Chapter 5

Order-Finding and Factoring

The inverse quantum Fourier transform and the quantum Fourier transform are the quantum circuits of implementing the Fourier transform and they can be applied to solve a variety of interesting questions. In this chapter, we now introduce two of the most interesting of those questions that are respectively the *order-finding problem* and the *factoring problem*. Miller in 1976 proved that solving the order-finding problem is equivalent to solve the factoring problem. For the RSA public-key cryptosystem, People have currently installed more than 400,000,000 copies of its algorithms and it is the primary cryptosystem used for security on the Internet and World Wide Web. The security of the RSA public-key cryptosystem is dependent on that the problem of factoring a big nature number into the production of two large prime numbers is intractable on a classical computer.

Shor's order-finding algorithm can solve the problems of order-finding and factoring exponential faster than any conventional computer. By means of using Shor's algorithm to factor a big nature number with 1024 bits into the production of two prime numbers with 512 bits each, Imre and Ferenc in [Imre and Ferenc 2005] indicate that the execution time is approximately 0.01 second. This is to say that Shor's algorithm will make the RSA public-key cryptosystem obsolete once its reliable physical implementation becomes available on the market. In this chapter, we first introduce a little background in number theory. Next, we explain how the order-finding problem implies the ability to factor as well. We also explain how shor's algorithm solves the order-finding problem. Next, we describe how to write quantum algorithms to implement Shor's algorithm for solving the simplest case in the problems of order-finding and factoring.

5.1 Introduction to Fundamental of Number Theory

We denote the set $\boldsymbol{Z}$ of *integers* is $\{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$. We may often refer to the set of *non-negative integers* to be $\{0, 1, 2, 3, \ldots\}$ and the set of *positive integers* to be $\{1, 2, 3, \ldots\}$. This is to say that 0 (zero) is one element in the set of *non-negative integers* and is not one element in the set of *positive integers*. We may occasionally say natural numbers that mean *positive integers* and the set of natural numbers is to $\{1, 2, 3, \ldots\}$.

More formally, given any positive integers w and n, we represent uniquely w in the following form

$$w = q \times n + r, \tag{5.1}$$

where q is a non-negative integer that is the quotient (result) of dividing w by n and the *remainder* r lies in the range 0 to $(n - 1)$. If the value of the remainder r is equal to zero, then we say that in this case n is a *factor* or a *divisor* of w. Otherwise, n is not a factor of w. Notice that 1 and w are always factors of w. Modular arithmetic is simply ordinary arithmetic in which we just pay attention to remainders. We make use of the notation (mod N) to point out that we are working in modular arithmetic, with respect to the positive integer N. For example, because 1, 3, 5, 7, 9 and 11 all have the same remainder (1) when divided by 2, we write $1 = 3 = 5 = 7 = 9 = 11 \pmod 2$.

The *greatest common divisor* or *factor* of integers, c and d, is the largest integer which is a divisor or a factor of both c and d. We write the number as gcd(c, d). For instance, the greatest common divisor or factor of 14 and 10 is 2. An easy method of obtaining this number is to enumerate the positive divisors of 14 (1, 2, 7, 14) and 10 (1, 2, 5, 10), and then pick out the largest common element in the two lists that is equal to two (2). Integers, c and d, are said to be *co-prime* if their greatest common divisor is 1. Because the greatest common divisor of 3 and 5 is 1, we say that 3 and 5 are co-prime. A *prime number* is an integer greater than 1, which has only itself and 1 as factors. If a number is an integer greater than 1 and it is not a prime number, then we call it as a composite number. The first few prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23, …. The most important single fact about the positive integers perhaps is that we may represent uniquely them as a product of factors, which are prime numbers. Let b be any integer greater than 1. Then b has a *prime factorization* of the following form

$$b = p_1^{b_1} \times p_2^{b_2} \times \ldots \times p_n^{b_n}, \tag{5.2}$$

where $p_1, p_2, \ldots, p_n$ are distinct prime numbers, and $b_1, b_2, \ldots, b_n$ are positive integers. For small numbers, finding the prime factorization by trial and error is very easy. For example, for a small number 10, its prime factorization is $10 = 2^1 \times 5^1$. Though huge effort aimed at finding one method that can efficiently determine the prime factorization of large numbers, there is no efficient method known in a digital computer to complete the task.

5.2 Description to Euclid's Algorithm

Euclid's algorithm is a much more efficient method of computing the greatest common divisor. Figure 5.1 is the flowchart of Euclid's algorithm. We use an example to explain how Euclid's algorithm works out the greatest common divisor. The example is to determine the greatest common divisor of two positive integers $c = 15$ and $d = 12$. From the first execution of Statement S_1 in Figure 5.1, it obtains the quotient $q = 15 / 12 = 1$. Next, from the first execution of Statement S_2 in Figure 5.1, it gets the remainder $r = 15 \pmod{12} = 3$. Because the value of r is not equal to zero, on the first execution of Statement S_3 in Figure 5.1, it returns to a *false*. Therefore, next, from the first execution of Statement S_6 in Figure 5.1, it gains the new value of the dividend $c = 12$. Similarly, from the first execution of Statement S_7 in Figure 5.1, it acquires the new value of the divisor $d = 3$.

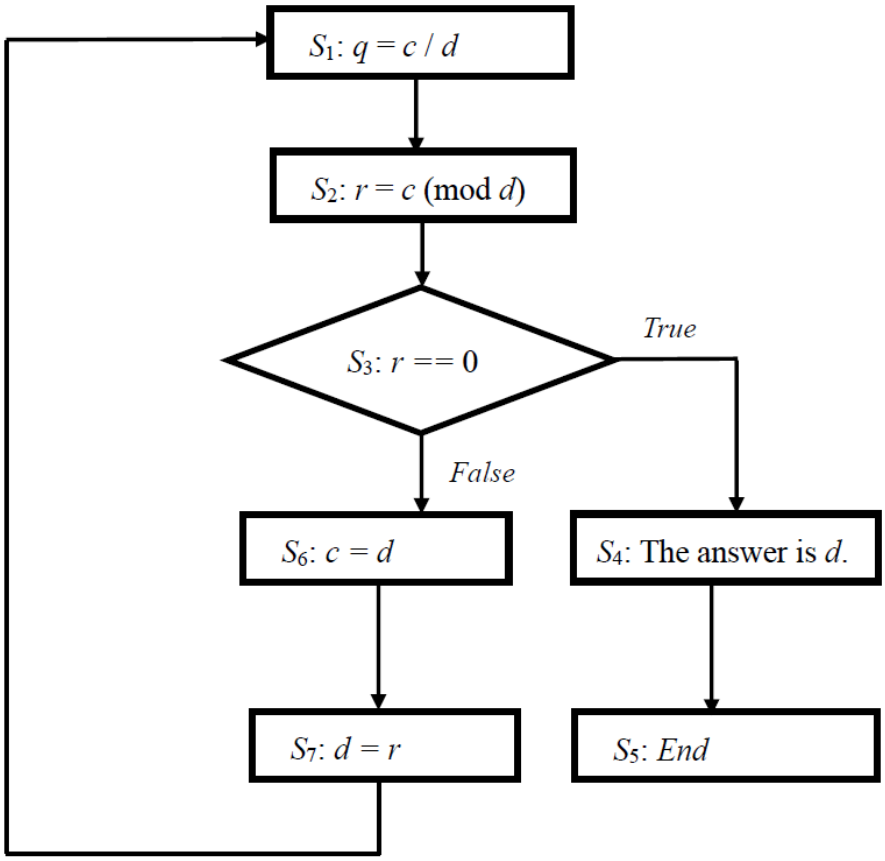

Figure 5.1: The flowchart of Euclid's algorithm.

Next, from the second execution of Statement S_1 in Figure 5.1, it obtains the quotient $q = 12 / 3 = 4$. From the second execution of Statement S_2 in Figure 5.1, it gets the remainder $r = 12 \pmod{3} = 0$. Since the value of r is equal to zero, on the second

execution of Statement S_3 in Figure 5.1, it returns to a *true*. Next, from the first execution of Statement S_4 in Figure 5.1, it gives that the answer is 3. This is to say that the greatest common divisor of 15 and 12 is 3. Next, from the first execution of Statement S_5 in Figure 5.1, it terminates the execution of Euclid's algorithm.

What resources does Euclid's algorithm consume? We assume that two positive integers, c and d, may be represented as bit strings of at most L bits each. This implies that none of the quotient q and the remainder r can be more than L bits long. Therefore, we may suppose that using L bit arithmetic does each computation. From Figure 5.1, the divide-and-remainder operation is the heart of Euclid's algorithm. At most using the divide-and-remainder operation of $O(L)$ times completes Euclid's algorithm. Because each divide-and-remainder operation requires $O(L^2)$ operations, the total cost of Euclid's algorithm is $O(L^3)$.

5.3 Illustration to Quadratic Congruence

We assume that N is a composite number with n bits. 1 and N are two trivial factors of N itself. We also suppose that there is a function β that is $\{X|0 \le X \le N\} \rightarrow \{X^2 \pmod N\}$. The domain of the function β is $\{X|0 \le X \le N\}$ and its range is $\{X^2 \pmod N\}$. If there is an integer $0 \le X \le N$ such that $\beta(X) = X^2 = C \pmod N$, i.e., the congruence has a solution, then C is said to be a quadratic congruence (mod N). Quadratic congruence (mod N) is a NP-complete problem in [Manders and Adleman 1978]. If the value of C is equal to one, then four integer solutions for $X^2 = 1 \pmod N$ are, respectively, b, $N - b$, 1 and $N - 1$, where $1 < b < (N / 2)$ and $(N / 2) < N - b < N - 1$. 1 and $N - 1$ are trivial solutions and b and $N - b$ are non-trivial solutions. This is a special case of quadratic congruence (mod N) and it is still a NP-complete problem. **Lemma 5-1** is used to show that we can determine a factor of N if we can find a non-trivial solution $X \neq \pm 1 \pmod N$ to the equation $X^2 = 1 \pmod N$.

Lemma 5-1: We assume that N is a composite number with n bits, and X is a non-trivial solution to the equation $X^2 = 1 \pmod N$ in the range $0 \le X \le N$, that is, neither $X = 1 \pmod N$ nor $X = N - 1 = -1 \pmod N$. Then at least one of $\gcd(N, X - 1)$ and $\gcd(N, X + 1)$ is a non-trivial factor of N that can be determined using Euclid's algorithm with $O(n^3)$ operations.

Proof:

Because $X^2 = 1 \pmod N$, it must be that N divides $X^2 - 1 = (X + 1) \times (X - 1)$. Since

$X \neq 1$ and $X \neq N - 1$, it must be that N does not divide $(X + 1)$ and does not divide $(X - 1)$. This is to say that N must have a common factor with one or the other of $(X + 1)$ and $(X - 1)$ and $1 < X < N - 1$. Therefore, we obtain $X - 1 < X + 1 < N$. From the condition $X - 1 < X + 1 < N$, we know that the common factor cannot be N itself. Applying Euclid's algorithm with $O(n^3)$ operations we may figure out gcd(N, $X - 1$) and gcd(N, $X + 1$) and therefore gain a non-trivial factor of N. ■

We consider one example in which N is equal to 15 and any given function β that is $\{X|0 \leq X \leq 15\} \rightarrow \{X^2 \pmod{15}\}$. The domain of the given function β is $\{X|0 \leq X \leq 15\}$ and its range is $\{X^2 \pmod{15}\}$. Sixteen outputs of $\beta(X)$ from the first input zero through the last input fifteen are subsequently 0, 1, 4, 9, 1, 10, 6, 4, 4, 6, 10, 1, 9, 4, 1 and 0. Four inputs 1, 4, 11 and 14 to satisfy $X^2 = 1 \pmod{15}$. Therefore, four integer solutions for $X^2 = 1 \pmod{15}$ are, respectively, 4, 11, 1 and 14. 1 and 14 are trivial solutions. 4 and 11 are non-trivial solutions. Because $4^2 = 1 \pmod{15}$ and $11^2 = 1 \pmod{15}$, it must be that 15 divides $4^2 - 1 = (4 + 1) \times (4 - 1)$ and $11^2 - 1 = (11 + 1) \times (11 - 1)$. Hence, 15 must have a common factor with one or the other of $(4 + 1)$ and $(4 - 1)$ and 15 must have a common factor with one or the other of $(11 + 1)$ and $(11 - 1)$. This is to say that using Euclid's algorithm we may figure out gcd(15, 5) = 5 and gcd(15, 3) = 3 or gcd(15, 12) = 3 and gcd(15, 10) = 5. This means that 15 has a prime factorization that is to $15 = 5 \times 3$.

5.4 Introduction to Continued Fractions

Between the continuum of real numbers and integers, there are many valuable connections. The theory of *continued fractions* is one such beautiful connection. If c and d are integers, then we call (c / d) as the *rational fraction* or *rational number*. A *finite simple continued fraction* is denoted by a finite collection $q[1], q[2], q[3], \ldots, q[i]$ of positive integers,

$$(q[1], q[2], q[3], \ldots, q[i]) = q[1] + \frac{1}{q[2] + \frac{1}{q[3] + \cdots + \frac{1}{q[i]}}}. \qquad (5.3)$$

We denote the kth convergent $(1 \leq k \leq i)$ to this continued fraction to be

$$(q[1], q[2], q[3], \ldots, q[k]) = q[1] + \frac{1}{q[2] + \frac{1}{q[3] + \cdots + \frac{1}{q[k]}}}. \qquad (5.4)$$

Figure 5.2 is the flowchart of the continued fractional algorithm. We can use the

continued fractional algorithm to determine a finite collection $q[1]$, $q[2]$, $q[3]$, …, $q[i]$ of positive integers in (5.3) for representing continued fraction of a *rational fraction*, (c / d). Therefore, applying the right-hand side equivalence in (5.3) we can describe c / d as

$$\frac{c}{d} = q[1] + \frac{1}{q[2]+\frac{1}{q[3]+\cdots+\frac{1}{q[i]}}}. \tag{5.5}$$

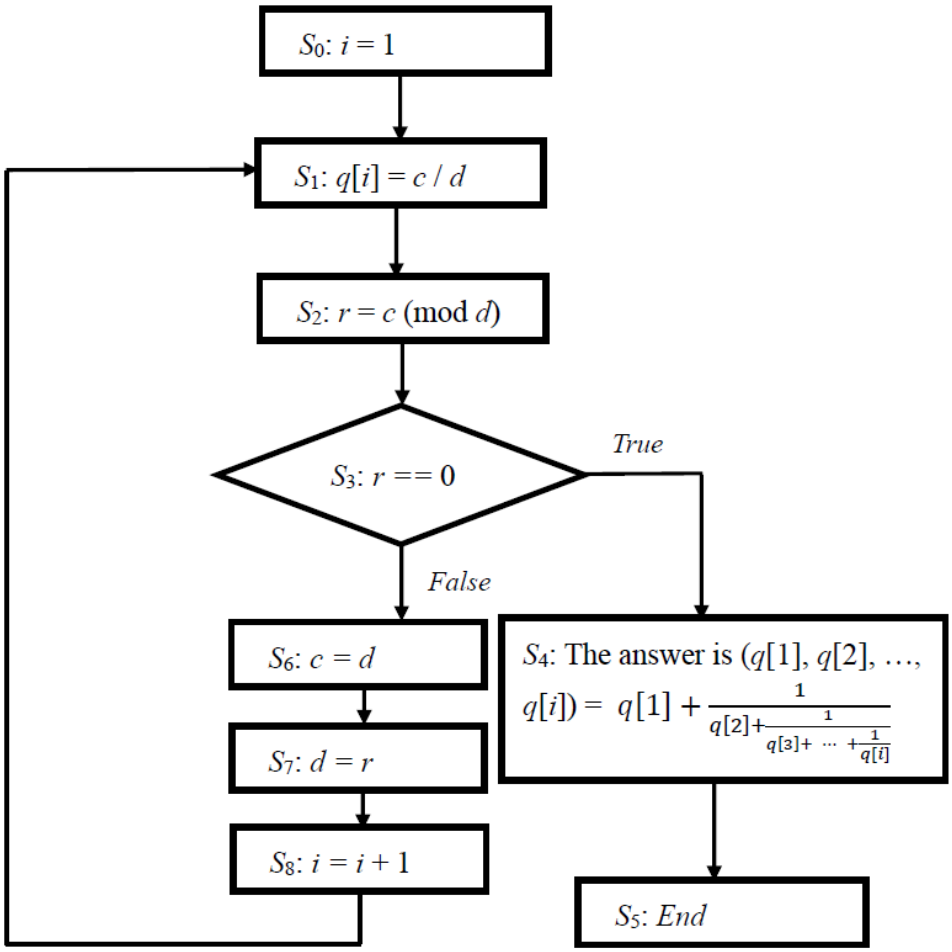

Figure 5.2: The flowchart of the continued fractional algorithm.

We make use of one example to explain how the continued fractional algorithm in Figure 5.2 determines the continued fractional representation of (c / d) if c = 31 and d = 13 and the corresponding convergent. From the first execution of statement S_0 through statement S_2, it obtains i = 1, $q[1] = c / d$ = 31 / 13 = 2 and r = 31 / (mod 13) = 5. This is to split (31 / 13) into its integer and fractional part and to invert its fractional part,

$$\frac{31}{13} = 2 + \frac{5}{13} = 2 + \frac{1}{\frac{13}{5}}. \tag{5.6}$$

Because the value of r is equal to 5, from the first execution of statement S_3, it returns to a *false*. Thus, next, from the first execution of statement S_6 through statement S_8, it gets that the new value of the *numerator* c is equal to 13, the new value of the *denominator* d is equal to 5 and the value of index variable i is equal to 2.

Next, from the second execution of statement S_1 through statement S_2, it obtains $q[2] = c / d = 13 / 5 = 2$ and $r = 13 / (\text{mod } 5) = 3$. These steps – split then invert – are now used to (13 / 5), giving

$$\frac{31}{13} = 2 + \frac{1}{2+\frac{3}{5}} = 2 + \frac{1}{2+\frac{1}{\frac{5}{3}}}. \tag{5.7}$$

Since the value of r is equal to 3, from the second execution of statement S_3, it returns to a *false*. Hence, next, from the second execution of statement S_6 through statement S_8, it gets that the new value of the numerator c is equal to 5, the new value of the denominator d is equal to 3 and the value of index variable i is equal to 3.

