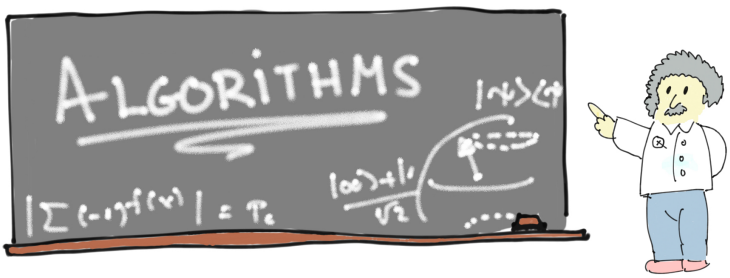

Algorithms: Deutsch-Jozsa Strikes Again [500 points]

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 [500 points]

The Deutsch-Jozsa algorithm was one of the first to demonstrate a quantum advantage. Briefly, we are given a black-box function (or oracle) $f : \{0,1\}^N \rightarrow \{0,1\}$, which satisfies one of these two properties:

- f is constant: it always returns the same value (it can be 0 or 1).
- f is balanced: half of the values, it takes the value 0 and for the other half, the value 1.

An oracle with these characteristics needs three qubits: the first two qubits prepresent the input x, and the third qubit marks the output $f(x)$.

Here, we will be looking at a variation of the original Deutsch-Jozsa problem: we are given four different functions f_0, f_1, f_2 and f_3 representing four different oracles which are either constant or balanced. As before, each oracle is a 3-qubit

operator: the first two qubits represent the input to the function and the third qubit is the output. We are guaranteed that this set of functions satisfies one of the following conditions:

- All functions are constant.
- All functions are balanced.
- Two functions are constant and two are balanced.

The objective of this challenge is to determine whether the four functions are of the same type (all constant or all balanced) or are equally split (two functions are constant and two are balanced). You are only allowed to run one shot in one circuit, and you may not use more than eight qubits.

The idea is to apply the original Deutsch-Jozsa algorithm on a very particular oracle (see Figure 1). In this case, the input of the quantum circuit will be a number i from 0 to 3 (binary-encoded in two qubits) and the output (the last qubit) will be 1 if the function f_i is constant or 0 if it is balanced. In addition to these three qubits, we can use up to five additional auxiliary qubits. Thus, if this new oracle is constant, it means that f_0, f_1, f_2 and f_3 are all of the same type; otherwise, half are constant and half are balanced.

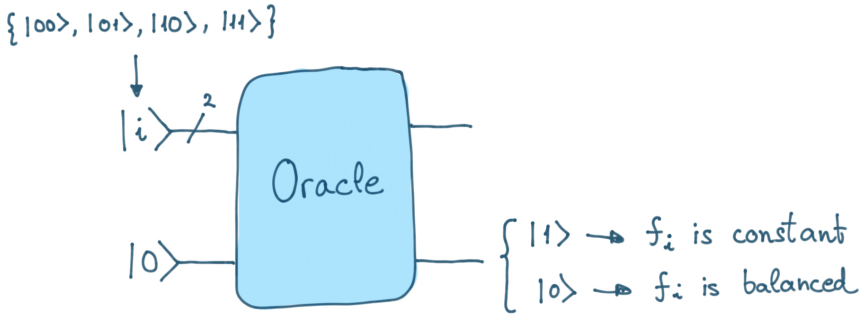

Figure 1: Oracle structure

The provided template `deustch_jozsa_strikes_again_template.py` contains a function called `deutsch_jozsa` that you must complete. This function determines whether the four input functions are of the same type (all constant or all balanced) or if two are constant and two are balanced. In this function, construct a circuit that allows you to make this distinction between the function types and output a string ("4 same" or "2 and 2") corresponding to the function types.

Input

- `list(int)`: A list of integers that are used to define the four functions f_{1-4}.

Output

- `str` : "4 same" or "2 and 2".

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.