Problem: Check whether a function permutes on a given set of values.

Example:

This function does permute:
F: {0,9,10,12} → {10,0,12,9}

This function doesn't permute:
F: {0,9,10,12} → {10,0,12,13}

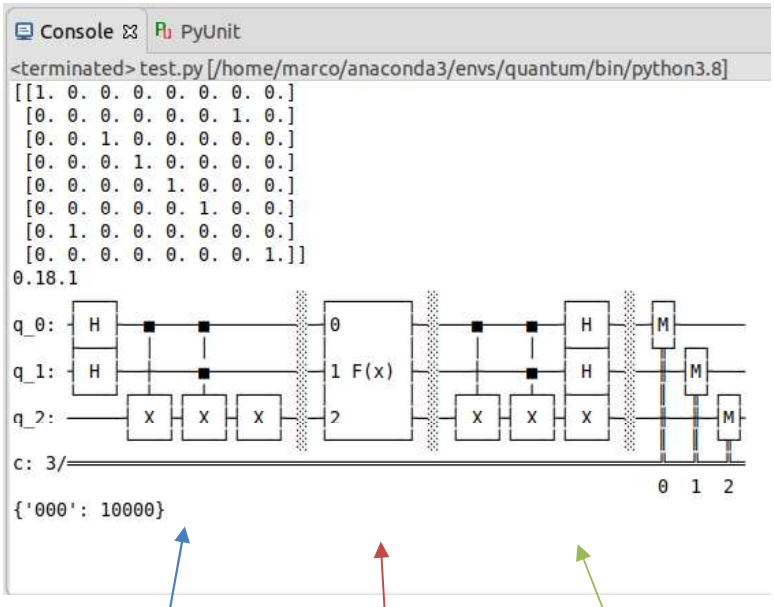
Classically, we would calculate each of the n elements individually, and compare the result set with the original set: F(0)=10, F(9)=0, F(10)=12, F(12)=9
So n-steps are needed. Both for calculating and for comparing.

On a quantum computer, this can be done in **one** step by putting elements in superposition.

$$F\left(\frac{1}{2}[0\rangle+\frac{1}{2}[9\rangle+\frac{1}{2}[10\rangle+\frac{1}{2}[12\rangle\right)=\frac{1}{2}[F(0)\rangle+\frac{1}{2}[F(9)\rangle+\frac{1}{2}[F(10)\rangle+\frac{1}{2}[F(12)\rangle=\frac{1}{2}[10\rangle+\frac{1}{2}[0\rangle+\frac{1}{2}[12\rangle+\frac{1}{2}[9\rangle$$

Inputstate Outputstate

You can see that the input and output states are the same, except for the swap.
However, because of the summation, the order doesn't matter. The state has not changed.
So the function only permutes the elements. Input and output state can be compared easily.
We can check with a calculation step whether a function only exchanges/permutes its elements, or produces new function values.



Measurement of final state in register

Encoder Inputstate
$\frac{1}{2}[1\rangle+\frac{1}{2}[4\rangle$
$+\frac{1}{2}[6\rangle+\frac{1}{2}[7\rangle$

Apply function!

Decoder Outputstate
$\frac{1}{2}[6\rangle+\frac{1}{2}[4\rangle$
$+\frac{1}{2}[1\rangle+\frac{1}{2}[7\rangle$

This algorithm was validated on an IBM-3Qbit-System.[1]

If the function permutes, the measure should return state |000> with 100% certainty.
There are other operations that give outputstate |000> as well. But assuming the permuting function is also injective, the probability of alternative ways goes straight against zero. The probability decreases with the number of elements to compare, and exponentially with the number of measurements you do.
So, at the end, 10 measures are enough to compare billion elements.

Injective permutation could be the scrambled letters in ciphertext:
A-> H, B->X, C->D, D->U, E->I, F->K, ...

Another application could be, to prove permutations and properties of the monstergroup (group theory), which has almost 10^54 elements.

[1] https://quantum-computing.ibm.com/login

Proof by exhausting all possible cases:

There are binomial coefficient (8|4) = 70 combinations.
Superposition of 4 numbers out of 8 possibilities (3bits = 2^3 = 8)
Nested loops can iterate through every combination.
As you can see in the output below, just line 17 maps to the correct result (1,0,0,0,0,0,0,0),
where all three qbits are zero.
In all other cases, the highest amplitude for finding all three qbits in zero, is: 0.75
So the highest probability to find all three qbits in zero, is the square of that: 0,5625.

