## Exercise: simulating a perfect quantum computer with a classical computer

The goal is to write a Python code that emulates the execution of a quantum circuit on a (perfect) quantum computer.
You are given a list of pairs $(G_i, \vec{q}_i)$ $(i = 1 \dots n_G)$ with $G_i$ the name of the gate (possibly with a parameter if it is e.g a rotation gate) and $\vec{q}_i$ the qubits on which it is applied. It completely characterizes the quantum circuit. (The definition of the matrices corresponding to each gate name $G_i$ is assumed to be known and stored elsewhere). We aim at applying these gates to an initial state

$$|\Psi\rangle = |0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle \tag{2.19}$$

(on $n$ qubits) in order to obtain the final state of the quantum computer, from which we can compute output probabilities using Born's rule:

$$p(i) = |\langle i|\Psi\rangle|^2$$

To represent the state $|\Psi\rangle$, we are going to use a tensor representation as $\Psi_{b_1,b_2,\dots,b_n}$ with $b_i \in \{0,1\}$ via a numpy.array of shape $(2, 2, \dots, 2)$.

1. Create a "QuantumCircuit" class containing the above information (total number of qubits, sequence of gates and their target qubits)
2. Initialize a numpy.array to represent the initial state.
3. Given a single-qubit gate acting on qubit $q$, use the numpy.tensordot method to apply the gate on the current state.
4. Write a test checking that the Hadamard gate acting on the first qubit of the $|0, 0\rangle$ state has the right action. (Write the test with an "assert" function so that running the test will raise an exception if it fails)
5. Given a generic two-qubit gate acting on qubits $(q_1, q_2)$, use the numpy.tensordot method to apply the gate on the current state.
6. Write a test checking that a CNOT gate acting on qubits $(0, 1)$ of state $(|10\rangle + |00\rangle)/\sqrt{2}$ yields the right state.
7. What about a CNOT gate acting on $(1, 0)$?
8. Given the final state $|\Psi\rangle$, compute the vector of bitstring probabilities $p(i)$.
9. Write a test checking that it works.
10. What is the space complexity (=memory cost) of the algorithm as a function of the number $n$ of qubits?
11. What is the time complexity of the algorithm as a function of the number $n$ of qubits and number $n_G$ of gates?
12. Check your answer to question 9 by measuring the duration of execution of your simulator for an increasing number of qubits and gates. Is it what you predicted?
13. Bonus 1:
    (a) create a simple random circuit generator by repeating the following operation $n_{\text{layers}}$ times: (i) apply random $x, y, z$ rotations on each of the $n$ qubits and (ii) add a wall of CNOT gates between qubits $2i$ and $2i + 1$ for even layers (swap even and odd qubits for odd layers)
    (b) execute this circuit with your simulator for a large $n_{\text{layers}}$, and compute the histogram of the probabilities $\mathbb{P}(p)$ (i.e the probability of getting a probability $p$). What do you observe?
14. Bonus 2:
    (a) write a function that creates the circuit corresponding to the quantum Fourier transform on $n$ qubits.
    (b) check that it works by checking that the output amplitudes are related to the input amplitudes through the formula of the discrete Fourier transform.