Next, from the third execution of statement S_1 through statement S_2, it gains $q[3] = c / d = 5 / 3 = 1$ and $r = 5 / (\text{mod } 3) = 2$. These steps – split then invert – are now used to (5 / 3), giving

$$\frac{31}{13} = 2 + \frac{1}{2+\frac{1}{1+\frac{2}{3}}} = 2 + \frac{1}{2+\frac{1}{1+\frac{1}{\frac{3}{2}}}}. \tag{5.8}$$

Because the value of r is equal to 2, from the third execution of statement S_3, it returns to a *false*. Hence, next, from the third execution of statement S_6 through statement S_8, it obtains that the new value of the numerator c is equal to 3, the new value of the denominator d is equal to 2 and the value of index variable i is equal to 4.

Next, from the fourth execution of statement S_1 through statement S_2, it acquires $q[4] = c / d = 3 / 2 = 1$ and $r = 3 / (\text{mod } 2) = 1$. These steps – split then invert – are now used to (3 / 2), giving

$$\frac{31}{13} = 2 + \frac{1}{2+\frac{1}{1+\frac{1}{1+\frac{1}{2}}}} = 2 + \frac{1}{2+\frac{1}{1+\frac{1}{1+\frac{1}{\frac{2}{1}}}}}. \tag{5.9}$$

Since the value of r is equal to 1, from the fourth execution of statement S_3, it returns to a *false*. Thus, next, from the fourth execution of statement S_6 through statement S_8, it obtains that the new value of the numerator c is equal to 2, the new value of the denominator d is equal to 1 and the value of index variable i is equal to 5.

Next, from the fifth execution of statement S_1 through statement S_2, it acquires $q[5] = c / d = 2 / 1 = 2$ and $r = 2 / (\text{mod } 1) = 0$. Because the value of r is equal to zero, this is to split (2 / 1) into its integer and fractional part and *not* to invert its fractional part. This means that $(2 / 1) = 2 + \frac{0}{1} = 2$. Therefore, these steps – split then *no* invert – are now used to (2 / 1), giving

$$\frac{31}{13} = 2 + \cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{2+\frac{0}{1}}}}} = 2 + \cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\frac{1}{2}}}}. \tag{5.10}$$

Because the value of r is equal to 0, from the fifth execution of statement S_3, it returns to a *true*. Therefore, next, from the first execution of statement S_4, the answer is to the continued fractional representation of (31 / 13)

$$\frac{31}{13} = (q[1] = 2, q[2] = 2, q[3] = 1, q[4] = 1, q[5] = 2) = 2 + \cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\frac{1}{2}}}}. \tag{5.11}$$

Next, from the first execution of Statement S_5, it terminates the execution of the continued fractional algorithm. For a rational number (31 / 13), the first convergent through the fifth convergent are subsequently $(q[1]) = 2$, $(q[1], q[2]) = 2 + \frac{1}{2} = \frac{5}{2}$, $(q[1], q[2], q[3]) = 2 + \frac{1}{2+\frac{1}{1}} = \frac{7}{3}$, $(q[1], q[2], q[3], q[4]) = 2 + \cfrac{1}{2+\cfrac{1}{1+\frac{1}{1}}} = \frac{12}{5}$ and $(q[1], q[2], q[3], q[4], q[5]) = 2 + \cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{1+\frac{1}{2}}}} = \frac{31}{13}$.

What resources does the continued fractional algorithm in Figure 5.2 consume for obtaining a continued fractional expansion to a rational number $c / d > 1$, where c and d are integers of L bits? This is to say that how many values of $q[k]$ for $1 \le k \le i$ in (5.3)

must be determined from the continued fractional algorithm in Figure 5.2. From statement S_1 and statement S_2 in the continued fractional algorithm of Figure 5.2, each quotient $q[k]$ for $1 \le k \le i$ is at most L bits long, and the remainder r is at most L bits long. Thus, we may assume that making use of L bit arithmetic does each computation. From Figure 5.2, the divide-and-remainder operation is the heart of the continued fractional algorithm. At most applying the divide-and-remainder operation of $O(L)$ times completes the continued fractional algorithm. Because each divide-and-remainder operation requires $O(L^2)$ operations, the total cost of the continued fractional algorithm is $O(L^3)$.

5.5 Order-finding and Factoring

We assume that N is a positive integer, the greatest common divisor of X and N is one for $1 \le X \le N$ and X is co-prime to N. The *order* of X modulo N is to the *least* positive integer r such that $X^r = 1 \pmod N$. The *ordering-finding problem* is to compute r, given X and N. There is no efficient algorithm on a classical computer to solve the ordering-finding problem and the problem of factoring numbers. However, solving the problem of factoring numbers is equivalent to solve the ordering-finding problem. This is to say that if there is one efficient algorithm to solve the ordering-finding problem on a quantum computer, then it can solve the problem of factoring numbers quickly. We use **Lemma 5-2** to demonstrate that we can determine a factor of N if we can find the *order* r of X modulo N to satisfy $X^r = 1 \pmod N$ and to be even such that a non-trivial solution $X^{\frac{r}{2}} \neq \pm 1 \pmod N$ to the equation $X^r = (X^{\frac{r}{2}})^2 = 1 \pmod N$. Simultaneously, we use **Lemma 5-3** to show representation theorem for the greatest common divisor of two integers c and d.

Lemma 5-2: We suppose that N is a composite number with n bits, and the *order* r of X modulo N satisfies $X^r = 1 \pmod N$ and is even such that a non-trivial solution $X^{\frac{r}{2}} \neq \pm 1 \pmod N$ to the equation $X^r = (X^{\frac{r}{2}})^2 = 1 \pmod N$ in the range $0 \le X^{\frac{r}{2}} \le N$. That is that neither $X^{\frac{r}{2}} = 1 \pmod N$ nor $X^{\frac{r}{2}} = N - 1 = -1 \pmod N$. Then at least one of $\gcd(N, X^{\frac{r}{2}} - 1)$ and $\gcd(N, X^{\frac{r}{2}} + 1)$ is a non-trivial factor of N that can be determined using Euclid's algorithm with $O(n^3)$ operations.

Proof:

Since $X^r = (X^{\frac{r}{2}})^2 = 1 \pmod N$, it must be that N divides $(X^{\frac{r}{2}})^2 - 1 = (X^{\frac{r}{2}} + 1) \times (X^{\frac{r}{2}} - 1)$. Because $X^{\frac{r}{2}} \neq 1$ and $X^{\frac{r}{2}} \neq N - 1$, it must be that N does not divide $(X^{\frac{r}{2}} + 1)$ and does not divide $(X^{\frac{r}{2}} - 1)$. This means that N must have a common factor with one or the other of $(X^{\frac{r}{2}} + 1)$ and $(X^{\frac{r}{2}} - 1)$ and $1 < X^{\frac{r}{2}} < N - 1$. Thus, we get $X^{\frac{r}{2}} - 1 < X^{\frac{r}{2}} + 1 < N$. From the condition $X^{\frac{r}{2}} - 1 < X^{\frac{r}{2}} + 1 < N$, we see that the common factor cannot be N itself. Using Euclid's algorithm with $O(n^3)$ operations we may compute $\gcd(N, X^{\frac{r}{2}} - 1)$ and $\gcd(N, X^{\frac{r}{2}} + 1)$ and hence obtain a non-trivial factor of N. ■

Lemma 5-3: The greatest common divisor of two integers c and d is the least positive integer that can be written in the form $c \times u + d \times v$, where u and v are integers.

Proof:

Let $t = c \times u + d \times v$ be the smallest positive integer written in this form. Let w is the greatest common divisor of c and d. Therefore, w is a divisor to both c and d and it is a divisor of t. This means that $w \leq t$. For completing the proof, we show $t \leq w$ by demonstrating that t is a divisor of both c and d. The proof is by contradiction. We assume that t is not a divisor of c. Then $c = q \times t + r$, where the remainder r is in the range 1 to $t - 1$. Rearranging this equation $c = q \times t + r$ and using $t = c \times u + d \times v$, we obtain $r = c \times (1 - q \times u) + d \times (-q \times v)$ that is a positive integer that is a linear combination of c and d. Because r is smaller than t, this contradicts the definition of t as the smallest positive integer written in a linear combination of c and d. Therefore, we infer that t must divide c. Similarly, by symmetry t must also be a divisor of d. This means that $t \leq w$ and $w \leq t$. Therefore, we complete the proof. ■

Lemma 5-4: We assume that integer a divides both integer c and integer d. Then a divides the greatest common divisor of both c and d.

Proof:

From **Lemma 5-3**, the greatest common divisor of c and d is $c \times u + d \times v$, where u and v are integers. Because a divides both c and d, it must also divide $c \times u + d \times v$. Therefore, we at once infer that a divides the greatest common divisor of both c and d. ■

When does a number, c, have a multiplicative inverse in modular arithmetic? This is to ask, given c and N, when does there exist a d such that $c \times d = 1 \pmod{N}$? We consider one example in which $3 \times 4 = 1 \pmod{11}$. This gives that the number 3 has a multiplicative inverse 4 in arithmetic modulo 11. On the other hand, trial and error explains that 3 has no multiplicative inverse modulo 6. Determining multiplicative inverse in modular arithmetic is actually related to the greatest common divisor by the notion *co-primality*: integer c and integer d are *co-prime* if their greatest common divisor is 1 (one). For example, 3 and 11 are co-prime, because the positive divisors of 3 is 1 and 3 and the positive divisors of 11 is 1 and 11. We use **Lemma 5-5** to characterize the existence of multiplicative inverse in modular arithmetic applying co-primality. Simultaneously, we use **Lemma 5-6** to show that the order r of X modulo N satisfies $r \leq N$.

Lemma 5-5: Let N be an integer that is greater than 1. An integer X has a multiplicative inverse modulo N if and only if the greatest common divisor of X and N is 1.

Proof:

We use $\gcd(X, N)$ to represent the greatest common divisor of X and N. We assume that X has a multiplicative inverse, which we define X^{-1}, modulo N. Then $X \times X^{-1} = 1 \pmod{N}$. This gives that $X \times X^{-1} = u \times N + 1$ for some integer u, and hence $X \times X^{-1} + (-u) \times N = 1$. From **Lemma 5-3**, we obtain $\gcd(X, N) = X \times X^{-1} + (-u) \times N = 1$. Therefore, we at once conclude that $\gcd(X, N) = 1$.

Conversely, if $\gcd(X, N) = 1$, then from **Lemma 5-3** there must exist integer y and integer z such that $X \times y + z \times N = 1$. After applying the modular operation to both sides we obtain $X \times y \pmod{N} + z \times N \pmod{N} = 1 \pmod{N}$. As $z \times N \pmod{N} = 0$, we have then that $X \times y \pmod{N} = 1 \pmod{N}$. This means that the remainder of $X \times y$ modulo N is equal to one. Therefore, we obtain $X \times y = 1 \pmod{N}$. So, $y = X^{-1}$ is the multiplicative inverse of X. ■

Lemma 5-6: Let N be an integer greater than 1 and $1 \leq X \leq N$. X and N are co-prime and r is the *least* positive integer such that $X^r = 1 \pmod{N}$. Then $r \leq N$.

Proof:

Consider a sequence of different order values for X: $X^0 \pmod N$, $X^1 \pmod N$, $X^2 \pmod N$, …, $X^{N-1} \pmod N$, $X^N \pmod N$. Under the modular operation there can only be N unique values 0, 1, 2, …, $N - 1$ for $X^i \pmod N$ in the above sequence. If there are more items in the above sequence than N, some $X^i \pmod N$ will have the same value when the modular operation is applied (there are $N + 1$ items in the above sequence). For example, let $N = 5$ and $X = 2$. Then $X^1 = 2 \pmod 5$ and $X^5 = 2 \pmod 5$.

Hence, among the different items in the above sequence, there are two items that are equivalent under the modular operation, $X^n = X^m \pmod N$, where we can assume, without loss of generality, that $n > m$ and $n, m \leq N$. Since from **Lemma 5-5** $\gcd(X, N) = 1$, we know that there exists a multiplicative inverse X^{-1} of X such that $X \times X^{-1} = 1 \pmod N$. Because the greatest common divisor of X and N is one, the greatest common divisor of X^m and N is equal to one. From **Lemma 5-5**, $\gcd(X^m, N) = 1$, we know that there exists a multiplicative inverse X^{-m} of X^m such that $X^m \times X^{-m} = 1 \pmod N$. Next, multiply both sides of the modular operation, $X^n = X^m \pmod N$, by X^{-m} to obtain $X^n \times X^{-m} = X^m \times X^{-m} \pmod N$ and $X^{n-m} = X^{m-m} \pmod N = X^0 \pmod N = 1 \pmod N$. From the statements above we have that $r = n - m$. Furthermore, as $n, m \leq N$ and $n > m$, it follows that $r = n - m \leq N$. ■

5.6 Compute the Order of 2 Modulo 15 and the Prime Factorization for 15

We would like to find the prime factorization for $N = 15$. We need to search for the nontrivial factor for $N = 15$. From **Lemma 5-1** and **Lemma 5-6**, we select a number $X = 2$ so that the greatest common divisor of $X = 2$ and $N = 15$ is 1 (one). This is to say that $X = 2$ is co-prime to $N = 15$. From **Lemma 5-6,** the order r of 2 modulo 15 satisfies $r \leq 15$. Because the number of bit representing $N = 15$ is four bits long, we also only need to use four bits that represent the value of r.

Determining the order r of 2 modulo 15 is equivalent to determine the period r of a given oracular function P_f: $\{r_1\ r_2\ r_3\ r_4 \mid \forall\ r_d \in \{0, 1\} \text{ for } 1 \leq d \leq 4\} \rightarrow \{2^{r_1 r_2 r_3 r_4} \pmod{15} \mid \forall\ r_d \in \{0, 1\} \text{ for } 1 \leq d \leq 4\}$. The period r of P_f is to satisfy $P_f(r_1\ r_2\ r_3\ r_4) = P_f(r_1\ r_2\ r_3\ r_4 + r)$ to any two inputs $(r_1\ r_2\ r_3\ r_4)$ and $(r_1\ r_2\ r_3\ r_4 + r)$. Sixteen outputs of P_f that takes each input from ${r_1}^0\ {r_2}^0\ {r_3}^0\ {r_4}^0$ through ${r_1}^1\ {r_2}^1\ {r_3}^1\ {r_4}^1$ are subsequently 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4 and 8. The frequency f of P_f is equal to the number of the period

per sixteen outputs and is equal to four. The period r of P_f is the *reciprocal* of the frequency f of P_f. Thus, we obtain $r = \frac{1}{\frac{f}{16}} = \frac{16}{f} = \frac{16}{4} = 4$ and $r \times f = 4 \times 4 = 16$.

On the other hand, we think of the input domain of P_f as the time domain and its output as signals. Computing the order r of 2 modulo 15 is equivalent to determine the period r and the frequency f of signals in the time domain (the input domain). Because the output of each input from $r_1^0\ r_2^0\ r_3^0\ r_4^0$ through $r_1^1\ r_2^1\ r_3^1\ r_4^1$ is subsequently 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4 and 8, we take the sixteen input values as the corresponding sixteen time units and the sixteen outputs as the sixteen samples of signals. Each sample encodes an output of P_f. The output can take 1, 2, 4 or 8. The sixteen input values from $r_1^0\ r_2^0\ r_3^0\ r_4^0$ through $r_1^1\ r_2^1\ r_3^1\ r_4^1$ corresponds to sixteen time units from zero through fifteen.

We use Figure 5.3 to explain the reason of why computing the order r of 2 modulo 15 is equivalent to determine the period r and the frequency f of signals in the time domain (the input domain). In Figure 5.3, the horizontal axis is to represent the time domain in which it contains the input domain of P_f and the vertical axis is to represent signals in which it consists of the sixteen outputs of P_f. For convenience of presentation, we use variable k to represent the decimal value of each binary input and make use of $2^k \bmod 15$ to represent $2^{r_1 r_2 r_3 r_4} \pmod{15}$.

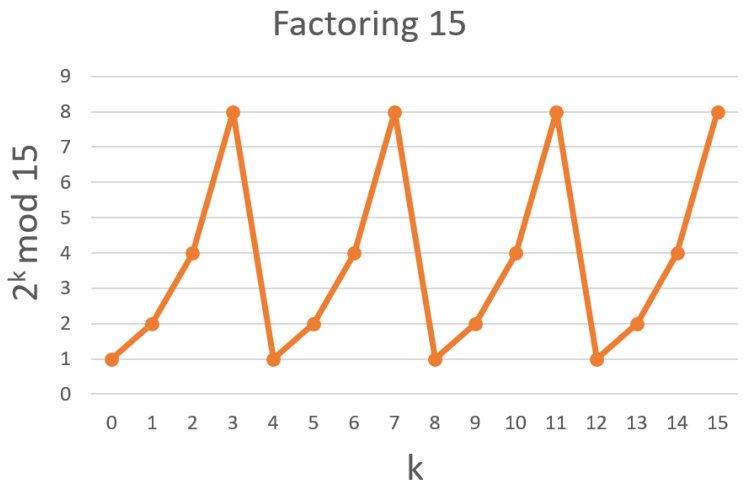

Figure 5.3: Sampling sixteen points from sixteen outputs of a given oracular function that is P_f: $\{r_1\ r_2\ r_3\ r_4 \mid \forall\ r_d \in \{0, 1\}$ for $1 \le d \le 4\} \rightarrow \{2^{r_1 r_2 r_3 r_4} \pmod{15} \mid \forall\ r_d \in \{0, 1\}$ for $1 \le d \le 4\}$.

From Figure 5.3, hidden patterns and information stored in a given oracular function

P_f are that the signal rotates back to its *first* signal (output with 1) *four* times. Its signal rotates back to its *second* signal (output with 2) *four* times. Its signal rotates back to its *third* signal (output with 4) *four* times and its signal rotates back to its *fourth* signal (output with 8) *four* times. This indicates that there are four periods of signals per sixteen time units and the frequency f of signals is equal to four.