Conclusion:
In the worst case, you almost have to measure twice.
Not for times as classically needed!

```
import numpy as np

decoder = np.array([[ 0. ,  0.5 , 0.,   0. ,  0.5 , 0.,   0.5 , 0.5],
 [ 0.,  -0.5,  0.  , 0. ,  0.5 , 0. ,  0.5 ,-0.5],
 [ 0. ,  0.5 , 0. ,  0. ,  0.5 , 0. , -0.5, -0.5],
 [ 0. , -0.5 , 0. ,  0.  , 0.5 , 0. , -0.5 , 0.5],
 [ 0.5 , 0. ,  0.5 , 0.5 , 0.  , 0.5 , 0. ,  0. ],
 [ 0.5 , 0.  , 0.5 ,-0.5 , 0. , -0.5 , 0. ,  0. ],
 [ 0.5 , 0. , -0.5, -0.5 , 0.  , 0.5 , 0. ,  0. ],
 [ 0.5 , 0. , -0.5 , 0.5 , 0. , -0.5 , 0. ,  0. ]])

'''
decoder_complex = np.array([[ 0. +0.0000000e+00j , 0.5-6.1232340e-17j , 0. +0.0000000e+00j,
   0. +0.0000000e+00j , 0.5+0.0000000e+00j , 0. +0.0000000e+00j,
   0.5-6.1232340e-17j , 0.5-1.2246468e-16j],
 [ 0. +0.0000000e+00j ,-0.5+6.1232340e-17j , 0. +0.0000000e+00j,
   0. +0.0000000e+00j , 0.5+0.0000000e+00j,  0. +0.0000000e+00j,
   0.5-6.1232340e-17j ,-0.5+1.2246468e-16j],
 [ 0. +0.0000000e+00j , 0.5-6.1232340e-17j , 0. +0.0000000e+00j,
   0. +0.0000000e+00j , 0.5+0.0000000e+00j , 0. +0.0000000e+00j,
  -0.5+6.1232340e-17j ,-0.5+1.2246468e-16j],
 [ 0. +0.0000000e+00j ,-0.5+6.1232340e-17j , 0. +0.0000000e+00j,
   0. +0.0000000e+00j , 0.5+0.0000000e+00j , 0. +0.0000000e+00j,
  -0.5+6.1232340e-17j , 0.5-1.2246468e-16j],
 [ 0.5+0.0000000e+00j , 0. +0.0000000e+00j , 0.5-6.1232340e-17j,
   0.5-1.2246468e-16j , 0. +0.0000000e+00j , 0.5-6.1232340e-17j,
   0. +0.0000000e+00j , 0. +0.0000000e+00j],
 [ 0.5+0.0000000e+00j , 0. +0.0000000e+00j , 0.5-6.1232340e-17j,
  -0.5+1.2246468e-16j , 0. +0.0000000e+00j, -0.5+6.1232340e-17j,
   0. +0.0000000e+00j , 0. +0.0000000e+00j],
 [ 0.5+0.0000000e+00j , 0. +0.0000000e+00j ,-0.5+6.1232340e-17j,
  -0.5+1.2246468e-16j , 0. +0.0000000e+00j , 0.5-6.1232340e-17j,
   0. +0.0000000e+00j , 0. +0.0000000e+00j],
 [ 0.5+0.0000000e+00j , 0. +0.0000000e+00j ,-0.5+6.1232340e-17j,
   0.5-1.2246468e-16j , 0. +0.0000000e+00j ,-0.5+6.1232340e-17j,
   0. +0.0000000e+00j , 0. +0.0000000e+00j]])
'''

#Nested loops:

count = 0
for a in range(2):
    for b in range(2):
        for c in range(2):
            for d in range(2):
```

```
            for e in range(2):
                for f in range(2):
                    for g in range(2):
                        for h in range(2):
                            if a+b+c+d+e+f+g+h==4:
                                statevector = 0.5*np.array([a,b,c,d,e,f,g,h])
                                print(count+1, " ", np.dot((decoder),statevector))
                                count = count + 1


print("Number of combinations: ",count)
```

```
1   [ 0.75  0.25 -0.25  0.25  0.25 -0.25  0.25 -0.25]
2   [ 0.5  0.  -0.5  0.   0.5 -0.5  0.   0. ]
3   [ 0.75  0.25 -0.25  0.25  0.25 -0.25 -0.25  0.25]
4   [ 0.5  0.   0.   0.5  0.5 -0.5  0.   0. ]
5   [ 0.5  0.5  0.   0.   0.5 -0.5  0.   0. ]
6   [ 0.5  0.  -0.5  0.   0.5  0.   0.  -0.5]
7   [ 0.75  0.25 -0.25  0.25  0.25  0.25 -0.25 -0.25]
8   [ 0.5  0.   0.   0.5  0.5  0.   0.  -0.5]
9   [ 0.5  0.5  0.   0.   0.5  0.   0.  -0.5]
10   [ 0.5  0.  -0.5  0.   0.5  0.  -0.5  0. ]
11   [ 0.25 -0.25 -0.25  0.25  0.75 -0.25 -0.25 -0.25]
12   [ 0.25  0.25 -0.25 -0.25  0.75 -0.25 -0.25 -0.25]
13   [ 0.5  0.   0.   0.5  0.5  0.  -0.5  0. ]
14   [ 0.5  0.5  0.   0.   0.5  0.  -0.5  0. ]
15   [ 0.25  0.25  0.25  0.25  0.75 -0.25 -0.25 -0.25]
16   [ 0.75 -0.25 -0.25 -0.25  0.25 -0.25  0.25 -0.25]
17   [1. 0. 0. 0. 0. 0. 0. 0.]
18   [ 0.75 -0.25  0.25  0.25  0.25 -0.25  0.25 -0.25]
19   [ 0.75  0.25  0.25 -0.25  0.25 -0.25  0.25 -0.25]
20   [ 0.75 -0.25 -0.25 -0.25  0.25 -0.25 -0.25  0.25]
21   [ 0.5 -0.5  0.   0.   0.5 -0.5  0.   0. ]
22   [ 0.5  0.   0.  -0.5  0.5 -0.5  0.   0. ]
23   [ 0.75 -0.25  0.25  0.25  0.25 -0.25 -0.25  0.25]
24   [ 0.75  0.25  0.25 -0.25  0.25 -0.25 -0.25  0.25]
25   [ 0.5  0.   0.5  0.   0.5 -0.5  0.   0. ]
26   [ 0.75 -0.25 -0.25 -0.25  0.25  0.25 -0.25 -0.25]
27   [ 0.5 -0.5  0.   0.   0.5  0.   0.  -0.5]
28   [ 0.5  0.   0.  -0.5  0.5  0.   0.  -0.5]
29   [ 0.75 -0.25  0.25  0.25  0.25  0.25 -0.25 -0.25]
30   [ 0.75  0.25  0.25 -0.25  0.25  0.25 -0.25 -0.25]
31   [ 0.5  0.   0.5  0.   0.5  0.   0.  -0.5]
32   [ 0.5 -0.5  0.   0.   0.5  0.  -0.5  0. ]
33   [ 0.5  0.   0.  -0.5  0.5  0.  -0.5  0. ]
34   [ 0.25 -0.25  0.25 -0.25  0.75 -0.25 -0.25 -0.25]
35   [ 0.5  0.   0.5  0.   0.5  0.  -0.5  0. ]
36   [ 0.5  0.  -0.5  0.   0.5  0.   0.5  0. ]
37   [ 0.75  0.25 -0.25  0.25  0.25  0.25  0.25  0.25]
38   [0.5 0.  0.  0.5 0.5 0.  0.5 0. ]
39   [0.5 0.5 0.  0.  0.5 0.  0.5 0. ]
40   [ 0.5  0.  -0.5  0.   0.5  0.   0.   0.5]
41   [ 0.25 -0.25 -0.25  0.25  0.75 -0.25  0.25  0.25]
42   [ 0.25  0.25 -0.25 -0.25  0.75 -0.25  0.25  0.25]
43   [0.5 0.  0.  0.5 0.5 0.  0.  0.5]
44   [0.5 0.5 0.  0.  0.5 0.  0.  0.5]
45   [ 0.25  0.25  0.25  0.25  0.75 -0.25  0.25  0.25]
```

