Comm Lab3

Author: B10901076 陳柏宏

1. Deutsch-Jozsa Algorithm

(a)

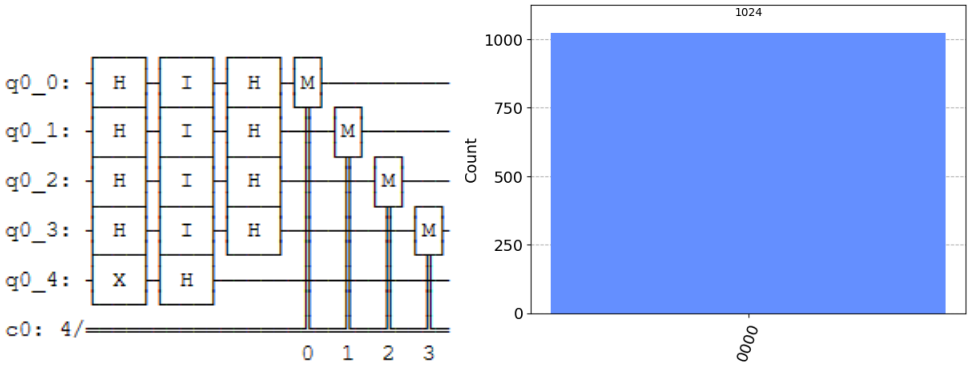
Oracle 1 / Oracle 2 / Oracle 3

x	f(x)	x	f(x)	x	f(x)
0000	0	0000	1	0000	0
0001	0	0001	1	0001	1
0010	0	0010	1	0010	1
0011	0	0011	1	0011	0
0100	0	0100	1	0100	1
0101	0	0101	1	0101	0
0110	0	0110	1	0110	0
0111	0	0111	1	0111	1
1000	0	1000	1	1000	1
1001	0	1001	1	1001	0
1010	0	1010	1	1010	0
1011	0	1011	1	1011	1
1100	0	1100	1	1100	0
1101	0	1101	1	1101	1
1110	0	1110	1	1110	1
1111	0	1111	1	1111	0

We can see that oracle 1 and 2 are indeed constant, and oracle 3 is balanced.

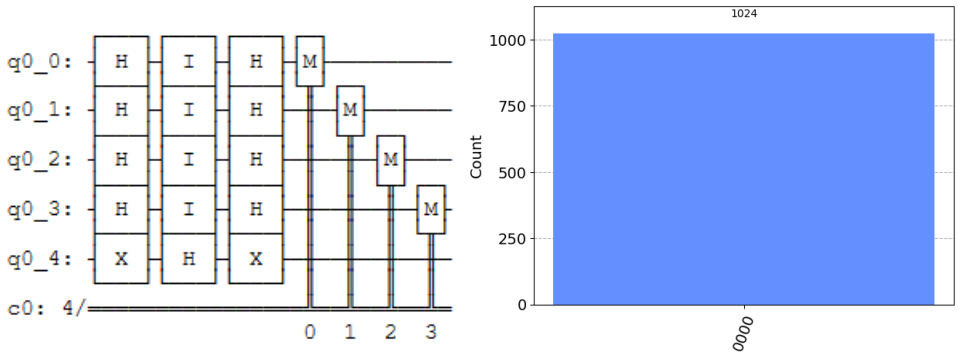
(b)

Oracle 1:



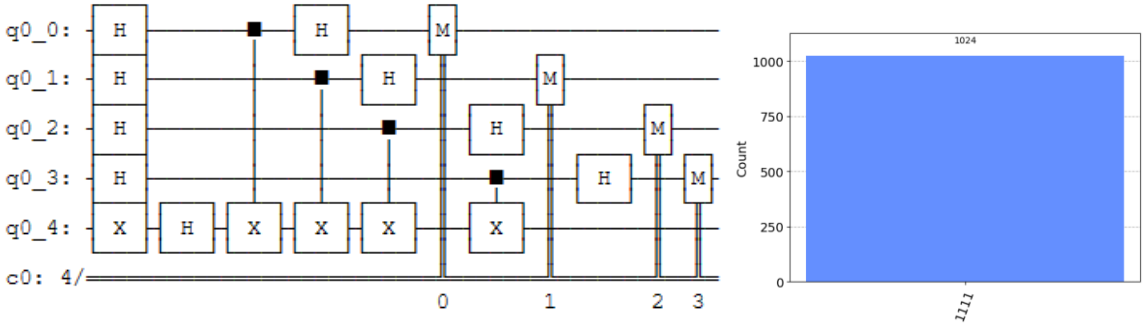
Since all results are 0000, thus it is constant.