Because in Figure 5.3 the period r of signals is the *reciprocal* of the frequency f of signals, the period r of signals is $\frac{1}{\frac{4}{16}}$ = 16 / 4 = 4. The period r = 4 of signals in Figure 5.3 satisfies $P_f(r_1^0\ r_2^1\ r_3^0\ r_4^0) = 2^{r_1^0 r_2^1 r_3^0 r_4^0} \pmod{15} = 2^4 \pmod{15} = 16 \pmod{15} = 1 \pmod{15}$, so the period r = 4 of signals in P_f is equivalent to the order r = 4 of 2 modulo 15. The cost to find the order r = 4 of 2 modulo 15 is to implement sixteen (2^4) operations of modular exponentiation, $2^{r_1 r_2 r_3 r_4} \pmod{15}$. Since r = 4 is even and is less than 15, from **Lemma 5-2**, we use Euclid's algorithm to compute gcd(15, $2^{\frac{4}{2}} + 1$) and gcd(15, $2^{\frac{4}{2}} - 1$). This is to say that two nontrivial factors for N = 15 are respectively 5 and 3. Therefore, the prime factorization for N = 15 is $N = 5 \times 3$.

Because ($\frac{1}{\frac{4}{16}}$ = 16 / 4) is a rational number and is an integer, we make use of the continued fractional algorithm in Figure 5.2 to determine the continued fractional representation of (c / d) if c = 16 and d = 4 and the corresponding convergent for explaining how the continued fractional algorithm works out in real applications. From the first execution of statement S_0 through statement S_2, it gets i = 1, $q[1] = c / d$ = 16 / 4 = 4 and r = 16 (mod 4) = 0. This is to split (16 / 4) into its integer and fractional part and not to invert its fractional part,

$$\frac{16}{4} = 4 + \frac{0}{4} = 4. \tag{5.12}$$

Because the value of r is equal to 0, from the first execution of statement S_3, it returns to a *true*. Thus, next, from the first execution of statement S_4, the answer is to the continued fractional representation of (16 / 4)

$$\frac{16}{4} = (q[1] = 4) = 4. \tag{5.13}$$

Next, from the first execution of Statement S_5, it terminates the execution of the continued fractional algorithm. For a rational number (16 / 4), the first convergent is (q[1]) = 4 = $\frac{4}{1}$ that is the closest to ($\frac{1}{\frac{4}{16}} = \frac{16}{4}$) and is actually equal to $\frac{16}{4}$. This means that the first convergent (q[1]) = 4 = $\frac{4}{1}$ is equal to the period $r = \frac{r}{1}$. Hence, we obtain that the period r is equal to the numerator 4 of the first convergent. Because the numerator r = 4 of the first convergent is less than N= 15, the numerator r = 4 is equivalent to that the order r = 4 of 2 modulo 15 satisfies $2^4 = 1 \pmod{15}$.

5.7 Determine the Order of 2 Modulo 21 and the Prime Factorization for 21

We want to search for the prime factorization for N = 21. We need to find the nontrivial factor for N = 21. From **Lemma 5-1** and **Lemma 5-6**, we select a number X = 2 so that the greatest common divisor of X = 2 and N = 21 is 1 (one). This indicates that X = 2 is co-prime to N = 21. From **Lemma 5-6,** the order r of 2 modulo 21 satisfies $r \leq 21$. The number of bit representing N = 21 is five bits long, so we only need to make use of five bits that encode the value of r.

Computing the order r of 2 modulo 21 is equivalent to figure out the period r of a given oracular function A_f: $\{r_1\ r_2\ r_3\ r_4\ r_5 \mid \forall\ r_d \in \{0, 1\}$ for $1 \leq d \leq 5\} \rightarrow \{2^{r_1 r_2 r_3 r_4 r_5} \pmod{21} \mid \forall\ r_d \in \{0, 1\}$ for $1 \leq d \leq 5\}$. The period r of A_f is to satisfy $A_f(r_1\ r_2\ r_3\ r_4\ r_5) = A_f(r_1\ r_2\ r_3\ r_4\ r_5 + r)$ to any two inputs ($r_1\ r_2\ r_3\ r_4\ r_5$) and ($r_1\ r_2\ r_3\ r_4\ r_5 + r$). Thirty-two outputs of A_f that takes each input from $r_1^0\ r_2^0\ r_3^0\ r_4^0\ r_5^0$ through $r_1^1\ r_2^1\ r_3^1\ r_4^1\ r_5^1$ are subsequently 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, 11, 1 and 2. The frequency f of A_f is equal to the number of the period per thirty-two outputs. The period r of A_f is the *reciprocal* of the frequency f of A_f. Hence, we obtain $r = \frac{r}{1} = \frac{1}{\frac{f}{32}} = \frac{32}{f}$ and $r \times f = 32 \times 1 = 32$.

On the other hand, we think of the input domain of A_f as the time domain and its output as signals. Figuring out the order r of 2 modulo 21 is equivalent to compute the period r and the frequency f of signals in the time domain (the input domain). The output of each input from $r_1^0\ r_2^0\ r_3^0\ r_4^0\ r_5^0$ through $r_1^1\ r_2^1\ r_3^1\ r_4^1\ r_5^1$ is subsequently 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, 11, 1, 2, 4, 8, 16, 11, 1 and 2. Therefore, we take the thirty-two input values as the corresponding thirty-two time units and the thirty-two outputs as the thirty-two samples of signals. Each sample encodes an

output of A_f. The output can take 1, 2, 4, 8, 16 or 11. The thirty-two input values from $r_1^0\ r_2^0\ r_3^0\ r_4^0\ r_5^0$ through $r_1^1\ r_2^1\ r_3^1\ r_4^1\ r_5^1$ corresponds to thirty-two time units from zero through thirty-one.

We apply Figure 5.4 to explain the reason of why determining the order r of 2 modulo 21 is equivalent to determine the period r and the frequency f of signals in the time domain (the input domain). In Figure 5.4, the horizontal axis is to represent the time domain in which it contains the input domain of A_f and the vertical axis is to represent signals in which it consists of the thirty-two outputs of A_f. For convenience of presentation, we make use of variable k to represent the decimal value of each binary input and use $2^k \bmod 21$ to represent $2^{r_1 r_2 r_3 r_4 r_5} \pmod{21}$.

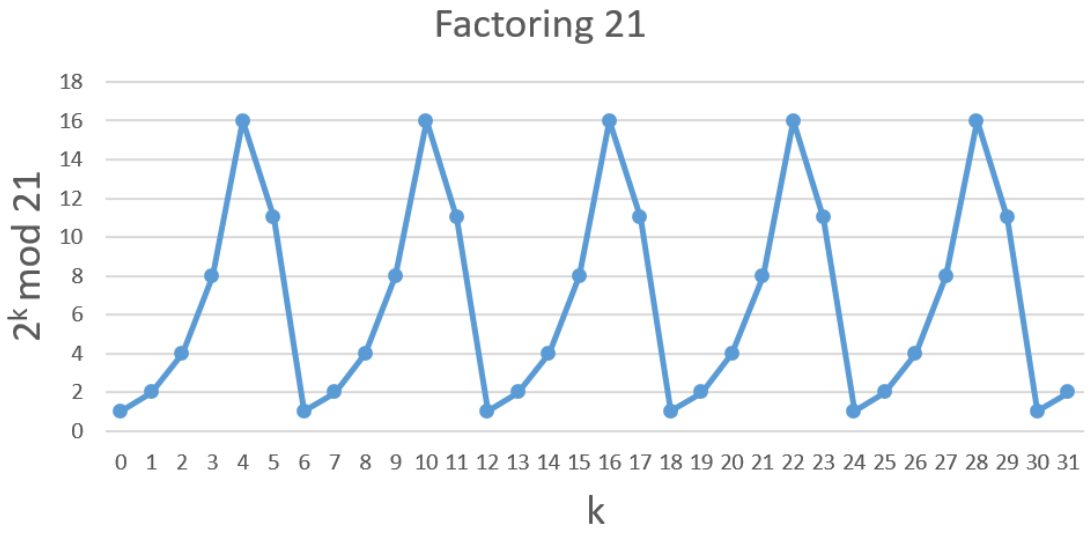

Figure 5.4: Sampling thirty-two points from thirty-two outputs of a given oracular function that is A_f: $\{r_1\ r_2\ r_3\ r_4\ r_5 \mid \forall\ r_d \in \{0, 1\}$ for $1 \le d \le 5\} \rightarrow \{2^{r_1 r_2 r_3 r_4 r_5} \pmod{21} \mid \forall\ r_d \in \{0, 1\}$ for $1 \le d \le 5\}$.

From Figure 5.4, hidden patterns and information stored in a given oracular function A_f are that the signal rotates back to its *first* signal (output with 1) *six* times. Its signal rotates back to its *second* signal (output with 2) *six* times. Its signal rotates back to its *third* signal (output with 4) *five* times and its signal rotates back to its *fourth* signal (output with 8) *five* times. Its signal rotates back to its *fifth* signal (output with 16) *five* times and its signal rotates back to its *sixth* signal (output with 11) *five* times. This is to say that there are $(5\frac{2}{6})$ periods of signals per thirty-two time units and the frequency f of signals is equal to $(5\frac{2}{6})$.

Since in Figure 5.4 the period r of signals is the *reciprocal* of the frequency f of signals, the period r of signals is $\frac{1}{\frac{5^{\frac{2}{6}}}{32}} = \frac{1}{\frac{32}{\frac{6}{32}}} = \frac{32}{\frac{32}{6}} = 6 / 1 = 6$. The period $r = 6$ of signals in Figure 5.4 satisfies $A_f(r_1^0\, r_2^0\, r_3^1\, r_4^1\, r_5^0) = 2^{r_1^0 r_2^0 r_3^1 r_4^1 r_5^0} \pmod{21} = 2^6 \pmod{21} = 64 \pmod{21} = 1 \pmod{21}$, so the period $r = 6$ of signals in A_f is equivalent to the order $r = 6$ of 2 modulo 21. The cost to find the order $r = 6$ of 2 modulo 21 is to implement thirty-two (2^5) operations of modular exponentiation, $2^{r_1 r_2 r_3 r_4 r_5} \pmod{21}$. Because $r = 6$ is even and is less than 21, from **Lemma 5-2**, we use Euclid's algorithm to compute gcd(21, $2^{\frac{6}{2}} + 1$) and gcd(21, $2^{\frac{6}{2}} - 1$). This implies that two nontrivial factors for $N = 21$ are respectively 3 and 7. Thus, the prime factorization for $N = 21$ is $N = 3 \times 7$.

Because $(\frac{1}{\frac{5^{\frac{2}{6}}}{32}} = \frac{1}{\frac{32}{\frac{6}{32}}} = \frac{32}{\frac{32}{6}} = 6 / 1)$ is a rational number and is an integer, we apply the continued fractional algorithm in Figure 5.2 to determine the continued fractional representation of (c / d) if $c = 6$ and $d = 1$ and the corresponding convergent for explaining how the continued fractional algorithm works out in real applications. From the first execution of statement S_0 through statement S_2, it gets $i = 1$, $q[1] = c / d = 6 / 1 = 6$ and $r = 6 \pmod 1 = 0$. This is to split (6 / 1) into its integer and fractional part and not to invert its fractional part,

$$\frac{6}{1} = 6 + \frac{0}{1} = 6. \tag{5.14}$$

Because the value of r is equal to 0, from the first execution of statement S_3, it returns to a *true*. Hence, next, from the first execution of statement S_4, the answer is to the continued fractional representation of (6 / 1)

$$\frac{6}{1} = (q[1] = 6) = 6. \tag{5.15}$$

Next, from the first execution of Statement S_5, it terminates the execution of the continued fractional algorithm. For a rational number (6 / 1), the first convergent is $(q[1]) = 6 = \frac{6}{1}$ that is the closest to $(\frac{1}{\frac{5^{\frac{2}{6}}}{32}} = \frac{1}{\frac{32}{\frac{6}{32}}} = \frac{32}{\frac{32}{6}} = \frac{6}{1})$ and is actually equal to $\frac{6}{1}$. This indicates that the first convergent $(q[1]) = 6 = \frac{6}{1}$ is equal to the period $r = \frac{r}{1}$. Thus,

we obtain that the period r is equal to the numerator 6 of the first convergent. Since the numerator $r = 6$ of the first convergent is less than $N = 21$, the numerator $r = 6$ is equivalent to that the order $r = 6$ of 2 modulo 21 satisfies $2^6 = 1 \pmod{21}$.

5.8 Calculate the Order of 2 Modulo 35 and the Prime Factorization for 35

We would like to find the prime factorization for $N = 35$. We need to search for the nontrivial factor for $N = 35$. From **Lemma 5-1** and **Lemma 5-6**, we select a number $X = 2$ so that the greatest common divisor of $X = 2$ and $N = 35$ is 1 (one). This implies that $X = 2$ is co-prime to $N = 35$. Because from **Lemma 5-6,** the order r of 2 modulo 35 satisfies $r \leq 35$. The number of bit representing $N = 35$ is six bits long, we only need to use six bits that encode the value of r.

Calculating the order r of 2 modulo 35 is equivalent to compute the period r of a given oracular function B_f: $\{r_1\, r_2\, r_3\, r_4\, r_5\, r_6 \mid \forall\, r_d \in \{0, 1\} \text{ for } 1 \leq d \leq 6\} \rightarrow \{2^{r_1 r_2 r_3 r_4 r_5 r_6} \pmod{35} \mid \forall\, r_d \in \{0, 1\} \text{ for } 1 \leq d \leq 6\}$. The period r of B_f is to satisfy $B_f(r_1\, r_2\, r_3\, r_4\, r_5\, r_6) = B_f(r_1\, r_2\, r_3\, r_4\, r_5\, r_6 + r)$ to any two inputs $(r_1\, r_2\, r_3\, r_4\, r_5\, r_6)$ and $(r_1\, r_2\, r_3\, r_4\, r_5\, r_6 + r)$. The front *twenty-four* outputs of B_f that takes each input from $r_1^0\, r_2^0\, r_3^0\, r_4^0\, r_5^0\, r_6^0$ through $r_1^1\, r_2^1\, r_3^1\, r_4^1\, r_5^1\, r_6^1$ are subsequently 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9, 18, 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9 and 18. The middle *twenty-four* outputs of B_f are respectively 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9, 18, 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9 and 18. The last *sixteen* outputs of B_f are subsequently 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9, 18, 1, 2, 4 and 8. The frequency f of B_f is equal to the number of the period per sixty-four outputs. The period r of B_f is the *reciprocal* of the frequency f of B_f. Therefore, we obtain $r = \frac{r}{1} = \frac{1}{\frac{f}{64}} = \frac{64}{f}$ and $r \times f = 64 \times 1 = 64$.

On the other hand, we think of the input domain of B_f as the time domain and its output as signals. Determining the order r of 2 modulo 35 is equivalent to figure out the period r and the frequency f of signals in the time domain (the input domain). The front *twenty-four* outputs of each input from $r_1^0\, r_2^0\, r_3^0\, r_4^0\, r_5^0\, r_6^0$ through $r_1^1\, r_2^1\, r_3^1\, r_4^1\, r_5^1\, r_6^1$ are subsequently 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9, 18, 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9 and 18. The middle *twenty-four* outputs are respectively 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9, 18, 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9 and 18. The last *sixteen* outputs are subsequently 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9, 18, 1, 2, 4 and 8. Hence, we take the sixty-four input values as the corresponding sixty-four time units and the sixty-four outputs as the sixty-four samples of signals. Each sample encodes an output of B_f. The

output can take 1, 2, 4, 8, 16, 32, 29, 23, 11, 22, 9 or 18. The sixty-four input values from $r_1^0\ r_2^0\ r_3^0\ r_4^0\ r_5^0\ r_6^0$ through $r_1^1\ r_2^1\ r_3^1\ r_4^1\ r_5^1\ r_6^1$ corresponds to sixty-four time units from zero through sixty-three.

We apply Figure 5.5 to show the reason of why computing the order r of 2 modulo 35 is equivalent to determine the period r and the frequency f of signals in the time domain (the input domain). In Figure 5.5, the horizontal axis is to represent the time domain in which it consists of the input domain of B_f and the vertical axis is to represent signals in which it includes the sixty-four outputs of B_f. For convenience of presentation, we use variable k to represent the decimal value of each binary input and use 2^k mod 35 to represent $2^{r_1 r_2 r_3 r_4 r_5 r_6} \pmod{35}$.

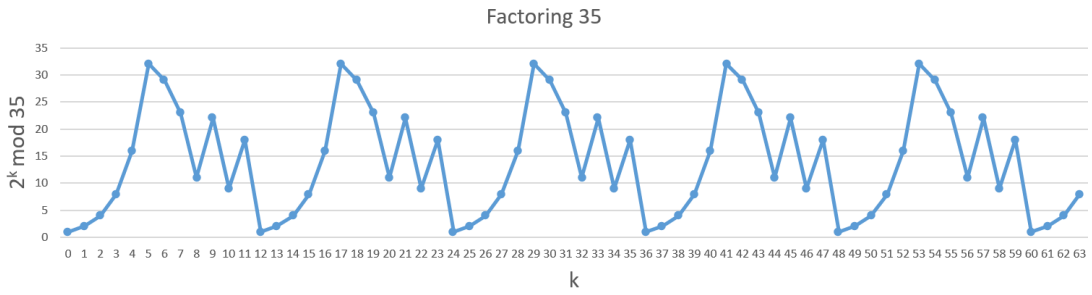

Figure 5.5: Sampling sixty-four points from sixty-four outputs of a given oracular function that is B_f: $\{r_1\ r_2\ r_3\ r_4\ r_5\ r_6 \mid \forall\ r_d \in \{0, 1\} \text{ for } 1 \le d \le 6\} \rightarrow \{2^{r_1 r_2 r_3 r_4 r_5 r_6} \pmod{35} \mid \forall\ r_d \in \{0, 1\} \text{ for } 1 \le d \le 6\}$.