46 [0.5 0. -0.5 0. 0.5 0.5 0. 0.]
47 [0.25 -0.25 -0.25 0.25 0.75 0.25 0.25 -0.25]
48 [0.25 0.25 -0.25 -0.25 0.75 0.25 0.25 -0.25]
49 [0.5 0. 0. 0.5 0.5 0.5 0. 0.]
50 [0.5 0.5 0. 0. 0.5 0.5 0. 0.]
51 [0.25 0.25 0.25 0.25 0.75 0.25 0.25 -0.25]
52 [0.25 -0.25 -0.25 0.25 0.75 0.25 -0.25 0.25]
53 [0.25 0.25 -0.25 -0.25 0.75 0.25 -0.25 0.25]
54 [0. 0. 0. 0. 1. 0. 0. 0.]
55 [0.25 0.25 0.25 0.25 0.75 0.25 -0.25 0.25]
56 [0.75 -0.25 -0.25 -0.25 0.25 0.25 0.25 0.25]
57 [0.5 -0.5 0. 0. 0.5 0. 0.5 0.]
58 [0.5 0. 0. -0.5 0.5 0. 0.5 0.]
59 [0.75 -0.25 0.25 0.25 0.25 0.25 0.25 0.25]
60 [0.75 0.25 0.25 -0.25 0.25 0.25 0.25 0.25]
61 [0.5 0. 0.5 0. 0.5 0. 0.5 0.]
62 [0.5 -0.5 0. 0. 0.5 0. 0. 0.5]
63 [0.5 0. 0. -0.5 0.5 0. 0. 0.5]
64 [0.25 -0.25 0.25 -0.25 0.75 -0.25 0.25 0.25]
65 [0.5 0. 0.5 0. 0.5 0. 0. 0.5]
66 [0.5 -0.5 0. 0. 0.5 0.5 0. 0.]
67 [0.5 0. 0. -0.5 0.5 0.5 0. 0.]
68 [0.25 -0.25 0.25 -0.25 0.75 0.25 0.25 -0.25]
69 [0.5 0. 0.5 0. 0.5 0.5 0. 0.]
70 [0.25 -0.25 0.25 -0.25 0.75 0.25 -0.25 0.25]
Number of combinations: 70

