addition (the output is binary-encoded).

Input

- `list(int)`: The first element of the list is m (the quantity to add to $n = 2^{N-1}$). The second element of the list is N (the number of qubits/wires).

Output

- `list(int)`: The result of $m + n$ encoded in binary. For instance, if the integer 9 is the result of $m + n$, then the output will be `[1,0,0,1]` for `n_wires = 4`.

Acceptance Criteria

In order for your submission to be judged as "correct":

- The outputs generated by your solution when run with a given `.in` file must match those in the corresponding `.ans` file.
- Your solution must take no longer than the `60s` specified below to produce its outputs.

You can test your solution by passing the `#.in` input data to your program as stdin and comparing the output to the corresponding `#.ans` file:

```
python3 {name_of_file}.py < 1.in
```

WARNING: Don't modify the code outside of the `# QHACK #` markers in the template file, as this code is needed to test your solution. Do not add any print statements to your solution, as this will cause your submission to fail.

Specs
Time limit: **60 s**

Version History

Version 1: Initial document.