From Figure 5.5, hidden patterns and information stored in a given oracular function B_f are that the signal rotates back to its *first* signal (output with 1) *six* times. Its signal rotates back to its *second* signal (output with 2) *six* times. Its signal rotates back to its *third* signal (output with 4) *six* times and its signal rotates back to its *fourth* signal (output with 8) *six* times. Its signal rotates back to its *fifth* signal (output with 16) *five* times and its signal rotates back to its *sixth* signal (output with 32) *five* times. Its signal rotates back to its *seventh* signal (output with 29) *five* times and its signal rotates back to its *eighth* signal (output with 23) *five* times. Its signal rotates back to its *ninth* signal (output with 11) *five* times and its signal rotates back to its *tenth* signal (output with 22) *five* times. Its signal rotates back to its *eleventh* signal (output with 9) *five* times and its signal rotates back to its *twelfth* signal (output with 18) *five* times. This indicates that there are $(5\frac{4}{12})$ periods of signals per sixty-four time units and the frequency f of signals is equal to $(5\frac{4}{12})$.

Since in Figure 5.5 the period r of signals is the *reciprocal* of the frequency f of signals, the period r of signals is $\frac{1}{5\frac{4}{\frac{12}{64}}} = \frac{1}{\frac{64}{\frac{12}{64}}} = \frac{64}{\frac{64}{12}}$ = 12 / 1 = 12. The period r = 12 of signals in Figure 5.5 satisfies $B_f(r_1^0\ r_2^0\ r_3^1\ r_4^1\ r_5^0\ r_6^0) = 2^{r_1^0 r_2^0 r_3^1 r_4^1 r_5^0 r_6^0} \pmod{35} = 2^{12} \pmod{35}$ = 4096 (mod 35) = 1 (mod 35), so the period r = 12 of signals in B_f is equivalent to the order r = 12 of 2 modulo 35. The cost to find the order r = 12 of 2 modulo 35 is to implement sixty-four (2^6) operations of modular exponentiation, $2^{r_1 r_2 r_3 r_4 r_5 r_6} \pmod{35}$. Because r = 12 is even and is less than 35, from **Lemma 5-2**, we use Euclid's algorithm to compute gcd(35, $2^{\frac{12}{2}} + 1$) and gcd(35, $2^{\frac{12}{2}} - 1$). This implies that two nontrivial factors for N = 35 are respectively 5 and 7. Thus, the prime factorization for N = 35 is $N = 5 \times 7$.

Because ($\frac{1}{5\frac{4}{\frac{12}{64}}} = \frac{1}{\frac{64}{\frac{12}{64}}} = \frac{64}{\frac{64}{12}}$ = 12 / 1) is a rational number and is an integer, we make use of the continued fractional algorithm in Figure 5.2 to determine the continued fractional representation of (c / d) if c = 12 and d = 1 and the corresponding convergent for explaining how the continued fractional algorithm works out in real applications. From the first execution of statement S_0 through statement S_2, it gets i = 1, $q[1] = c / d$ = 12 / 1 = 12 and r = 12 (mod 1) = 0. This is to split (12 / 1) into its integer and fractional part and not to invert its fractional part,

$$\frac{12}{1} = 12 + \frac{0}{1} = 12. \tag{5.16}$$

Since the value of r is equal to 0, from the first execution of statement S_3, it returns to a *true*. Thus, next, from the first execution of statement S_4, the answer is to the continued fractional representation of (12 / 1)

$$\frac{12}{1} = (q[1] = 12) = 12. \tag{5.17}$$

Next, from the first execution of Statement S_5, it terminates the execution of the continued fractional algorithm. For a rational number (12 / 1), the first convergent is $(q[1]) = 12 = \frac{12}{1}$ that is the closest to ($\frac{1}{5\frac{4}{\frac{12}{64}}} = \frac{1}{\frac{64}{\frac{12}{64}}} = \frac{64}{\frac{64}{12}} = \frac{12}{1}$) and is actually equal to $\frac{12}{1}$. This is to say that the first convergent $(q[1]) = 12 = \frac{12}{1}$ is equal to the period $r = \frac{r}{1}$.

Therefore, we obtain that the period r is equal to the numerator 12 of the first convergent. The numerator $r = 12$ of the first convergent is less than $N = 35$, so the numerator $r = 12$ is equivalent to that the order $r = 12$ of 2 modulo 35 satisfies $2^{12} = 1 \pmod{35}$.

5.9 Determine the Order of 5 Modulo 33 and the Prime Factorization for 33

We would like to search for the prime factorization for $N = 33$. We need to find the nontrivial factor for $N = 33$. From **Lemma 5-1** and **Lemma 5-6**, we select a number $X = 5$ so that the greatest common divisor of $X = 5$ and $N = 33$ is 1 (one). This is to say that $X = 5$ is co-prime to $N = 33$. Since from **Lemma 5-6,** the order r of 5 modulo 33 satisfies $r \leq 33$. The number of bit representing $N = 33$ is six bits long, we only need to use six bits that encode the value of r.

Computing the order r of 5 modulo 33 is equivalent to calculate the period r of a given oracular function C_f: $\{r_1\, r_2\, r_3\, r_4\, r_5\, r_6 \mid \forall\, r_d \in \{0, 1\} \text{ for } 1 \leq d \leq 6\} \rightarrow \{5^{r_1 r_2 r_3 r_4 r_5 r_6} \pmod{33} \mid \forall\, r_d \in \{0, 1\} \text{ for } 1 \leq d \leq 6\}$. The period r of C_f is to satisfy $C_f(r_1\, r_2\, r_3\, r_4\, r_5\, r_6) = C_f(r_1\, r_2\, r_3\, r_4\, r_5\, r_6 + r)$ to any two inputs $(r_1\, r_2\, r_3\, r_4\, r_5\, r_6)$ and $(r_1\, r_2\, r_3\, r_4\, r_5\, r_6 + r)$. The front *twenty* outputs of C_f that takes each input from $r_1^0\, r_2^0\, r_3^0\, r_4^0\, r_5^0\, r_6^0$ through $r_1^1\, r_2^1\, r_3^1\, r_4^1\, r_5^1\, r_6^1$ are subsequently 1, 5, 25, 26, 31, 23, 16, 14, 4, 20, 1, 5, 25, 26, 31, 23, 16, 14, 4 and 20. The middle *twenty* outputs of C_f are respectively 1, 5, 25, 26, 31, 23, 16, 14, 4, 20, 1, 5, 25, 26, 31, 23, 16, 14, 4 and 20. The last *twenty-four* outputs of C_f are subsequently 1, 5, 25, 26, 31, 23, 16, 14, 4, 20, 1, 5, 25, 26, 31, 23, 16, 14, 4, 20, 1, 5, 25 and 26. The frequency f of C_f is equal to the number of the period per sixty-four outputs. The period r of C_f is the *reciprocal* of the frequency f of C_f. Hence, we get $r = \frac{r}{1} = \frac{1}{\frac{f}{64}} = \frac{64}{f}$ and $r \times f = 64 \times 1 = 64$.

On the other hand, we think of the input domain of C_f as the time domain and its output as signals. Calculating the order r of 5 modulo 33 is equivalent to determine the period r and the frequency f of signals in the time domain (the input domain). The front *twenty* outputs of each input from $r_1^0\, r_2^0\, r_3^0\, r_4^0\, r_5^0\, r_6^0$ through $r_1^1\, r_2^1\, r_3^1\, r_4^1\, r_5^1\, r_6^1$ are subsequently 1, 5, 25, 26, 31, 23, 16, 14, 4, 20, 1, 5, 25, 26, 31, 23, 16, 14, 4 and 20. The middle *twenty* outputs are respectively 1, 5, 25, 26, 31, 23, 16, 14, 4, 20, 1, 5, 25, 26, 31, 23, 16, 14, 4 and 20. The last *twenty-four* outputs are subsequently 1, 5, 25, 26, 31, 23, 16, 14, 4, 20, 1, 5, 25, 26, 31, 23, 16, 14, 4, 20, 1, 5, 25 and 26. Therefore, we take the sixty-four input values as the corresponding sixty-four time units and the sixty-four outputs as the sixty-four samples of signals. Each sample encodes an output of C_f.

The output can take 1, 5, 25, 26, 31, 23, 16, 14, 4 or 20. The sixty-four input values from $r_1^0\ r_2^0\ r_3^0\ r_4^0\ r_5^0\ r_6^0$ through $r_1^1\ r_2^1\ r_3^1\ r_4^1\ r_5^1\ r_6^1$ corresponds to sixty-four time units from zero through sixty-three.

We use Figure 5.6 to explain the reason of why figuring out the order r of 5 modulo 33 is equivalent to compute the period r and the frequency f of signals in the time domain (the input domain). In Figure 5.6, the horizontal axis is to represent the time domain that is the input domain of C_f. The vertical axis is to represent signals that encode the sixty-four outputs of C_f. For convenience of presentation, we make use of variable k to represent the decimal value of each binary input and apply 5^k mod 33 to represent $5^{r_1r_2r_3r_4r_5r_6} \pmod{33}$.

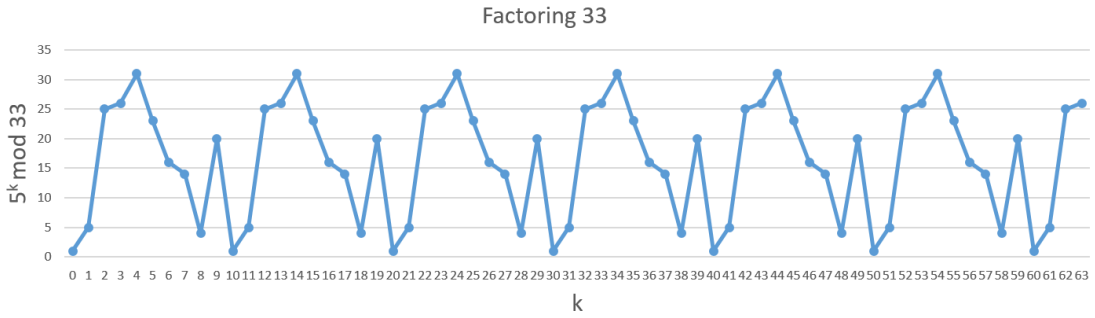

Figure 5.6: Sampling sixty-four points from sixty-four outputs of a given oracular function that is C_f: $\{r_1\ r_2\ r_3\ r_4\ r_5\ r_6 \mid \forall\ r_d \in \{0, 1\} \text{ for } 1 \le d \le 6\} \rightarrow \{5^{r_1r_2r_3r_4r_5r_6} \pmod{33} \mid \forall\ r_d \in \{0, 1\} \text{ for } 1 \le d \le 6\}$.

From Figure 5.6, hidden patterns and information stored in a given oracular function C_f are that the signal rotates back to its *first* signal (output with 1) *seven* times. Its signal rotates back to its *second* signal (output with 5) *seven* times. Its signal rotates back to its *third* signal (output with 25) *seven* times and its signal rotates back to its *fourth* signal (output with 26) *seven* times. Its signal rotates back to its *fifth* signal (output with 31) *six* times and its signal rotates back to its *sixth* signal (output with 23) *six* times. Its signal rotates back to its *seventh* signal (output with 16) *six* times and its signal rotates back to its *eighth* signal (output with 14) *six* times. Its signal rotates back to its *ninth* signal (output with 4) *six* times and its signal rotates back to its *tenth* signal (output with 20) *six* times. This implies that there are $(6\frac{4}{10})$ periods of signals per sixty-four time units and the frequency f of signals is equal to $(6\frac{4}{10})$.

Because in Figure 5.6 the period r of signals is the *reciprocal* of the frequency f of

signals, the period r of signals is $\frac{1}{6-\frac{4}{\frac{10}{64}}} = \frac{1}{\frac{64}{\frac{10}{64}}} = \frac{64}{\frac{64}{10}}$ = 10 / 1 = 10. The period r = 10 of signals in Figure 5.6 satisfies $C_f(r_1^0\ r_2^0\ r_3^1\ r_4^0\ r_5^1\ r_6^0) = 5^{r_1^0 r_2^0 r_3^1 r_4^0 r_5^1 r_6^0} \pmod{33} = 5^{10} \pmod{33} = 9765625 \pmod{33} = 1 \pmod{33}$, so the period r = 10 of signals in C_f is equivalent to the order r = 10 of 5 modulo 33. The cost to find the order r = 10 of 5 modulo 33 is to implement sixty-four (2^6) operations of modular exponentiation, $5^{r_1 r_2 r_3 r_4 r_5 r_6} \pmod{33}$. Since r = 10 is even and is less than 33, from **Lemma 5-2**, we use Euclid's algorithm to compute gcd(33, $5^{\frac{10}{2}} + 1$) and gcd(33, $5^{\frac{10}{2}} - 1$). This indicates that two nontrivial factors for N = 33 are respectively 3 and 11. Therefore, the prime factorization for N = 33 is $N = 3 \times 11$.

Since ($\frac{1}{6-\frac{4}{\frac{10}{64}}} = \frac{1}{\frac{64}{\frac{10}{64}}} = \frac{64}{\frac{64}{10}}$ = 10 / 1) is a rational number and is an integer, we use the continued fractional algorithm in Figure 5.2 to determine the continued fractional representation of (c / d) if c = 10 and d = 1 and the corresponding convergent for explaining how the continued fractional algorithm works out in real applications. From the first execution of statement S_0 through statement S_2, it obtains i = 1, $q[1] = c / d$ = 10 / 1 = 10 and r = 10 (mod 1) = 0. This is to split (10 / 1) into its integer and fractional part and not to invert its fractional part,

$$\frac{10}{1} = 10 + \frac{0}{1} = 10. \tag{5.18}$$

Because the value of r is equal to 0, from the first execution of statement S_3, it returns to a *true*. Therefore, next, from the first execution of statement S_4, the answer is to the continued fractional representation of (10 / 1)

$$\frac{10}{1} = (q[1] = 10) = 10. \tag{5.19}$$

Next, from the first execution of Statement S_5, it terminates the execution of the continued fractional algorithm. For a rational number (10 / 1), the first convergent is (q[1]) = 10 = $\frac{10}{1}$ that is the closest to ($\frac{1}{6-\frac{4}{\frac{10}{64}}} = \frac{1}{\frac{64}{\frac{10}{64}}} = \frac{64}{\frac{64}{10}} = \frac{10}{1}$) and is actually equal to $\frac{10}{1}$. This means that the first convergent (q[1]) = 10 = $\frac{10}{1}$ is equal to the period $r = \frac{r}{1}$. Hence, we get that the period r is equal to the numerator 10 of the first convergent.

Because the numerator $r = 10$ of the first convergent is less than $N = 33$, the numerator $r = 10$ is equivalent to that the order $r = 10$ of 5 modulo 33 satisfies $5^{10} = 1 \pmod{33}$.

5.10 The Possibility of Finding the Even Order of X Modulo N

We assume that the set $\boldsymbol{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ of integers and the set $\boldsymbol{Y} = \{0, 1, 2, 3, \ldots\}$ of natural numbers. The notion $d \mid a$ (read "d ***divides*** a") means that $a = q \times d$ for some integer q and a is a ***multiple*** of d and d is a divisor of a. For example, 15 is a multiple of 1, 3, 5, and 15 and 1, 3, 5, and 15 are the divisors of 15. Every integer a is divisible by the ***trivial divisor*** 1 and a. Nontrivial divisor of integer a are also called ***factors*** of a. For example, the factors (the nontrivial divisors) of 15 are 3 and 5.

An integer $a > 1$ whose only divisors are the trivial divisor a and 1 is said to be a ***prime*** number (or, more simply, a ***prime***). The first six prime, in order, are 2, 3, 5, 7, 11 and 13. An integer $a > 1$ that is not a prime is said to be a ***composite*** number (or, more simply, a ***composite***). For example, 15 is a composite because it has the factors 3 and 5. The integer 1 is said to be a ***unit*** and is neither prime nor composite. Similarly, the integer 0 and all negative integers are neither prime nor composite. From (5.1), for given any positive integers w and N, there are unique integers q and r such that $0 \le r < N$ and $w = q \times N + r$. Integer q is the ***quotient*** (result) of dividing w by N. Integer r is the ***remainder*** of dividing w by N and we write $r = w \pmod{N}$. Given any integer N, we can partition the integers into those that are multiples of N and those that are not multiples of N. By classifying the multiples and the non-multiples of N in light of their remainders when divided by N, we can obtain the refinement of this partition.

According to their remainders modulo N, the integer can be divided into N equivalent classes. The ***equivalent class modulo*** N including an integer w is

$$[w]_N = \{w + q \times N : q \in \boldsymbol{Z}\}. \tag{5.20}$$

For example, $[2]_5 = \{\ldots, 2, 7, 12, 17, \ldots\}$. If the remainder of w modulo N is the same as that of a modulo N, then we can say that writing $a \in [w]_N$ is the same as writing $a = w \pmod{N}$. The set of all such equivalent classes is

$$\boldsymbol{Z}_N = \{[w]_N : 0 \le w \le N - 1\} = \{0, 1, 2, 3, \ldots N - 1\}. \tag{5.21}$$

In (5.21), we use 0 to represent $[0]_N$, we apply 1 to represent $[1]_N$, we make use of 2 to represent $[2]_N$ and so on with that we use apply $N - 1$ to represent $[N - 1]_N$. We use its

least nonnegative element to represent each class.

Because the equivalent class of two integers uniquely determines the equivalent class of their sum, product or difference, we can easily define addition, multiplication and subtraction operations for $\boldsymbol{Z}_N$. This is to say that if $c = c^1 \pmod N$ and $d = d^1 \pmod N$, then

$$c + d = c^1 + d^1 \pmod N,\ c \times d = c^1 \times d^1 \pmod N \text{ and } c - d = c^1 - d^1 \pmod N \tag{5.22}$$

Therefore, we denote addition, multiplication and subtraction modulo N, defined $+_N$, $\times_N$ and $-_N$, as follows:

$$[c]_N +_N [d]_N = [c + d]_N,\ [c]_N \times_N [d]_N = [c \times d]_N \text{ and } [c]_N -_N [d]_N = [c - d]_N. \tag{5.23}$$

Applying this definition of addition modulo N in (5.23), we define the ***additive group modulo*** N as $(\boldsymbol{Z}_N, +_N)$. We use **Lemma 5-7** to show that the system $(\boldsymbol{Z}_N, +_N)$ is a finite abelian group.

Lemma 5-7: The system $(\boldsymbol{Z}_N, +_N)$ is a finite abelian group.