Oracle 2:



Since all results are 0000, thus it is constant.

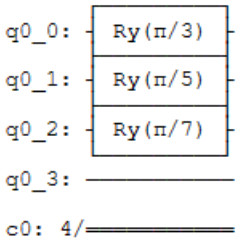
Oracle 3:



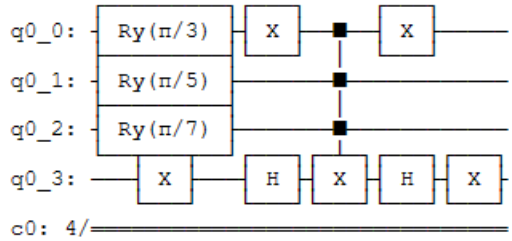
Since none of the results are 0000, it is balanced.

2. Grover's search

(a)



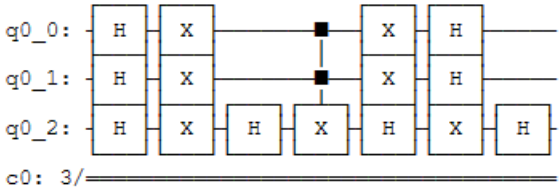
The first statevector: [0.80298875+0.j, 0.46360577+0.j, 0.26090686+0.j,
0.15063465+0.j, 0.18327694+0.j, 0.10581499+0.j,
0.05955029+0.j, 0.03438138+0.j, 0. +0.j,
0. +0.j, 0. +0.j, 0. +0.j,
0. +0.j, 0. +0.j, 0. +0.j,
0. +0.j]



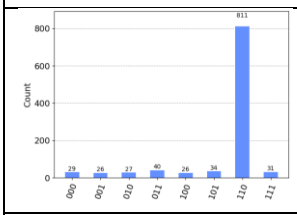
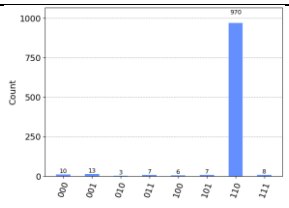
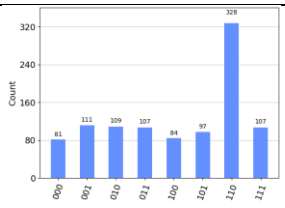
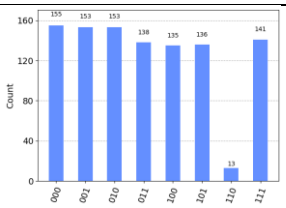
The second statevector: [8.02988752e-01+4.91688802e-17j,
4.63605772e-01+2.83876662e-17j,
2.60906861e-01+1.59759376e-17j,
1.50634647e-01+9.22371189e-18j,
1.83276943e-01+1.12224761e-17j,
1.05814992e-01+6.47929957e-18j,
-5.95502885e-02-3.64640351e-18j,
3.43813751e-02+2.10525205e-18j,
2.65293038e-17-4.91688802e-17j,
7.94526005e-18-2.83876662e-17j,
1.03796933e-17-1.59759376e-17j,
3.15210512e-19-9.22371189e-18j,
-2.71976820e-18-1.12224761e-17j,
-1.46198164e-18-6.47929957e-18j,
5.59293268e-19-3.64640351e-18j,
1.07454256e-18-2.10525205e-18j]
≈ [8.02988752e-01+0j,
4.63605772e-01+0j,
2.60906861e-01+0j,
1.50634647e-01+0j,
1.83276943e-01+0j,
1.05814992e-01+0j,
-5.95502885e-02-0j,
3.43813751e-02+0j,
0-0j, 0-0j, 0-0j, 0-0j, 0-0j, 0-0j, 0-0j, 0-0j]