Codeblock:

```
from qiskit import *
import matplotlib.pyplot as plt
from qiskit.circuit import classical_function,  Int1
from qiskit.visualization.counts_visualization import plot_histogram
from custom_gate4 import *
backend = Aer.get_backend('qasm_simulator')
print(qiskit.__version__)

def superposition(qc):
    qc.h(0)
    qc.h(1)

def encoder(qc):
    qc.cx(0,2)
    qc.ccx(0,1,2)
    qc.x(2)

def decoder(qc):
    encoder(qc) #self-inverse

def measure(qc):
    qc.measure([0],[0])
    qc.measure([1],[1])
    qc.measure([2],[2])

#Schaltung und Operator erzeugen
############################
qc = QuantumCircuit(3,3)
operator = CustomGate(oracle, 'F(x)')

#Encoder
############################
superposition(qc)
encoder(qc)
qc.barrier()

#Operation F(x)
############################
qc.append(operator, [0,1,2])

#Decoder
############################
qc.barrier()
decoder(qc)
superposition(qc)

#Registermessung
############################
qc.barrier()
measure(qc)

print(qc.draw())
job = execute(qc, backend, shots=10000)
print(job.result().get_counts(qc))

#########################################################################

from qiskit.extensions import UnitaryGate
import numpy as np
'''
oracle = np.array(  [
                    [1. 0. 0. 0. 0. 0. 0. 0.]
                    [0. 0. 0. 0. 0. 0. 1. 0.]
                    [0. 0. 1. 0. 0. 0. 0. 0.]
                    [0. 0. 0. 1. 0. 0. 0. 0.]
                    [0. 0. 0. 0. 1. 0. 0. 0.]
                    [0. 0. 0. 0. 0. 1. 0. 0.]
                    [0. 1. 0. 0. 0. 0. 0. 0.]
                    [0. 0. 0. 0. 0. 0. 0. 1.]
                    ]
'''

oracle = np.identity(8)
oracle[1][1]=0
oracle[6][6]=0
oracle[6][1]=1
oracle[1][6]=1

class CustomGate(UnitaryGate):
    def __init__(self, data, label):
        super().__init__(data, label=label)
```