Proof:

For any two elements $[c]_N$ and $[d]_N$ in $\boldsymbol{Z}_N$, from (5.20) through (5.23), we obtain that $0 \le c \le N - 1$, $0 \le d \le N - 1$ and $[c]_N +_N [d]_N = [c + d]_N$. If $0 \le c + d \le N - 1$, then $[c]_N +_N [d]_N = [c + d]_N$ is an element in $\boldsymbol{Z}_N$. If $N \le c + d \le 2 \times N - 2$, then $[c]_N +_N [d]_N = [c + d]_N = [c + d - N]_N$. Because $0 \le c + d - N \le N - 2$, it is one element in $\boldsymbol{Z}_N$. This means that the system $(\boldsymbol{Z}_N, +_N)$ is closed.

For any three elements $[c]_N$, $[d]_N$ and $[e]_N$ in $\boldsymbol{Z}_N$, from (5.23), we obtain $([c]_N +_N [d]_N) +_N [e]_N = ([c + d]_N) +_N [e]_N = [(c + d) + e]_N = [c + (d + e)]_N = [c]_N +_N ([d + e]_N) = [c]_N +_N ([d]_N) +_N [e]_N)$. This indicates that the system $(\boldsymbol{Z}_N, +_N)$ satisfies the associativity of $+_N$.

For any two elements $[c]_N$ and $[d]_N$ in $\boldsymbol{Z}_N$, from (5.23), we obtain $[c]_N +_N [d]_N = [c + d]_N = [d + c]_N = [d]_N +_N [c]_N$. This implies that the system $(\boldsymbol{Z}_N, +_N)$ satisfies the commutativity of $+_N$.

The identity element of the system $(\boldsymbol{Z}_N, +_N)$ is $[0]_N$ because for any element $[c]_N$ in

$\boldsymbol{Z}_N$, from (5.23), we have $[c]_N +_N [0]_N = [c + 0]_N = [c]_N = [0 + c]_N = [0]_N +_N [c]_N$. The additive inverse of any element $[c]_N$ in $\boldsymbol{Z}_N$ is $[N - c]_N$ because $[c]_N +_N [N - c]_N = [c + N - c]_N = [N]_N = [0]_N$. The number of elements in the system $(\boldsymbol{Z}_N, +_N)$ is N, so it is finite. Therefore, from the statements above, we at once infer that the system $(\boldsymbol{Z}_N, +_N)$ is a finite abelian group. ■

The set $\boldsymbol{Z}_N^*$ is the set of elements in $\boldsymbol{Z}_N$ that are relatively prime to N and is

$$\boldsymbol{Z}_N^* = \{[w]_N \in \boldsymbol{Z}_N : \gcd(w, N) = 1\}. \tag{5.24}$$

Because $[w]_N = \{w + q \times N : q \in \boldsymbol{Z}\}$ and $\gcd(w, N) = 1$, we have $\gcd(w + q \times N, N) = 1$. For example, $\boldsymbol{Z}_{15}^* = \{[1]_{15}, [2]_{15}, [4]_{15}, [7]_{15}, [8]_{15}, [11]_{15}, [13]_{15}, [14]_{15}\}$. Using the definition of multiplication modulo N in (5.23), we denote the multiplicative group modulo N as $(\boldsymbol{Z}_N^*, \times_N)$. We make use of **Lemma 5-8** to demonstrate that the system $(\boldsymbol{Z}_N^*, \times_N)$ is a finite abelian group.

Lemma 5-8: The system $(\boldsymbol{Z}_N^*, \times_N)$ is a finite abelian group.

Proof:

For any two elements $[c]_N$ and $[d]_N$ in $\boldsymbol{Z}_N^*$, from (5.20) through (5.23), we get that $0 \le c \le N - 1$, $0 \le d \le N - 1$, $\gcd(c, N) = 1$, $\gcd(d, N) = 1$ and $[c]_N \times_N [d]_N = [c \times d]_N$. Because $\gcd(c, N) = 1$ and $\gcd(d, N) = 1$, we have $\gcd(c \times d, N) = 1$. This means that $[c]_N \times_N [d]_N = [c \times d]_N$ is an element in $\boldsymbol{Z}_N^*$. Therefore, the system $(\boldsymbol{Z}_N^*, \times_N)$ is closed.

For any three elements $[c]_N$, $[d]_N$ and $[e]_N$ in $\boldsymbol{Z}_N^*$, from (5.23), we obtain $([c]_N \times_N [d]_N) \times_N [e]_N = ([c \times d]_N) \times_N [e]_N = [(c \times d) \times e]_N = [c \times (d \times e)]_N = [c]_N \times_N ([d + e]_N) = [c]_N \times_N ([d]_N) \times_N [e]_N)$. This is to say that the system $(\boldsymbol{Z}_N^*, \times_N)$ satisfies the associativity of $\times_N$.

For any two elements $[c]_N$ and $[d]_N$ in $\boldsymbol{Z}_N^*$, from (5.23), we get $[c]_N \times_N [d]_N = [c \times d]_N = [d \times c]_N = [d]_N \times_N [c]_N$. This indicates that the system $(\boldsymbol{Z}_N^*, \times_N)$ satisfies the commutativity of $\times_N$.

For any element $[c]_N$ in $\boldsymbol{Z}_N^*$, from (5.23), we have $[c]_N \times_N [1]_N = [c \times 1]_N = [c]_N = [1 \times c]_N = [1]_N \times_N [c]_N$. This indicates that the identity element of the system $(\boldsymbol{Z}_N^*, \times_N)$ is $[1]_N$.

Any element $[c]_N$ in $\boldsymbol{Z}_N^*$ satisfies gcd(c, N) = 1. Therefore, from **Lemma 5-5**, there exists a unique multiplicative inverse $[c^{-1}]_N$ of $[c]_N$, modulo N, such that $[c]_N \times_N [c^{-1}]_N = [c \times c^{-1}]_N = [1]_N = [c^{-1} \times c]_N = [c^{-1}]_N \times_N [c]_N$. The number of elements in the system $(\boldsymbol{Z}_N^*, \times_N)$ is less than N, so it is finite. Hence, from the statements above, we at once derive that the system $(\boldsymbol{Z}_N^*, \times_N)$ is a finite abelian group. ■

The size of $\boldsymbol{Z}_N^*$ is known as ***Euler's phi function*** $\phi(N)$ satisfies the following equation

$$\phi(N) = N \times (\prod_{p|N}(1 - \frac{1}{p})), \tag{5.25}$$

where p runs over all the primes dividing n (including N itself, if N is a prime). For example, because the prime divisors of 15 are 3 and 5, we obtain $\phi(15) = 15 \times (1 - (1 / 3)) \times (1 - (1 / 5)) = 15 \times (2 / 3) \times (4 / 5) = 8$. This is to say that the size of $\boldsymbol{Z}_{15}^*$ is eight (8) and $\boldsymbol{Z}_{15}^*$ is equal to $\{[1]_{15}, [2]_{15}, [4]_{15}, [7]_{15}, [8]_{15}, [11]_{15}, [13]_{15}, [14]_{15}\}$. If p is a prime, then p itself is the only prime divisor. Therefore, from (5.25), we obtain $\phi(p) = p \times (1 - (1 / p)) = p - 1$. The only integers that are less than p^a and are not co-prime to p^a are the multiples of p: p, $2 \times p$, …, $(p^{a-1} - 1) \times p$, from which we infer

$$\phi(p^a) = (p^a - 1) - (p^{a-1} - 1) = p^a - p^{a-1} = p^{a-1} \times (p - 1). \tag{5.26}$$

Furthermore, if c and d are co-prime, then $\phi(c \times d)$ satisfies the following equation

$$\phi(c \times d) = \phi(c) \times \phi(d). \tag{5.27}$$

On the other hand, when N is a power of an odd prime p, $N = p^a$. It turns out that $\boldsymbol{Z}_N^* = \boldsymbol{Z}_{p^a}^*$ is a *cyclic* group, that is, there is an element h in $\boldsymbol{Z}_{p^a}^*$ which generates $\boldsymbol{Z}_{p^a}^*$ in the sense that any other element y may be written $y = h^m \pmod N = h^m \pmod{p^a}$ for some non-negative integer m. We use **Lemma 5-9** and **Lemma 5-10** to explain the possibility of finding the even order that are not equal to $N - 1$ of X modulo N.

Lemma 5-9: We assume that p is an odd prime and 2^b is the largest power of 2 dividing $\phi(p^a)$. Then with probability exactly one-half 2^b divides the order modulo p^a of a randomly chosen element of $\boldsymbol{Z}_{p^a}^*$.

Proof:

Because p is an odd prime, from (5.26) we obtain that $\phi(p^a) = p^{a-1} \times (p - 1)$ is even. Because $\phi(p^a)$ is even and 2^b divides $\phi(p^a)$, we obtain $b \geq 1$. Since $\boldsymbol{Z}_{p^a}^*$ is a *cyclic* group,

there exists an element h in $\boldsymbol{Z}_p{}^{a*}$ which generates $\boldsymbol{Z}_p{}^{a*}$ in the sense that any other element X may be written $X = h^m \pmod{p^a}$ for some m in the range 1 through $\phi(p^a)$ that is the size of $\boldsymbol{Z}_p{}^{a*}$. Let r the order of h^m modulo p^a and consider two cases. The first case is when m is odd. Since h is co-prime to (p^a) and $\phi(p^a)$ is the size of $\boldsymbol{Z}_p{}^{a*}$, $\phi(p^a)$ is the least value such that $h^{\phi(p^a)} = 1 \pmod{p^a}$. Because $(h^m)^r = h^{m \times r} = 1 \pmod{p^a}$, we infer that $\phi(p^a)$ divides $(m \times r)$. Since m is odd, $\phi(p^a)$ is even, 2^b divides $\phi(p^a)$ and $\phi(p^a)$ divides $(m \times r)$ and 2^b divides $(m \times r)$, we infer that 2^b divides r. The second case is when m is even. Because h is co-prime to p^a and m is even, we infer that $h^{m/2}$ modulo p^a is co-prime to p^a. Therefore, we have $(h^{m \times \phi(p^a)/2}) = (h^{\phi(p^a)})^{m/2} = (1)^{m/2} = 1 \pmod{p^a}$. Because r is the order of h^m modulo p^a that is the least value such that $(h^m)^r = h^{m \times r} = 1 \pmod{p^a}$, we infer that r divides $(\phi(p^a) / 2)$ and r is less than 2^b that is the largest power of 2 dividing $\phi(p^a)$. Thus, we infer that 2^b does not divide r.

Because the value of m is in the range 1 through $\phi(p^a)$ that is even and is the size of $\boldsymbol{Z}_p{}^{a*}$, we may partition $\boldsymbol{Z}_p{}^{a*}$ into two sets of equal size. The first set of equal size is those that may be written $h^m \pmod{p^a}$ with that m is odd, for which 2^b divides r that is the order of h^m modulo p^a. The second set of equal size is those that may be written $h^m \pmod{p^a}$ with that m is even, for which 2^b does not divide r that is the order of h^m modulo p^a. Therefore, with probability $(1 / 2)$ the integer 2^b divides the order r of a randomly chosen element $\boldsymbol{Z}_p{}^{a*}$, and with probability $(1 / 2)$ it does not. ■

Lemma 5-10: We assume that $N = p_1^{a_1} \times \ldots \times p_m^{a_m}$ is the prime factorization of an odd composite positive integer. Let X be chosen uniformly at random from $\boldsymbol{Z}_N{}^*$ and let r be the order of X modulo N. Then $P(r \text{ is even and } X^{r/2} \neq -1 \pmod{N}) \geq 1 - (1 / 2^m)$.

Proof:

We show that $P(r \text{ is odd or } X^{r/2} = -1 \pmod{N}) \leq 1 / 2^m$. According to the Chinese remainder theorem, selecting X uniformly at random from $\boldsymbol{Z}_N{}^*$ is equivalent to selecting X_k independently and uniformly at random from $\boldsymbol{Z}^*_{p_k{}^{a_k}}$, and satisfying that $X = X_k \pmod{p_k^{a_k}}$ for $1 \leq k \leq m$. Let r_k be the order of X_k modulo $(p_k^{a_k})$. Let 2^{b_k} be the largest power of 2 dividing r_k and 2^b is the largest power of 2 dividing r. Because X is co-prime to N $(p_1^{a_1} \times \ldots \times p_m^{a_m})$, $\phi(N) = \phi(p_1^{a_1} \times \ldots \times p_m^{a_m})$ is the size of $\boldsymbol{Z}_N{}^*$ that is the least value such that $X^{\phi(N)} = X^{\phi(p_1^{a_1} \times \ldots \times p_m^{a_m})} = 1 \pmod{N}$. Since r is the order of X modulo N that is the least value such that $X^r = 1 \pmod{N}$, we have $r = \phi(p_1^{a_1} \times \ldots \times p_m^{a_m}) = \phi(p_1^{a_1}) \times \ldots \times \phi(p_m^{a_m})$. Because X_k is co-prime to $(p_k^{a_k})$ for $1 \leq k \leq m$, $\phi(p_k^{a_k})$ is the size

of $\boldsymbol{Z}^*_{p_k^{a_k}}$ that is the least value such that $X_k^{\phi(p_k^{a_k})} = 1 \pmod{(p_k^{a_k})}$. Since r_k is the order of X_k modulo $(p_k^{a_k})$ that is the least value such that $X_k{}^r = 1 \pmod{(p_k^{a_k})}$, we have $r_k = \phi(p_k^{a_k})$ for $1 \le k \le m$. Because $r = \phi(p_1^{a_1}) \times \ldots \times \phi(p_m^{a_m})$ and $r_k = \phi(p_k^{a_k})$ for $1 \le k \le m$, we infer that r_k divides r for $1 \le k \le m$. We will show that to have r odd or $X^{r/2} = -1 \pmod{N}$ it is necessary that b_k takes the same value for $1 \le k \le m$. The result then follows, as from **Lemma 5-9** the probability of this occurring is at most $(1/2) \times (1/2) \times \ldots \times (1/2) = 1/2^m$.

We consider the first case is when r is odd. Because r_k divides r for $1 \le k \le m$, we infer r_k is odd. Because 2^{b_k} divides r_k for $1 \le k \le m$, we obtain $b_k = 0$ for $1 \le k \le m$. The second case is when r is even and $X^{r/2} = -1 \pmod{N}$. This is to say that $X^{r/2} = N - 1 = p_1^{a_1} \times \ldots \times p_m^{a_m} - 1$. Therefore, we have $X^{r/2} = N - 1 = p_1^{a_1} \times \ldots \times p_m^{a_m} - 1 = -1 \pmod{(p_k^{a_k})}$. So we obtain that r_k does not divide $(r/2)$. Because r_k divides r for $1 \le k \le m$, we must have $b_k = b$ for $1 \le k \le m$. Since $P(r \text{ is even and } X^{r/2} \ne -1 \pmod{N}) + P(r \text{ is odd or } X^{r/2} = -1 \pmod{N}) = 1$ and $P(r \text{ is odd or } X^{r/2} = -1 \pmod{N}) \le 1/2^m$, we have $P(r \text{ is even and } X^{r/2} \ne -1 \pmod{N}) \ge 1 - (1/2^m)$. ■

5.11 Public Key Cryptography and the RSA Cryptosystem

A consumer wants to buy something on the internet. He would like to transmit his credit card number over the internet in such a way that only the company offering the products that he is buying can receive the number. A *cryptographic protocol* or a *cryptosystem* on the internet can achieve such private communication. Effective cryptosystems make it easy for two parties who want to communicate each other, but make it very difficult for the eavesdropper to eavesdrop on the content of the conversation.

A particularly important class of cryptosystems are the *public key cryptosystems*. In a public key cryptosystem, Mary wants to send messages to her friends and to receive messages sent by her friends. She must first generate two *cryptographic keys*. One is a *public key*, *P*, and the other is a secret key, *S*. After Mary has generated her keys, she announces or publishes the public key so that anybody can gain access to the public key.

John is Mary's good friend. He would like to send a private message to Mary. Therefore, John first gets a copy of Mary's public key *P*. Then, he encrypts the private message he wants to send Mary, making use of Mary's public key *P* to complete the

encryption. Because the public key and the encoded message is the only information available to an eavesdropper, it will be impossible for the eavesdropper to recover the message. However, Mary has the secret key S that is not available to an eavesdropper. She uses the secret key S to decrypt the encrypted message and obtains the *original* message. This transformation known as decryption is inverse to encryption, allowing Mary to recover John's private message.

The most widely used of public key cryptosystems is the **RSA** cryptosystem, named **RSA** for the initials of its creators, Rivest, Shamir, and Adleman. The presumed security of the **RSA** cryptosystem is based on the apparent difficulty of factoring on a digital computer. Now Mary wishes to generate public and secret keys for use with the **RSA** cryptosystem. She makes use of the following procedure to generate them:

(1) Choose two large prime numbers, p and q.
(2) Calculate the product $N = p \times q$.
(3) Choose at random a small odd integer, e, which is relatively prime to $\phi(N) = (p - 1) \times (q - 1)$.
(4) Calculate d, the multiplicative inverse of e, modulo $\phi(N)$.
(5) The public key is the pair $P = (e, N)$.
(6) The secret key is the pair $S = (d, N)$.

Now John uses the public key (e, N) to encrypt a message M to send to Mary. We assume that the message M has only $\log_2 N$ bits, as longer messages may be encrypted by means of breaking M up into blocks of at most $\log_2 N$ bits and then encrypting the blocks separately. The encryption procedure for a single block is to calculate:

$$E(M) = M^e \pmod{N}. \tag{5.28}$$

$E(M)$ is the encrypted version of the message M, which John sends to Mary. Mary is able to decrypt quickly the message applying her secret key $S = (d, N)$, simply by raising the encrypted message to the dth power:

$$D(E(M)) = M^{e \times d} \pmod{N} = M \pmod{N}. \tag{5.29}$$

How can the **RSA** cryptosystem be broken? The answer is to that if we can efficiently factor a big composite number N into the production of two big prime numbers p and q. Then, we can extract p and q. This means that we can efficiently calculate $\phi(N) = (p - 1) \times (q - 1)$. Next, we can efficiently compute d, the multiplicative inverse of e,

modulo $\phi(N)$. Therefore, we can completely determine the secret key (d, N). So, if factoring large numbers were easy then breaking the RSA cryptosystem would be easy.