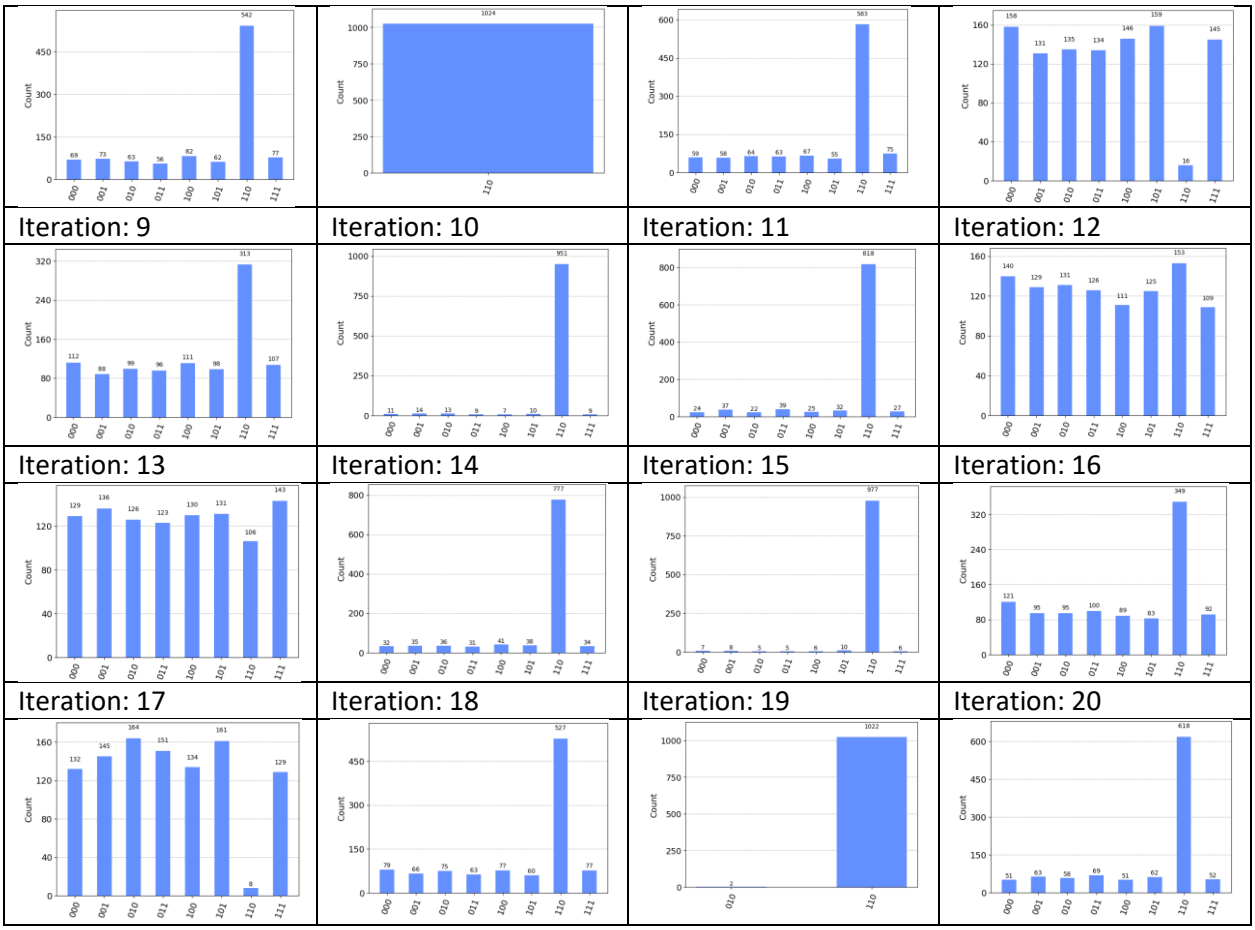
We can see that only the coefficient of 011 (since qiskit handles bits in reverse order, it corresponds to 110 which is the 7th element of the statevector) has its sign flipped.