5.12 Implementing the Controlled-Swap Gate of Three Quantum Bits

A (8 × 8) matrix ***CSWAP*** and its conjugate transpose $\overline{\boldsymbol{CSWAP}}$ are respectively

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (5.30)$$

Because ***CSWAP*** × $\overline{\boldsymbol{CSWAP}}$ is equal to a (8 × 8) identify matrix and $\overline{\boldsymbol{CSWAP}}$ × ***CSWAP*** is equal to a (8 × 8) identify matrix, matrix ***CSWAP*** and matrix $\overline{\boldsymbol{CSWAP}}$ are both a unitary matrix (a unitary operator). Matrix ***CSWAP*** is to matrix representation of a ***CSWAP*** (***controlled-SWAP***) gate of three quantum bits. The left picture in Figure 5.7 is the first graphical circuit representation of a ***CSWAP*** gate with three quantum bits. Quantum bit $|C_1\rangle$ at the bottom in the left picture in Figure 5.7 is the controlled bit, and quantum bit $|S_1\rangle$ at the top and quantum bit $|S_2\rangle$ at the middle in the left picture in Figure 5.7 are both the target bits. The functionality of the ***CSWAP*** gate is to that if the controlled bit $|C_1\rangle$ is equal to $|1\rangle$, then it exchanges the information contained in the two target bits $|S_1\rangle$ and $|S_2\rangle$. Otherwise, it does not exchange the information contained in the two target bits $|S_1\rangle$ and $|S_2\rangle$. The middle picture in Figure 5.7 is the second graphical circuit representation of a ***CSWAP*** gate with three quantum bits. The right picture in Figure 5.7 is to the graphical circuit representation of implementing the ***CSWAP*** gate by means of using three ***CCNOT*** gates. In the right picture in Figure 5.7, if the controlled bit $|C_1\rangle$ is equal to $|1\rangle$, then using three ***CNOT*** gates implements one ***SWAP*** gate to exchange the information contained in the two target bits $|S_1\rangle$ and $|S_2\rangle$. Otherwise, it does not implement three ***CNOT*** gates to complete one ***SWAP*** gate and to exchange the information contained in the two target bits $|S_1\rangle$ and $|S_2\rangle$.

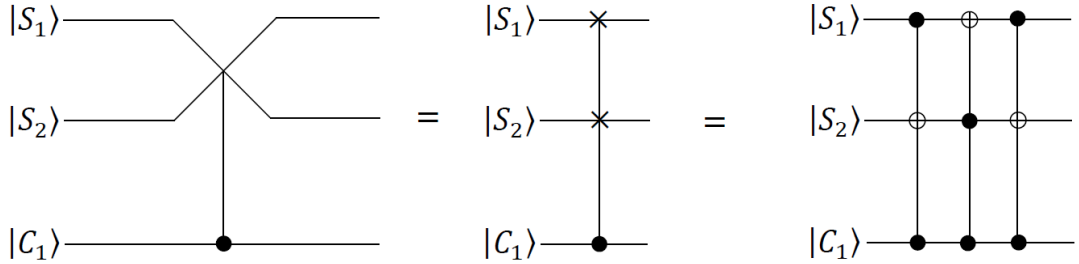

Figure 5.7: Circuit Representation of a **CSWAP** gate with three quantum bits.

5.12.1 Quantum Programs to Implement the Controlled-Swap Gate of Three Quantum Bits

In **IBM Q** Experience, it does not provide one quantum instruction (operation) of implementing the ***CCNOT*** gate (the Toffoli gate) with three quantum bits. We decompose **CCNOT** gate into *six* **CNOT** gates and *nine* gates of one quantum bits that are shown in Figure 5.8. In Figure 5.8, H is the Hadamard gate, $T = \begin{bmatrix} 1 & 0 \\ 0 & e^{\sqrt{-1}\times\frac{\pi}{4}} \end{bmatrix}$ and $T^{+} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-1\times\sqrt{-1}\times\frac{\pi}{4}} \end{bmatrix}$. In the backend *simulator* with thirty-two quantum bits, there is no limit for connectivity of a ***CNOT*** gate among thirty-two quantum bits.

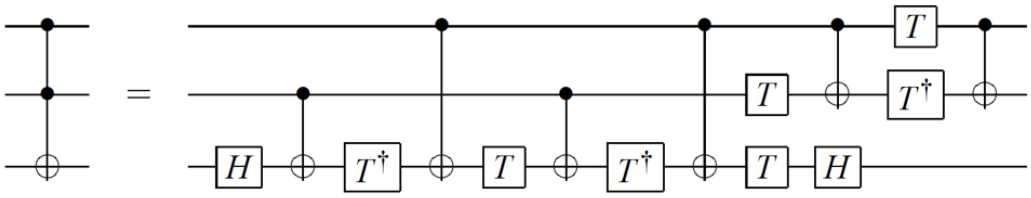

Figure 5.8: Decomposing **CCNOT** gate into six **CNOT** gates and nine gates of one bit.

In Listing 5.1, the program in the backend *simulator* with thirty-two quantum bits in **IBM**'s quantum computer is the *first* example of the *fifth* chapter in which we illustrate how to write a quantum program to implement a ***CSWAP*** gate with three quantum bits. Figure 5.9 is the corresponding quantum circuit of the program in Listing 5.1. For the convenience of our presentation, there are four instructions in the same line in Listing 5.1. We use "instruction number" or "line number" to indicate the order of the execution to each instruction in Listing 5.1.

The statement "OPENQASM 2.0;" on instruction number one in the first line of Listing 5.1 is to point out that the program is written with version 2.0 of Open QASM. Next, the statement "include "qelib1.inc";" on instruction number two in the first line of Listing 5.1 is to continue parsing the file "qelib1.inc" as if the contents of the file were pasted at the location of the include statement, where the file "qelib1.inc" is **Quantum Experience (QE) Standard Header** and the path is specified relative to the current working directory.

Next, the statement “qreg q[3];” on instruction number three in the first line of Listing 5.1 is to declare that in the program there are three quantum bits. In the left top of Figure 5.9, three quantum bits are subsequently q[0], q[1] and q[2]. The initial value of each quantum bit is set to |0>. We use three quantum bits q[0], q[1] and q[2] to subsequently encode the first target bit $|S_1>$, the second target bit $|S_2>$ and the controlled bit $|C_1>$.

```
1. OPENQASM 2.0;      2. include "qelib1.inc";      3. qreg q[3];      4. creg c[3];
5. x q[0];                    6. x q[2];
// Implement the first CCNOT gate in the right picture of Figure 5.7 with two
// controlled bits q[2] and q[0] and target bit q[1].

7. barrier q[0], q[1], q[2];      8. h q[1];       9. cx q[0],q[1];      10. tdg q[1];
11. cx q[2],q[1];                 12. t q[1];      13. cx q[0],q[1];     14. tdg q[1];
15. cx q[2],q[1];                 16. t q[0];      17. t q[1];           18. h q[1];
19. cx q[2],q[0];                 20 tdg q[0];     21 t q[2];            22. cx q[2],q[0];

// Implement the second CCNOT gate in the right picture of Figure 5.7 with two
// controlled bits q[2] and q[1] and target bit q[0].

23. barrier q[0], q[1], q[2];     24 h q[0];       25. cx q[1],q[0];     26. tdg q[0];
27. cx q[2],q[0];                 28. t q[0];      29. cx q[1],q[0];     30. tdg q[0];
31. cx q[2],q[0];                 32. t q[1];      33. t q[0];           34. h q[0];
35. cx q[2],q[1];                 36. tdg q[1];    37. t q[2];           38. cx q[2],q[1];

// Implement the third CCNOT gate in the right picture of Figure 5.7 with two
// controlled bits q[2] and q[0] and target bit q[1].

39. barrier q[0], q[1], q[2];     40. h q[1];      41. cx q[0],q[1];     42. tdg q[1];
43. cx q[2],q[1];                 44. t q[1];      45. cx q[0],q[1];     46. tdg q[1];
47. cx q[2],q[1];                 48. t q[0];      49. t q[1];           50. h q[1];
51. cx q[2],q[0];                 52. tdg q[0];    53. t q[2];           54. cx q[2],q[0];

55. measure q[0] -> c[0];
56. measure q[1] -> c[1];
57. measure q[2] -> c[2];
```

Listing 5.1: The program of implementing a ***CSWAP*** gate of three quantum bits.

For the convenience of our explanation, $q[k]^0$ for $0 \le k \le 2$ is to represent the value 0 of q[k] and $q[k]^1$ for $0 \le k \le 2$ is to represent the value 1 of q[k]. Similarly, for the convenience of our explanation, an initial state vector of implementing a ***CSWAP*** gate is as follows:

$$|\Phi_0\rangle = |q[2]^0\rangle\, |q[1]^0\rangle\, |q[0]^0\rangle = |0\rangle\, |0\rangle\, |0\rangle = |000\rangle.$$

Then, the statement "creg c[3];" on instruction number four in the first line of Listing 5.1 is to declare that there are three classical bits in the program. In the left bottom of Figure 5.9, three classical bits are respectively c[0], c[1] and c[2]. The initial value of each classical bit is set to 0. Classical bit c[2] is the most significant bit and classical bit c[0] is the least significant bit.

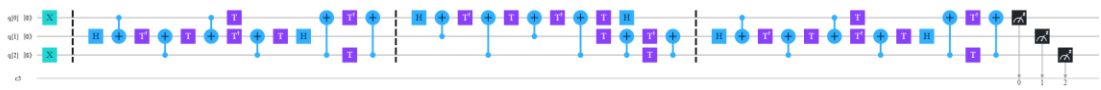

Figure 5.9: The corresponding quantum circuit of the program in Listing 5.1.

Next, the two statements "x q[0];" and "x q[2];" on line number *five* through line number *six* in the second line of Listing 5.1 implement two ***NOT*** gates to quantum bit q[0] and quantum bit q[2] in the *first* time slot of the quantum circuit in Figure 5.9. They both actually complete $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$. This indicates that converting q[0] from one state |0> to another state |1> and converting q[2] from one state |0> to another state |1> are completed. Since in the *first* time slot of the quantum circuit in Figure 5.9 there is no quantum gate to act on quantum bit q[1], its state |0> is not changed. Therefore, we have the following new state vector

$$|\Phi_1\rangle = |q[2]^1\rangle\, |q[1]^0\rangle\, |q[0]^1\rangle.$$

In the state vector $|\Phi_1\rangle = |q[2]^1\rangle\, |q[1]^0\rangle\, |q[0]^1\rangle$, quantum bit $|q[2]^1\rangle$ is the input state |1> of the controlled bit $|C_1\rangle$ in the ***CSWAP*** gate in Figure 5.7. Quantum bit $|q[1]^0\rangle$ is the input state |0> of the target bit $|S_2\rangle$ in the ***CSWAP*** gate in Figure 5.7. Quantum bit $|q[0]^1\rangle$ is the input state |1> of the target bit $|S_1\rangle$ in the ***CSWAP*** gate in Figure 5.7. Next, the statement "barrier q[0], q[1], q[2];" on line number seven in the *sixth* line of Listing 5.1 implements one barrier instruction to prevent optimization from reordering gates across its source line in the *second* time slot of the quantum circuit in Figure 5.9. Next, from instruction number *eight* through instruction number *twenty-two* in Listing

5.1, the fifteen statements are “h q[1];” “cx q[0],q[1];”, “tdg q[1];”, “cx q[2], q[1];”, “t q[1];”, “cx q[0],q[1];”, “tdg q[1];”, “cx q[2],q[1];”, “t q[0];”, “t q[1];”, “h q[1];”, “cx q[2],q[0];”, “tdg q[0];”, “t q[2];” and “cx q[2],q[0];”. They take the state vector $|\Phi_1\rangle = |q[2]^1\rangle\ |q[1]^0\rangle\ |q[0]^1\rangle$ as their input and implement the first ***CCNOT*** gate with two controlled bits q[2] and q[0] and one target bit q[1] from the *third* time slot through the *fifteen* time slot of Figure 5.9. Because the two controlled bits q[2] and q[0] are both state |1>, the state |0> of the target bit q[1] is converted into state |1>. Therefore, we have the following new state vector

$$|\Phi_{15}\rangle = |q[2]^1\rangle\ |q[1]^1\rangle\ |q[0]^1\rangle.$$

Next, the statement “barrier q[0], q[1], q[2];” on instruction number twenty-three in Listing 5.1 implements one barrier instruction to prevent optimization from reordering gates across its source line in the *sixteenth* time slot of the quantum circuit in Figure 5.9. Next, from instruction number *twenty-four* through instruction number *thirty-eight* in Listing 5.1, the fifteen statements are “h q[0];”, “cx q[1],q[0];”, “tdg q[0];”, “cx q[2],q[0];”, “t q[0];”, “cx q[1],q[0];”, “tdg q[0];”, “cx q[2],q[0];”, “t q[1];”, “t q[0];”, “h q[0];”, “cx q[2],q[1];”, “tdg q[1];”, “t q[2];” and “cx q[2],q[1];”. They take the state vector $|\Phi_{15}\rangle = |q[2]^1\rangle\ |q[1]^1\rangle\ |q[0]^1\rangle$ as their input and implement the second ***CCNOT*** gate with two controlled bits q[2] and q[1] and one target bit q[0] from the *seventeenth* time slot through the *twenty-eighth* time slot of Figure 5.9. Since the two controlled bits q[2] and q[1] are both state |1>, the state |1> of the target bit q[0] is converted into state |0>. Thus, we obtain the following new state vector

$$|\Phi_{28}\rangle = |q[2]^1\rangle\ |q[1]^1\rangle\ |q[0]^0\rangle.$$

Next, the statement “barrier q[0], q[1], q[2];” on instruction number *thirty-nine* in Listing 5.1 implements one barrier instruction to prevent optimization from reordering gates across its source line in the *thirty-ninth* time slot of the quantum circuit in Figure 5.9. Next, from instruction number *forty* through instruction number *fifty-four* in Listing 5.1, the fifteen statements are “h q[1];”, “cx q[0],q[1];”, “tdg q[1];”, “cx q[2],q[1];”, “t q[1];”, “cx q[0],q[1];”, “tdg q[1];”, “cx q[2],q[1];”, “t q[0];”, “t q[1];”, “h q[1];”, “cx q[2],q[0];”, “tdg q[0];”, “t q[2];” and “cx q[2],q[0];”. They take the state vector $|\Phi_{28}\rangle = |q[2]^1\rangle\ |q[1]^1\rangle\ |q[0]^0\rangle$ as their input and implement the third ***CCNOT*** gate with two controlled bits q[2] and q[0] and one target bit q[1] from the *thirtieth* time slot through the *forty-second* time slot of Figure 5.9. Because the first controlled bit q[2] is state |1> and the second controlled bit q[0] is state |0>, the state |1> of the target bit q[1] is not changed. Hence, we get the following new state vector

$$|\Phi_{42}> = |q[2]^1> |q[1]^1> |q[0]^0>.$$

Next, the three statements “measure q[0] -> c[0];”, “measure q[1] -> c[1];” and “measure q[2] -> c[2];” from instruction number fifty-five through instruction number in Listing 5.1 is to measure the first quantum bit q[0], the second quantum bit q[1] and the third quantum bit q[2]. They record the measurement outcome by overwriting the first classical bit c[0], the second classical bit c[1] and the third classical bit c[2]. In the backend *simulator* with thirty-two quantum bits in **IBM**’s quantum computers, we use the command “run” to execute the program in Listing 5.1. The measured result appears in Figure 5.10. From Figure 5.10, we obtain the answer 110 (c[2] = 1 = q[2] = |1>, c[1] = 1 = q[1] = |1> and c[0] = 0 = q[0] = |0>) with the probability 100%. Because the input state of the target bit $|S_1>$ is state |1>, the input state of the target bit $|S_2>$ is state |0> and the input state of the controlled bit $|C_1>$ is |1> in the ***CSWAP*** gate in Figure 5.7, the information contained in the two target bits $|S_1>$ and $|S_2>$ are exchanged. Therefore, we have the final state of the target bit $|S_1>$ encoded by $|q[0]^0>$ is state |0> and the final state of the target bit $|S_2>$ encoded by $|q[1]^1>$ is state |1> with the probability 100%.

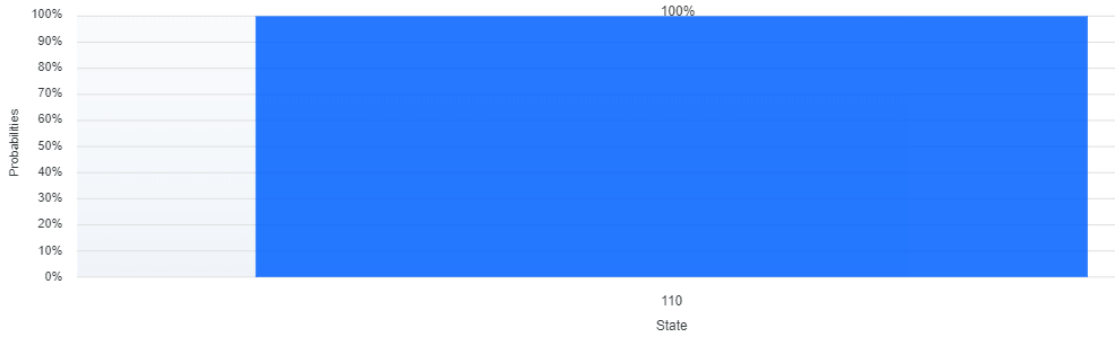

Figure 5.10: After the measurement to the program in Listing 5.1 is completed, we obtain the answer 110 with the probability 100%.

5.13 Shor’s Order-Finding Algorithm

For positive integers X and N with that the value of X is less than the value of N and the greatest common factor for them is one, the order (the period) of X modulo N is to the least positive integer r such that $X^r = 1 \pmod{N}$. The order-finding problem is to compute the order for some given X and N. On a digital computer, no algorithm *known* solves the problem with the number of the bit of specifying N that is L to be greater than or equal to $\log_2(N)$, by means of using resources polynomial in the $O(L)$ bits needed to specify the problem. In this section, we explain how Shor’s order-finding

algorithm is an efficient quantum algorithm to order finding.

Quantum circuit to Shor's order-finding algorithm is schematically depicted in Figure 5.11. The first quantum register of n quantum bits is $(\bigotimes_{k=1}^{n} |p_k^0>)$ and the initial state of each quantum bit is the |0> state. Quantum bit $|p_1{}^0>$ is the most significant bit

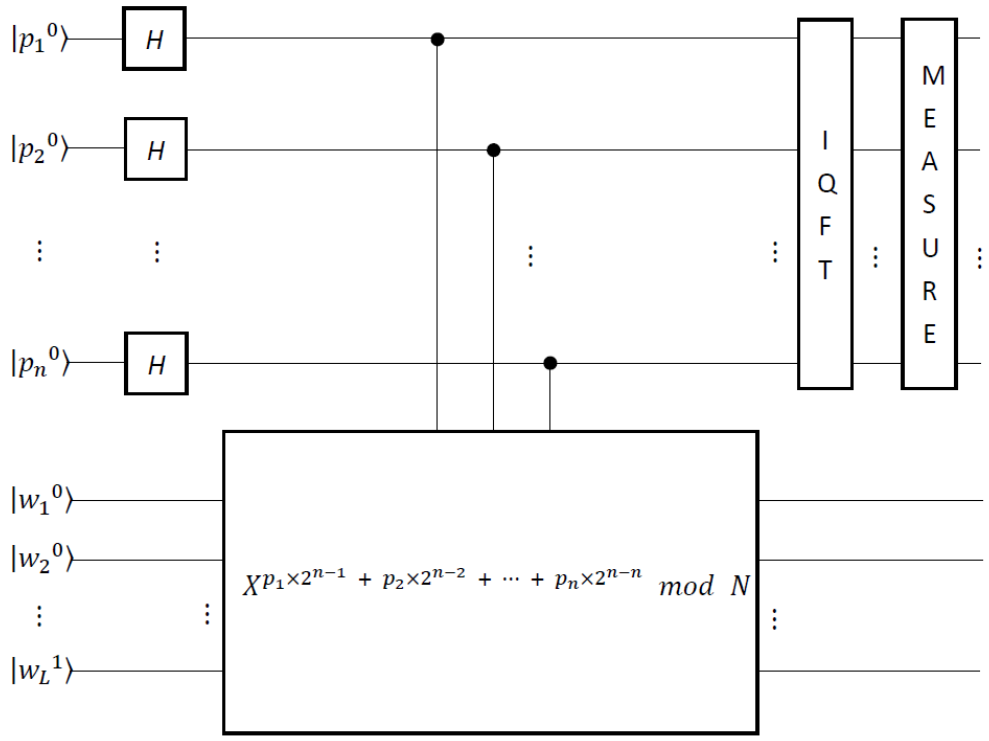

Figure 5.11: Quantum circuit of implementing Shor's order-finding algorithm.

and quantum bit $|p_n{}^0>$ is the least significant bit. Because the order of X modulo N is less than or equal to N, n is greater than or equal to $\log_2(N)$. Its decimal value is equal to $p_1 \times 2^{n-1} + p_2 \times 2^{n-2} + p_3 \times 2^{n-3} + \ldots + p_n \times 2^{n-n}$. The second quantum register of L quantum bits is $((\bigotimes_{y=1}^{L-1} |w_y^0>) \otimes (|w_L{}^1>))$. The initial state of each quantum bit in the front $(L-1)$ quantum bits is the |0> state. The initial state of the *least significant* quantum bit $(|w_L{}^1>)$ is the |1> state. Quantum bit $|w_1{}^0>$ is the most significant bit and quantum bit $|w_L{}^1>$ is the least significant bit. Its decimal value is equal to $w_1 \times 2^{L-1} + w_2 \times 2^{L-2} + w_3 \times 2^{L-3} + \ldots + p_L \times 2^{L-L}$. From Figure 5.11, the initial state vector is

$$|\varphi_0> = (\bigotimes_{k=1}^{n} |p_k^0>) \otimes ((\bigotimes_{y=1}^{L-1} |w_y^0>) \otimes (|w_L{}^1>)). \qquad (5.31)$$

From Figure 5.11, the initial state vector $|\varphi_0>$ in (5.31) is followed by n Hadamard gates on the first (upper) quantum register. This gives that the new state vector is

$$|\varphi_1\rangle = \frac{1}{\sqrt{2^n}} \left((\bigotimes_{k=1}^{n} |p_k^0\rangle + |p_k^1\rangle) \otimes ((\bigotimes_{y=1}^{L-1} |w_y^0\rangle) \otimes (|w_L{}^1\rangle))\right)$$

$$= \frac{1}{\sqrt{2^n}} \left(\sum_{P=0}^{2^n-1} |P\rangle\right) \otimes ((\bigotimes_{y=1}^{L-1} |w_y^0\rangle) \otimes (|w_L{}^1\rangle)). \quad (5.32)$$

Next, from Figure 5.11, the new state vector $|\varphi_1\rangle$ in (5.32) is followed by a quantum gate $|X^P \bmod N\rangle = |X^{p_1\times 2^{n-1} + p_2\times 2^{n-2} + \cdots + p_n\times 2^{n-n}} \bmod N\rangle$ operating on both quantum registers. This gives that the new state vector is

$$|\varphi_2\rangle = \frac{1}{\sqrt{2^n}} \left(\sum_{P=0}^{2^n-1} |P\rangle\right) |X^P \bmod N\rangle. \quad (5.33)$$

Because the order of X modulo N is r, terms of $|\varphi_2\rangle$ in (5.33) can be regrouped as r equivalent classes according to the computational basis states with the same remainder of $|X^P \bmod N\rangle$. The first equivalent class is $\{r \times y + 0 \mid 0 \le y \le \lfloor (2^n - 0) / r \rfloor\}$, where $\lfloor (2^n - 0) / r \rfloor$ is to obtain the greatest integer that is less than or equal to $((2^n - 0) / r)$. The second equivalent class is $\{r \times y + 1 \mid 0 \le y \le \lfloor (2^n - 1) / r \rfloor\}$. The third equivalent class is $\{r \times y + 2 \mid 0 \le y \le \lfloor (2^n - 2) / r \rfloor\}$. The fourth equivalent class is $\{r \times y + 3 \mid 0 \le y \le \lfloor (2^n - 3) / r \rfloor\}$. The rth equivalent class is $\{r \times y + (r - 1) \mid 0 \le y \le \lfloor (2^n - (r - 1)) / r \rfloor\}$. Not all the equivalent classes have the same number of elements. However, if r divides 2^n, then the number of elements in each equivalent class is the same. We assume that for $0 \le P \le (r - 1)$, $Y_P = \lfloor (2^n - P) / r \rfloor$. In light of the statements above, we rewrite the new state vector $|\varphi_2\rangle$ in (5.33) as follows

$$|\varphi_2\rangle = \sum_{P=0}^{r-1} \left(\frac{1}{\sqrt{2^n}} \sum_{y=0}^{Y_P} |r \times y + P\rangle\right) |X^P \bmod N\rangle. \quad (5.34)$$

For the convenience of our presentation, we assume that $|\varphi_{2P}\rangle = \left(\frac{1}{\sqrt{2^n}} \sum_{y=0}^{Y_P} |r \times y + P\rangle\right)$ for $0 \le P \le (r - 1)$.

As in Figure 5.11, the last step before measurement is to complete the inverse quantum Fourier transform (an **IQFT**) on the first (upper) quantum register. The superposition principle allows the unitary operator to act one by one on each $|\varphi_{2P}\rangle$. Therefore, we obtain the following new state vector

$$|\varphi_3\rangle = \frac{1}{\sqrt{2^n}} \sum_{P=0}^{r-1} \sum_{y=0}^{Y_P} \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times(y\times r+P)} |i\rangle |X^P \bmod N\rangle.$$

$$= \sum_{i=0}^{2^n-1} \sum_{P=0}^{r-1} \sum_{y=0}^{Y_P} \frac{e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times(y\times r+P)}}{2^n} \; |i\rangle |X^P \bmod N\rangle. \tag{5.35}$$

For the convenience of our presentation, we assume that $\varphi_{iP} = (\sum_{y=0}^{Y_P} \frac{e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times(y\times r+P)}}{2^n})$ for $0 \le P \le (r-1)$ and $0 \le i \le (2^n - 1)$. Coefficients φ_{iP} and $|\varphi_{iP}|^2$ subsequently represent the amplitude and the probability of measuring $|i\rangle |X^P \bmod N\rangle$ at the output of the circuit in Figure 5.11. The probability amplitudes may cancel each other while increasing the probability of measuring a suitable state.

For the convenience of our presentation, we assume that $\boldsymbol{P}(i)$ represents the probability of measuring $|i\rangle |X^P \bmod N\rangle$ at the output of the circuit in Figure 5.11. Basic probability theory guarantees that

$$\begin{aligned} \boldsymbol{P}(i) &= \sum_{P=0}^{r-1} |\varphi_{iP}|^2 = \sum_{P=0}^{r-1} |\sum_{y=0}^{Y_P} \frac{e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times(y\times r+P)}}{2^n}|^2 \\ &= \sum_{P=0}^{r-1} |e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times P}|^2 \times |\sum_{y=0}^{Y_P} \frac{(e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times r})^y}{2^n}|^2 \end{aligned} \tag{5.36}$$

Since basic probability theory ensures that $|e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times P}|^2 = (e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times P}) \times (e^{\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times P}) = 1$, we can rewrite $\boldsymbol{P}(i)$ in (5.36) as follows

$$\begin{aligned} \boldsymbol{P}(i) &= \sum_{P=0}^{r-1} 1 \times |\sum_{y=0}^{Y_P} \frac{(e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times r})^y}{2^n}|^2 \\ &= \sum_{P=0}^{r-1} |\frac{1}{2^n} \sum_{y=0}^{Y_P} (e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times r})^y|^2 \\ &= \frac{1}{2^{2\times n}} \times (\sum_{P=0}^{r-1} |\sum_{y=0}^{Y_P} (e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times r})^y|^2). \end{aligned} \tag{5.37}$$

For realizing a sum of geometrical sequence, we discuss the *ideal* case and the *practice* case. If the argument of the absolute value operator is $e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times r} = 1$, then $(i \times r / 2^n)$ is an integer and we can rewrite $\boldsymbol{P}(i)$ in (5.37) as follows

$$\boldsymbol{P}(i) = \frac{1}{2^{2\times n}} \times (\sum_{P=0}^{r-1} |\sum_{y=0}^{Y_P} 1^y|^2).$$

$$= \frac{1}{2^{2\times n}} \times (\sum_{P=0}^{r-1}(Y_P + 1)^2). \quad (5.38)$$

We call (5.38) as the *ideal* case. If the argument of the absolute value operator is $e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times r} \neq 1$, then $(i \times r / 2^n)$ is not an integer and we can rewrite $\boldsymbol{P}(i)$ in (5.37) as follows

$$\boldsymbol{P}(i) = \frac{1}{2^{2\times n}} \times (\sum_{P=0}^{r-1} \left| \frac{1-e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times r\times(1+Y_P)}}{1-e^{-\sqrt{-1}\times\frac{2\times\pi}{2^n}\times i\times r}} \right|^2). \quad (5.39)$$

We call (5.39) as the *practice* case. Because $|1 - e^{\sqrt{-1}\times\theta}|^2 = 4 \times \sin^2(\theta / 2)$ and $\sin(-\theta / 2) = -\sin(\theta / 2)$ and $\sin^2(-\theta / 2) = \sin(-\theta / 2) \times \sin(-\theta / 2) = (-\sin(\theta / 2)) \times (-\sin(\theta / 2)) = \sin^2(\theta / 2)$, we can rewrite $\boldsymbol{P}(i)$ in (5.39) as follows

$$\begin{aligned} \boldsymbol{P}(i) &= \frac{1}{2^{2\times n}} \times (\sum_{P=0}^{r-1} \frac{4\times\sin^2(\frac{-2\times\pi\times i\times r\times(Y_P+1)}{2^n}\times\frac{1}{2})}{4\times\sin^2(\frac{-2\times\pi\times i\times r}{2^n}\times\frac{1}{2})}) \\ &= \frac{1}{2^{2\times n}} \times (\sum_{P=0}^{r-1} \frac{\sin^2(\frac{\pi\times i\times r\times(Y_P+1)}{2^n})}{\sin^2(\frac{\pi\times i\times r}{2^n})}). \end{aligned} \quad (5.40)$$

5.14 Quantum Circuits of Factoring 15

We want to complete the prime factorization for $N = 15$. We need to find the nontrivial factor for $N = 15$. From **Lemma 5-1** and **Lemma 5-6**, we select a number $X = 2$ so that the greatest common divisor of $X = 2$ and $N = 15$ is 1 (one). This indicates that $X = 2$ is co-prime to $N = 15$. From **Lemma 5-6,** the order r of 2 modulo 15 satisfies $r \leq 15$. Since the number of bit representing $N = 15$ is four bits long, we also only need to make use of four bits that represent the value of r. If the value of r is an even, then the first nontrivial factor for $N = 15$ is equal to $\gcd(2^{r/2} + 1, N)$ and the second nontrivial factor for $N = 15$ is equal to $\gcd(2^{r/2} - 1, N)$.

Computing the order r of 2 modulo 15 is equivalent to calculate the period r of a given oracular function O_f: $\{p_1\ p_2\ p_3\ p_4 \mid \forall\ p_d \in \{0, 1\}$ for $1 \leq d \leq 4\} \rightarrow \{2^{p_1 p_2 p_3 p_4} \pmod{15} \mid \forall\ p_d \in \{0, 1\}$ for $1 \leq d \leq 4\}$. An input variable $p_1\ p_2\ p_3\ p_4$ is four bits long. Bit p_1 is the *most* significant bit and bit p_4 is the *least* significant bit. The corresponding decimal value is equal to $p_1 \times 2^{4-1} + p_2 \times 2^{4-2} + p_3 \times 2^{4-3} + p_4 \times 2^{4-4} = P$. The period r of O_f is to satisfy $O_f(p_1\ p_2\ p_3\ p_4) = O_f(p_1\ p_2\ p_3\ p_4 + r)$ to any two inputs $(p_1\ p_2\ p_3\ p_4)$ and $(p_1\ p_2\ p_3\ p_4 + r)$.

For implementing the operation of modular exponentiation, $2^{p_1p_2p_3p_4} \pmod{15}$, we assume that an auxiliary variable $w_1\ w_2\ w_3\ w_4$ is four bits long. Bit w_1 is the *most* significant bit and bit w_4 is the *least* significant bit. The corresponding decimal value is equal to $w_1 \times 2^{4-1} + w_2 \times 2^{4-2} + w_3 \times 2^{4-3} + w_4 \times 2^{4-4} = W$. The initial value of each bit in the front *three* bits is zero (0). The initial value of the *least significant* bit w_4 is one (1).

5.14.1 Flowchart of Computing the Order *r* of *X* Modulo *N*

Figure 5.12 is to flowchart of computing the order r of X modulo N. In Figure 5.12, in statement S_1, it sets the value of an auxiliary variable W to be one. It sets the value of X to be two and sets the value of N to be fifteen. Because the number of bits to represent N is four, it sets the value of an auxiliary variable n to be four. It sets the index

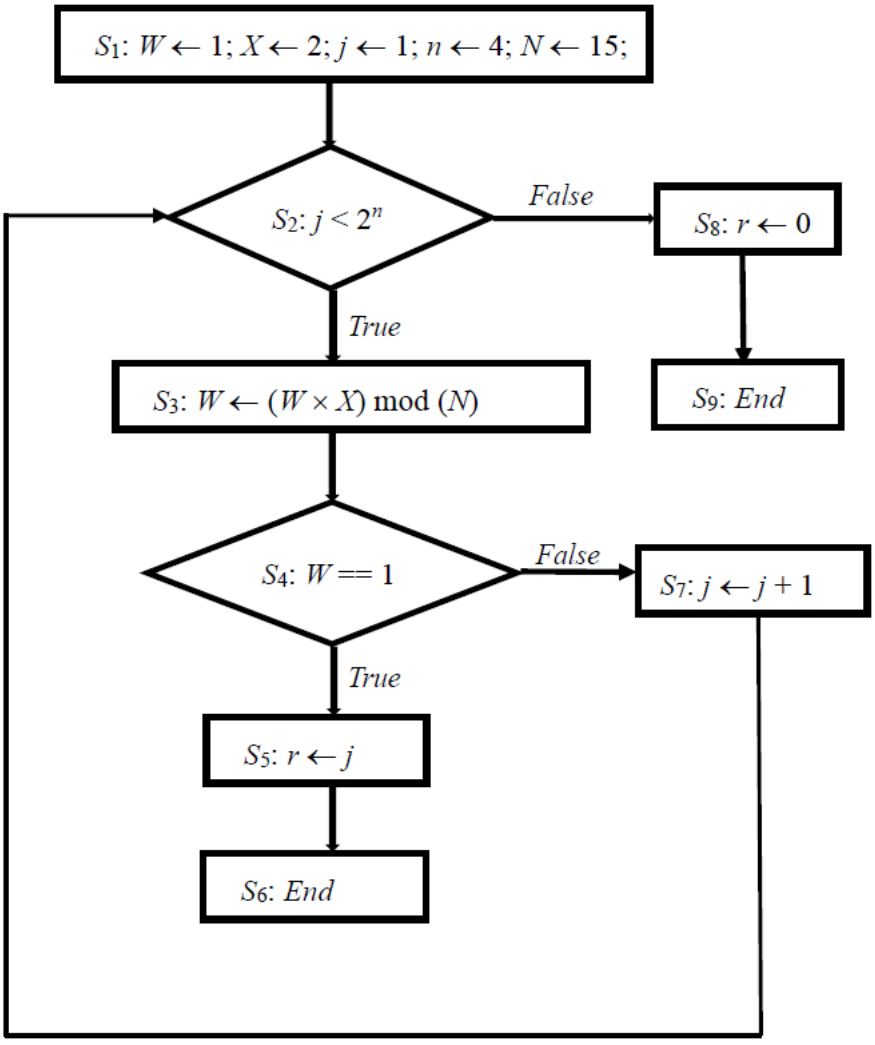

Figure 5.12: Flowchart of computing the order r of X modulo N.

variable j of the first loop to one. Next, in statement S_2, it executes the conditional judgement of the first loop. If the value of j is less than 2^n, then *next executed* instruction is statement S_3. Otherwise, in statement S_8, it sets the value of the order r to zero. This is to say that we cannot find the order r of X modulo N. Next, in statement S_9, it executes an *End* instruction to terminate the task that is to find the order r of X modulo N.