(b)



(c)

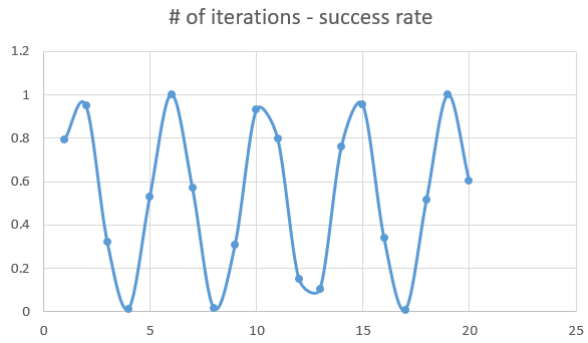
Iteration: 1	Iteration: 2	Iteration: 3	Iteration: 4
			
Iteration: 5	Iteration: 6	Iteration: 7	Iteration: 8



Iteration: 9 | Iteration: 10 | Iteration: 11 | Iteration: 12

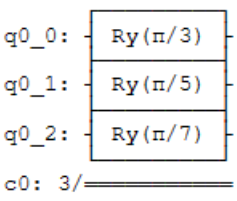
Iteration: 13 | Iteration: 14 | Iteration: 15 | Iteration: 16

Iteration: 17 | Iteration: 18 | Iteration: 19 | Iteration: 20



After iterating 2 times, the success rate is approximately 97%, meaning that there is still a chance of not getting the correct outcome 011.

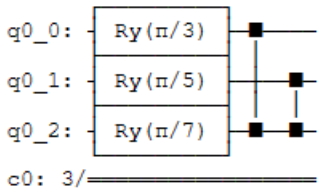
(d)



The first statevector: [0.80298875+0.j, 0.46360577+0.j, 0.26090686+0.j,

0.15063465+0.j, 0.18327694+0.j, 0.10581499+0.j,

0.05955029+0.j, 0.03438138+0.j]



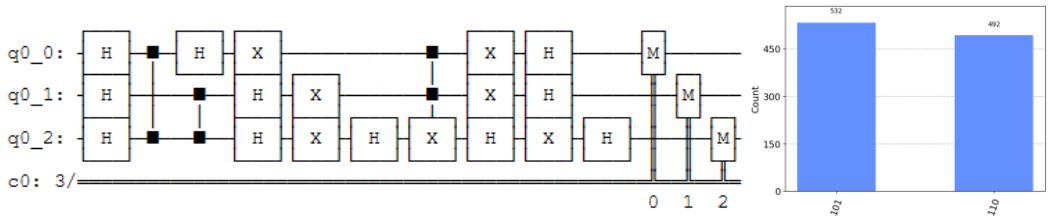
The second statevector: [0.80298875+0.j, 0.46360577+0.j, 0.26090686+0.j,

0.15063465+0.j, 0.18327694+0.j, -0.10581499+0.j,

-0.05955029+0.j, 0.03438138-0.j]

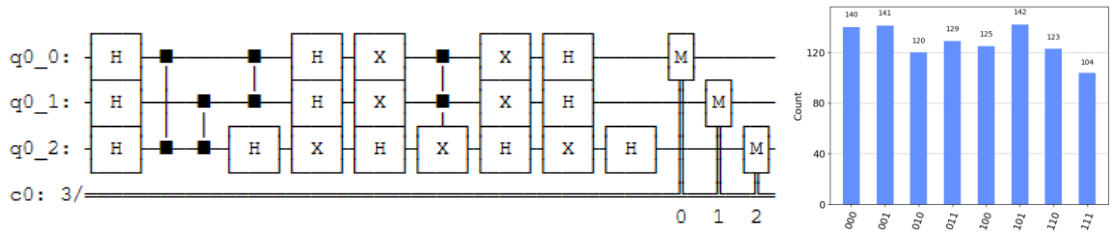
We can see that the coefficients of 011 and 101 have their sign flipped.

(e)



The outcomes are 101 and 011(reverse order of 110 due to qiskit rules), which means that it succeeded in 1 iteration.

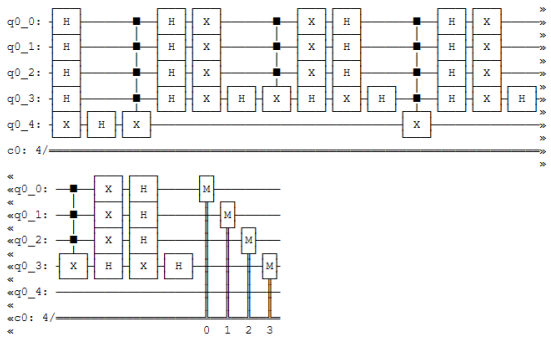
(f)

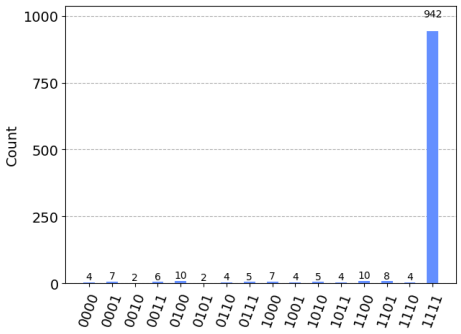


After more iterations, the probability distribution is still similar to that of one iteration, showing no sign of convergence. This is probably due the sampling problem with integers not being able to reach the points with high success probability.

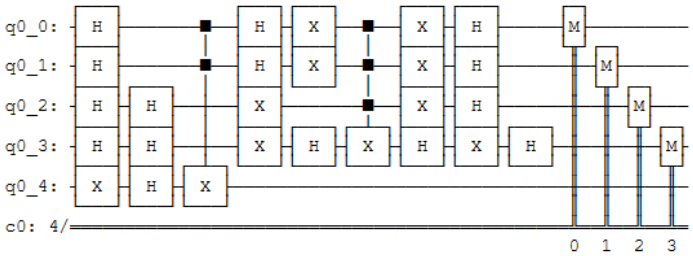
(bonus)

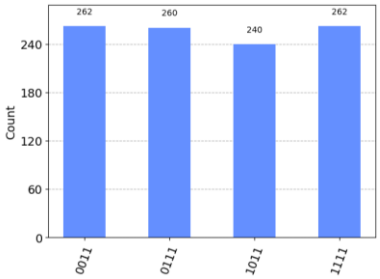
(1) Key set to 1111





Success probability reaches 94% in two iterations.





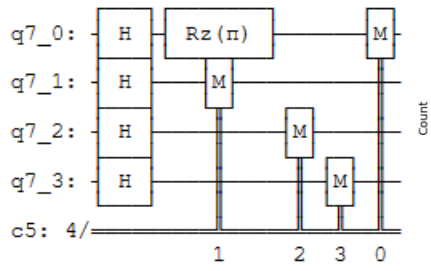
Success probability reaches 100% in one iteration. The result is always correct.

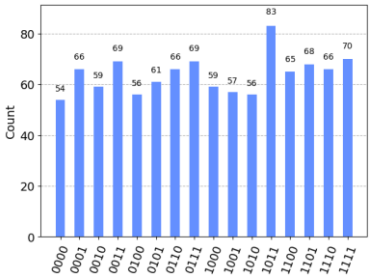
I believe that one of the challenges is to produce working oracles in large scale or designing efficient oracles with chosen keys.

3. QFT

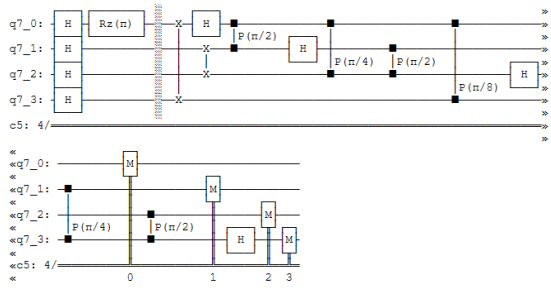
(1)

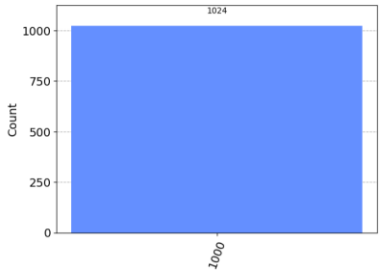
Before QFT:



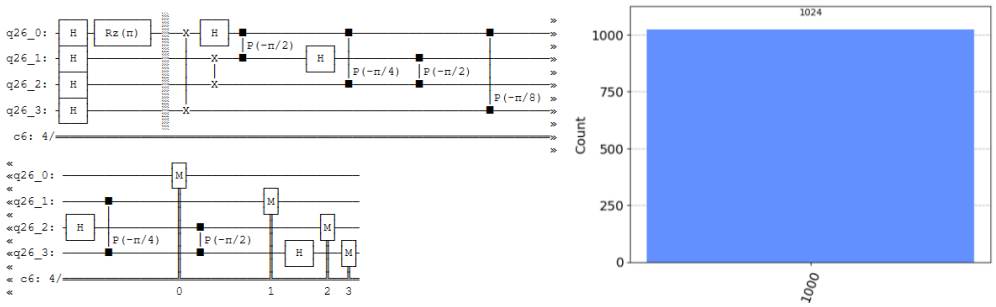


After QFT:



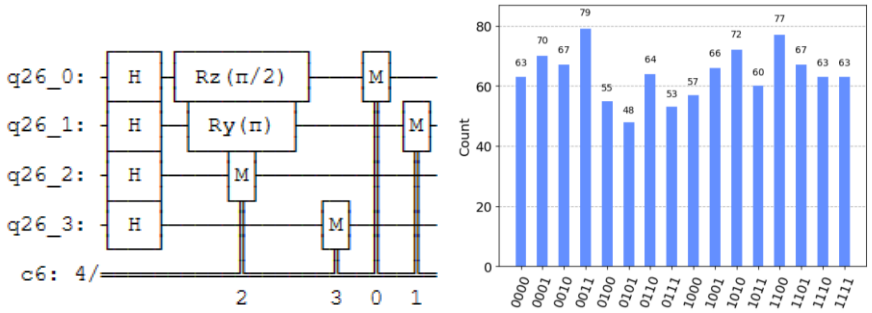


After QFT_dagger:

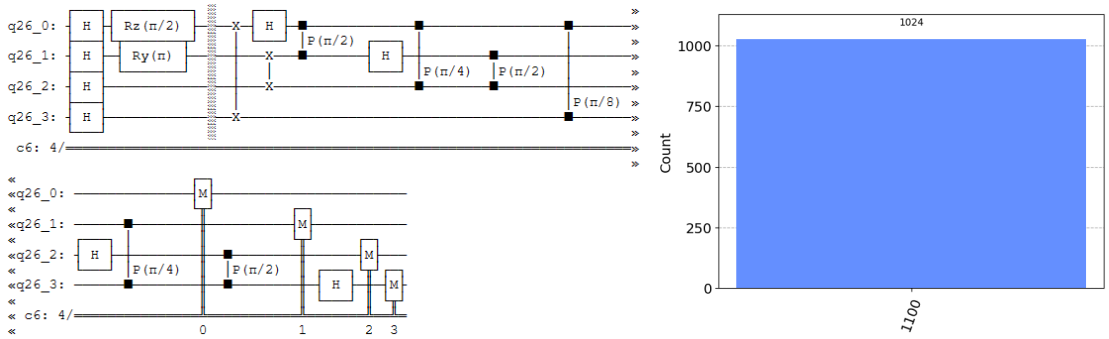


(2)

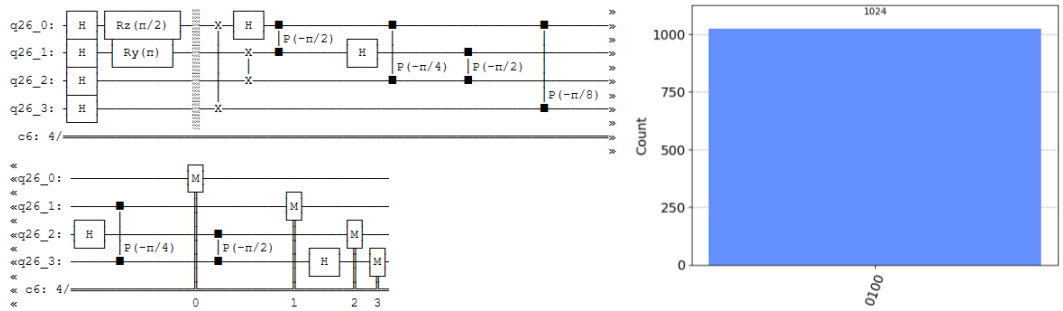
Before QFT:



After QFT:

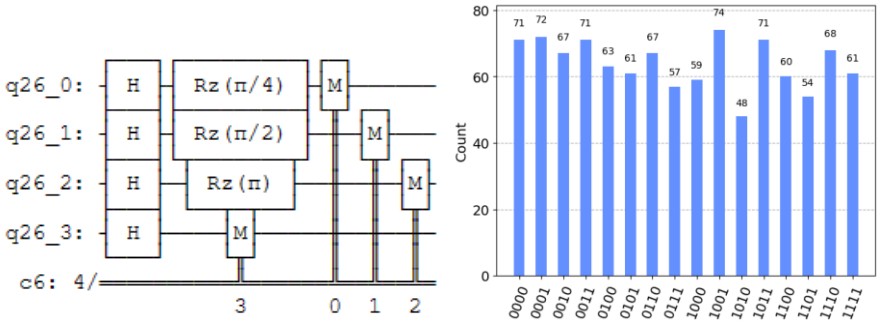


After QFT dagger:

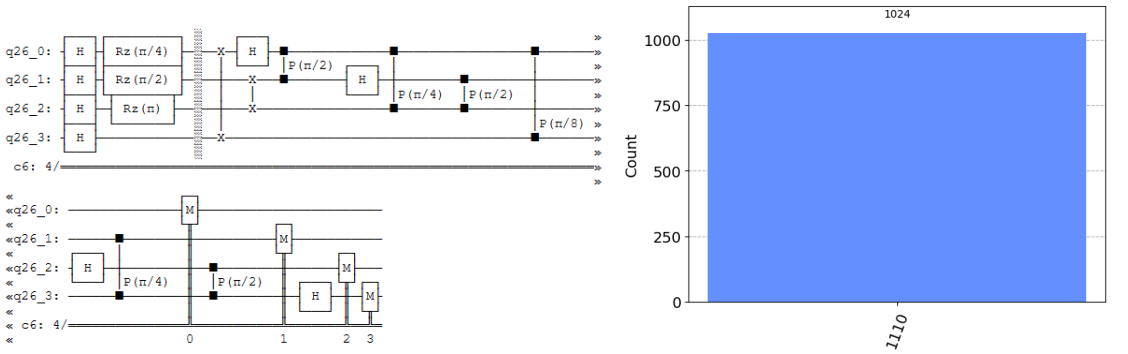


(3)

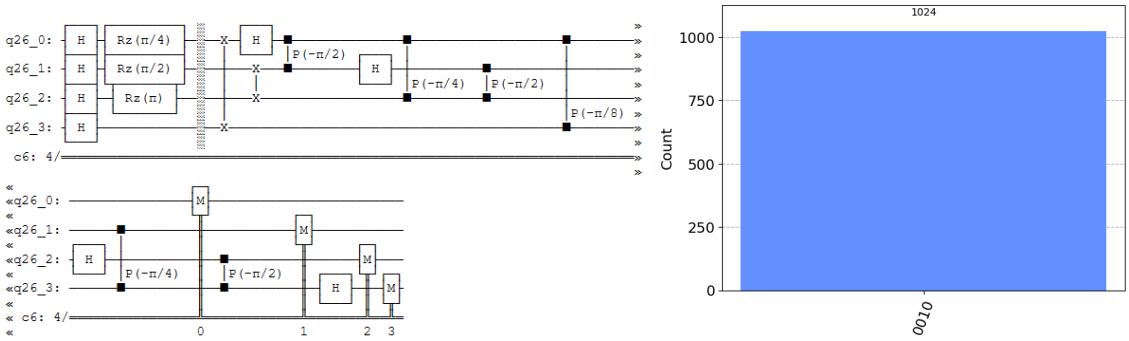
Before QFT:



After QFT:



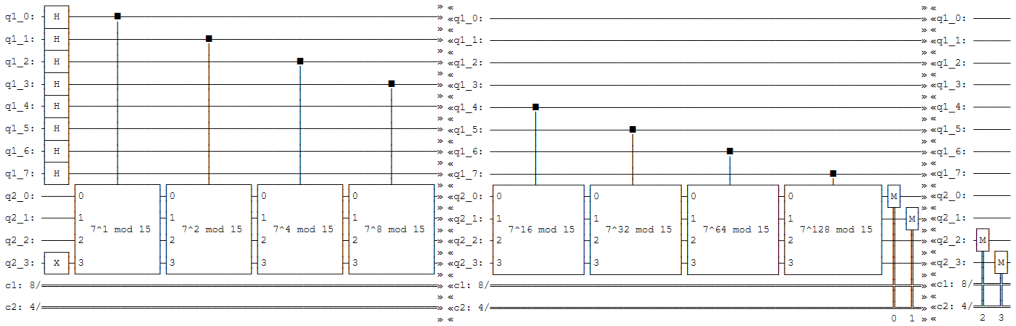
After QFT dagger:

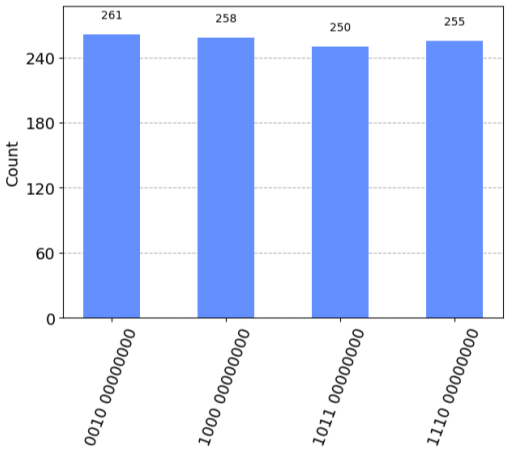


QFT transforms the computational basis into the fourier basis, while QFT dagger does the opposite.

4. Period Finding

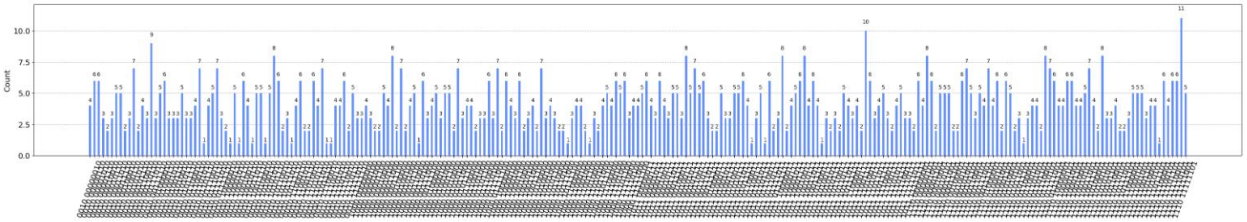
(Step 2) Measure Second Register





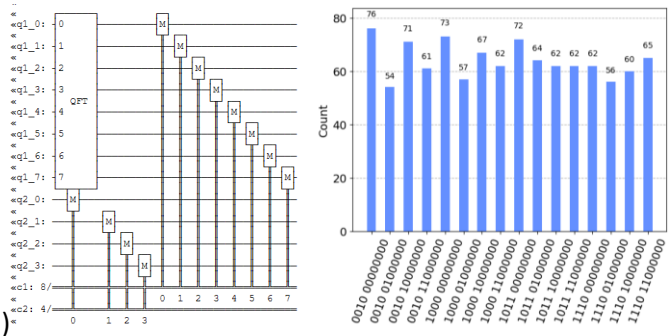
In decimal form, the measured result of the second register is either 1, 4, 7, or 13 (with similar probability). This is because 7^x=1, 4, 7, 13 (mod 15).

(Step 3) Measure both registers before applying QFT dagger



(Step 4) Measure both registers after applying QFT dagger

(following same circuit as above)



The corresponding c regardless of x0 are 0, 2^6, 2^7 or $3*2^6$ with approximately equal distribution, which makes sense because $2^8/4 = 2^6$.

(Step 5)

	Phase	Fraction	Guess for r
0	0.75	3/4	4
1	0.00	0/1	1
2	0.50	1/2	2
3	0.25	1/4	4

By the results above, we can see that the success rate is 50%, since both $c=2^6$ and $3*2^6$ can lead to k_0 being coprime to r.

(try a=2) success rate 50%

	Phase	Fraction	Guess for r
0	0.25	1/4	4
1	0.50	1/2	2
2	0.75	3/4	4
3	0.00	0/1	1

(try a=8) success rate 50%

	Phase	Fraction	Guess for r
0	0.25	1/4	4
1	0.50	1/2	2
2	0.00	0/1	1
3	0.75	3/4	4

(try a=11) success rate 50%

	Phase	Fraction	Guess for r
0	0.0	0/1	1
1	0.5	1/2	2

(try a=13) success rate 50%

	Phase	Fraction	Guess for r
0	0.00	0/1	1
1	0.25	1/4	4
2	0.50	1/2	2
3	0.75	3/4	4