In statement S_3, it completes one multiplication instruction and one modular instruction and stores the result into an auxiliary variable W. On the j^{th} execution of statement S_3, it actually completes $X^j \pmod N$ and stores the result into an auxiliary variable W. Because the range of the value to the index variable j is from one through $2^n - 1$, it actually at most completes $X^1 \pmod N$ through $X^{2^n-1} \pmod N$. As $X^0 \pmod N = 1 \pmod N$, in statement S_1 it sets an auxiliary variable W to one that is to complete $X^0 \pmod N$.

Next, in statement S_4, it executes the conditional judgement to decide whether the value of W is equal to one. If the value of W is equal to one, then next executed instruction is statement S_5. In statement S_5, it sets the value of the order r to be the value of j. This indicates that we have found the order r of X modulo N. Next, in statement S_6, it executes an *End* instruction to terminate the task that is to find the order r of X modulo N.

However, if the value of W in statement S_4 is not equal to one, then next executed instruction is statement S_7. Next, in statement S_7, it increases the value of the index variable j. Repeat to execute statement S_2 through statement S_7 until in statement S_2, the conditional judgement becomes a *false* value or in statement S_6, it executes an *End* instruction to terminate the task. From Figure 5.12, the total number of multiplication instruction and modular instruction is at most $(2^n - 1)$ multiplication instructions and $(2^n - 1)$ modular instructions. This is to say that the cost of finding the order r of X modulo N is at most to complete $(2^n - 1)$ multiplication instructions and $(2^n - 1)$ modular instructions.

5.14.2 Implementing Modular Exponentiation $X^P \pmod N$

In Figure 5.12, in sequent model, it computes each modular exponentiation $X^P \pmod N$ to $0 \le P \le 2^n - 1$. However, in Figure 5.11, in parallel model, it simultaneously computes each modular exponentiation $X^P \pmod N$ to $0 \le P \le 2^n - 1$. In Figure 5.11, it

uses a quantum gate $|X^P \bmod N\rangle = |X^{p_1\times 2^{n-1} + p_2\times 2^{n-2} + \cdots + p_n\times 2^{n-n}} \bmod N\rangle$ operating on both quantum registers. The method for figuring out the modular exponentiation $X^P \pmod N$ has two stages. The first stage makes use of modular multiplication to compute $X^2 \pmod N$, by squaring $X \pmod N$. Next, it computes $X^4 \pmod N$ by squaring $X^2 \pmod N$. Then, it computes $X^8 \pmod N$ by squaring $X^4 \pmod N$. In this way, it continues to compute $X^{2^k} \pmod N$ for all k up to $(n - 1)$. We make use of $n = O(L)$, so a total of $(n - 1) = O(L)$ squaring operations is completed at a cost of $O(L^2)$ each (this cost assumes that the circuit used to do the squaring operation implements a kind of multiplication). Therefore, a total cost of the first stage is $O(L^3)$.

The second stage of the method is to complete the following observation

$$X^P \pmod N = X^{p_1\times 2^{n-1} + p_2\times 2^{n-2} + \cdots + p_n\times 2^{n-n}} \pmod N$$
$$= (X^{p_1\times 2^{n-1}} \pmod N) \times_N (X^{p_2\times 2^{n-2}} \pmod N) \times_N \ldots \times_N (X^{p_n\times 2^{n-n}} \pmod N). \quad (5.41)$$

Completing $(n - 1) = O(L)$ modular multiplications with a cost $O(L^2)$ each, we see that using $O(L^3)$ gates can compute this product in (5.41). This is sufficiently efficient for finding the order r of X modulo N. Of course, methods that are more efficient are possible if there are the circuits of the better multiplication.

5.14.3 Computing the Order *r* of (*X* = 2) Modulo (*N* = 15)

The *order* of $(X = 2)$ modulo $(N = 15)$ is to the *least* positive integer r such that $2^r = 1 \pmod{15}$. Because $r \le (N = 15)$ and the number of bits representing $(N = 15)$ is four bits long, the number of bits representing r is four bits long. Therefore, an input variable $p_1\ p_2\ p_3\ p_4$ is four bits long. Bit p_1 is the *most* significant bit and bit p_4 is the *least* significant bit. The corresponding decimal value is equal to $p_1 \times 2^{4-1} + p_2 \times 2^{4-2} + p_3 \times 2^{4-3} + p_4 \times 2^{4-4} = P$. This is to say that the range of the value to P is from zero through fifteen. If P is to the *least* positive integer such that $2^P = 1 \pmod{15}$, then the value of r is equal to the value of P. We use $p_d{}^0$ to represent the value of p_d to be zero for $1 \le d \le 4$ and apply $p_d{}^1$ to represent the value of p_d to be one for $1 \le d \le 4$.

Because the remainder for $2^P \pmod{15}$ to $0 \le P \le 15$ is from zero through fourteen, we assume that an auxiliary variable $w_1\ w_2\ w_3\ w_4$ is four bits long and we use it to store the result of computing $2^P \pmod{15}$ to $0 \le P \le 15$. We use $w_d{}^0$ to represent the value of w_d to be zero for $1 \le d \le 4$ and apply $w_d{}^1$ to represent the value of w_d to be one for $1 \le d \le 4$. Bit w_1 is the *most* significant bit and bit w_4 is the *least* significant bit. The corresponding decimal value is equal to $w_1 \times 2^{4-1} + w_2 \times 2^{4-2} + w_3 \times 2^{4-3} + w_4 \times 2^{4-}$

$^4 = W$. The initial value of each bit in the front *three* bits is zero. The initial value of the *least significant* bit w_4 is one. This is to say that $W = w_1^0 \times 2^{4-1} + w_2^0 \times 2^{4-2} + w_3^0 \times 2^{4-3} + w_4^1 \times 2^{4-4} = 1$.

Computing the order r of 2 modulo 15 is equivalent to calculate the period r of a given oracular function O_f: $\{p_1\ p_2\ p_3\ p_4 \mid \forall\ p_d \in \{0, 1\}$ for $1 \le d \le 4\} \rightarrow \{2^{p_1 p_2 p_3 p_4} \pmod{15} \mid \forall\ p_d \in \{0, 1\}$ for $1 \le d \le 4\}$. The period r of O_f is to satisfy $O_f(p_1\ p_2\ p_3\ p_4) = O_f(p_1\ p_2\ p_3\ p_4 + r)$ to any two inputs $(p_1\ p_2\ p_3\ p_4)$ and $(p_1\ p_2\ p_3\ p_4 + r)$. The first stage of computing $O_f(p_1\ p_2\ p_3\ p_4) = 2^{p_1 p_2 p_3 p_4} \pmod{15}$ is to use modular multiplication to compute $2^2 \pmod{15}$, by squaring 2 (mod 15). We get the following result

$$2^2 \pmod{15} = (2 \pmod{15})^2 = (2 \pmod{15}) \times_{15} (2 \pmod{15}) = 4 \pmod{15} = 4. \quad (5.42)$$

Next, it computes $2^4 \pmod{15}$ by squaring $2^2 \pmod{15}$ and we get the following result

$$2^4 \pmod{15} = (2^2 \pmod{15})^2 = (2^2 \pmod{15}) \times_{15} (2^2 \pmod{15}) = 4^2 \pmod{15} = 1. \quad (5.43)$$

Then, it computes $2^8 \pmod{15}$ by squaring $2^4 \pmod{15}$ and we get the following result

$$2^8 \pmod{15} = (2^4 \pmod{15})^2 = (2^4 \pmod{15}) \times_{15} (2^4 \pmod{15}) = 1^2 \pmod{15} = 1. \quad (5.44)$$

Next, the second stage of computing $O_f(p_1\ p_2\ p_3\ p_4) = 2^{p_1 p_2 p_3 p_4} \pmod{15}$ is to complete the following observation

$$2^P \pmod{15} = 2^{p_1 \times 2^3 + p_2 \times 2^2 + p_3 \times 2^1 + p_4 \times 2^0} \pmod{15} = (2^{p_1 \times 2^3} \pmod{15}) \times_{15} (2^{p_2 \times 2^2} \pmod{15}) \times_{15} (2^{p_3 \times 2^1} \pmod{15}) \times_{15} (2^{p_4 \times 2^0} \pmod{15}). \quad (5.45)$$

If the value of bit p_1 is equal to one (1), then $(2^{p_1{}^1 \times 2^3} \pmod{15}) = (2^8 \pmod{15}) = 1$. Otherwise, $(2^{p_1{}^0 \times 2^3} \pmod{15}) = (2^0 \pmod{15}) = 1$. This is to say that $(2^{p_1 \times 2^3} \pmod{15}) = 1$. Next, if the value of bit p_2 is equal to one (1), then $(2^{p_2{}^1 \times 2^2} \pmod{15}) = (2^4 \pmod{15}) = 1$. Otherwise, $(2^{p_2{}^0 \times 2^2} \pmod{15}) = (2^0 \pmod{15}) = 1$. This indicates that $(2^{p_2 \times 2^2} \pmod{15}) = 1$. Therefore, we can rewrite the equation in (5.45) as follows

$$2^P \pmod{15} = (2^{p_3 \times 2^1} \pmod{15}) \times_{15} (2^{p_4 \times 2^0} \pmod{15}). \quad (5.46)$$

According to the equation in (5.46), sixteen outputs of O_f that takes each input from $p_1^0\ p_2^0\ p_3^0\ p_4^0$ through $p_1^1 p_2^1\ p_3^1\ p_4^1$ are subsequently 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4 and 8. The binary values to one (1), two (2), four (4) and eight (8) are subsequently

$w_1^0 w_2^0\ w_3^0\ w_4^1$, $w_1^0 w_2^0\ w_3^1\ w_4^0$, $w_1^0 w_2^1\ w_3^0\ w_4^0$ and $w_1^1 w_2^0\ w_3^0\ w_4^0$. The frequency f of O_f is equal to the number of the period per sixteen outputs. This gives that $r \times f = 16$. Hidden patterns and information stored in a given oracular function O_f are to that its output rotates back to its starting value (1) *four* times. This implies that the number of the period per sixteen outputs is *four* and the frequency f of O_f is equal to four. The period r of O_f is the *reciprocal* of the frequency f of O_f. Thus, we obtain $r = \frac{1}{\frac{f}{16}} = \frac{16}{f} = \frac{16}{4} = 4$ and $r \times f = 4 \times 4 = 16$. Therefore, the order r of ($X = 2$) modulo ($N = 15$) is equal to four.

5.14.4 Reduction of Implementing Modular Exponentiation 2^P (mod 15)

In (5.46), the oracular function is $O_f(p_1\ p_2\ p_3\ p_4) = 2^{p_1 p_2 p_3 p_4} \pmod{15} = (2^{p_3 \times 2^1} \pmod{15}) \times_{15} (2^{p_4 \times 2^0} \pmod{15})$. If the value of bit p_3 is equal to one (1), then $(2^{p_3{}^1 \times 2^1} \pmod{15}) = (2^2 \pmod{15}) = 4$. Otherwise, $(2^{p_3{}^0 \times 2^1} \pmod{15}) = (2^0 \pmod{15}) = 1$. This is to say that if the value of bit p_3 is equal to one (1), then the instruction "$(2^{p_3{}^1 \times 2^1} \pmod{15}) = (2^2 \pmod{15}) = 4$" can be implemented by means of multiply any auxiliary variable $w_1^0 w_2^0\ w_3^0\ w_4^1$ by 2^2. Otherwise, the instruction "$(2^{p_3{}^0 \times 2^1} \pmod{15}) = (2^0 \pmod{15}) = 1$" are not implemented.

Similarly, if the value of bit p_4 is equal to one (1), then $(2^{p_4{}^1 \times 2^0} \pmod{15}) = (2^1 \pmod{15}) = 2$. Otherwise, $(2^{p_4{}^0 \times 2^0} \pmod{15}) = (2^0 \pmod{15}) = 1$. This indicates that if the value of bit p_4 is equal to one (1), then the instruction "$(2^{p_4{}^1 \times 2^0} \pmod{15}) = (2^1 \pmod{15}) = 2$" can be implemented by means of multiply the auxiliary variable $w_1^0 w_2^0\ w_3^0\ w_4^1$ by 2^1. Otherwise, the instruction "$(2^{p_4{}^0 \times 2^0} \pmod{15}) = (2^0 \pmod{15}) = 1$" are not executed.

Of course, using a simple bit shift can achieve multiplication by 2 (or indeed any power of 2) on any binary auxiliary variable. Computing 2^P requires P multiplications by 2 on any binary auxiliary variable. Completing left bit shift of P times can implement P multiplications by 2 on any binary auxiliary variable. This implies that computing 2^P requires completing left bit shift of P times on any binary auxiliary variable.

For example, in our example we use bits p_3 and p_4 as the controlled bits. If the value of the controlled bit p_4 is equal to one (1), then we use left bit shift of one time on the binary auxiliary variable $w_1^0 w_2^0\ w_3^0\ w_4^1$ to implement the instruction "$(2^{p_4{}^1 \times 2^0}$ (mod

15)) = (2^1 (mod 15)) = 2". Left bit shift of one time is to that it exchanges each bit w_k for $1 \le k \le 4$ with the next highest weighted position. On the execution of the first time, it exchanges bit $w_1{}^0$ with bit $w_2{}^0$ and the result is $w_2{}^0 w_1{}^0\ w_3{}^0\ w_4{}^1$. Next, on the execution of the second time, it exchanges bit $w_1{}^0$ with bit $w_3{}^0$ and the result is $w_2{}^0 w_3{}^0\ w_1{}^0\ w_4{}^1$. Next, on the execution of the third time, it exchanges bit $w_1{}^0$ with bit $w_4{}^1$ and the result is $w_2{}^0 w_3{}^0\ w_4{}^1\ w_1{}^0$.

The corresponding decimal value of $w_2{}^0 w_3{}^0\ w_4{}^1\ w_1{}^0$ is two (2). This means that it implement the instruction "($2^{p_4{}^1 \times 2^0}$ (mod 15)) = (2^1 (mod 15)) = 2". We can use three ***CSWAP*** gates to implement them. Similarly, if the value of the controlled bit p_3 is equal to one (1), then we complete a shift by two bits to implement the instruction "($2^{p_3{}^1 \times 2^1}$ (mod 15)) = (2^2 (mod 15)) = 4". A shift by two bits is to that it exchanges the bit at the weighted position (2^3) with another bit at the weighted position (2^1) and exchanges at the weighted position (2^2) with another bit at the weighted position (2^0). We can make use of two ***CSWAP*** gates to implement them. This reduction makes us not to implement multiplication circuits and modular circuits.

5.14.5 Initialize Quantum Registers of Quantum Circuits to Find the Order *r* of (*X* = 2) Modulo (*N* = 15)

We use the quantum circuit in Figure 5.13 to find the order r of ($X = 2$) modulo ($N = 15$). The first (upper) quantum register has four quantum bits. Bit $|p_1\rangle$ is the *most* significant bit and bit $|p_4\rangle$ is the *least* significant bit. The corresponding decimal value is equal to $p_1 \times 2^{4-1} + p_2 \times 2^{4-2} + p_3 \times 2^{4-3} + p_4 \times 2^{4-4} = P$. The initial state of each

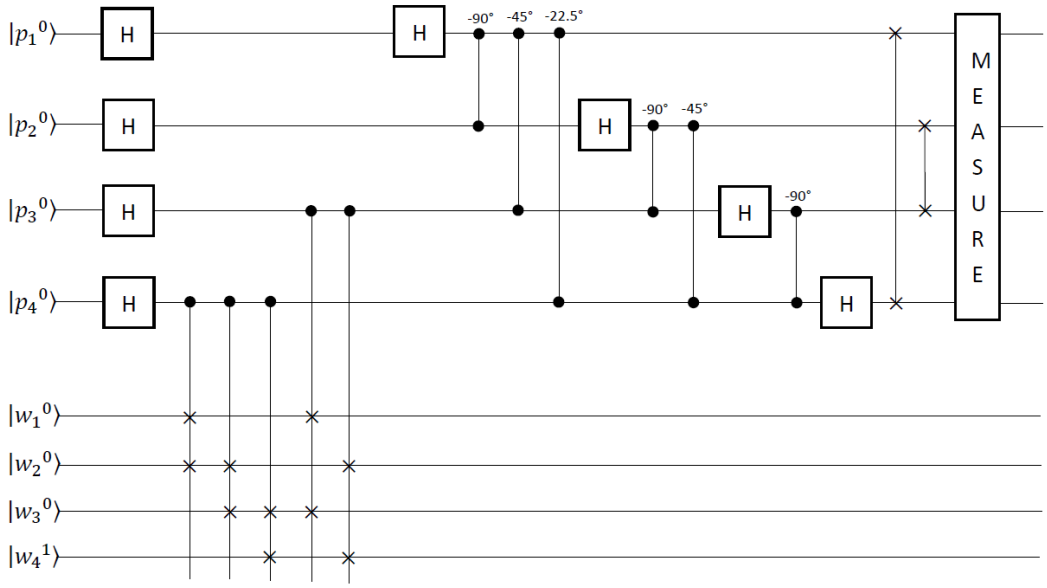

Figure 5.13: Quantum circuits of finding the order r of ($X = 2$) modulo ($N = 15$).

quantum bit $|p_d\rangle$ for $1 \le d \le 4$ is set to state $|0\rangle$. The second (lower) quantum register has four quantum bits. Bit $|w_1\rangle$ is the *most* significant bit and bit $|w_4\rangle$ is the *least* significant bit. The corresponding decimal value is equal to $w_1 \times 2^{4-1} + w_2 \times 2^{4-2} + w_3 \times 2^{4-3} + w_4 \times 2^{4-4} = W$. The initial state of the front three quantum bits $|w_d\rangle$ for $1 \le d \le 3$ is set to state $|0\rangle$. The initial state of the least significant quantum bits $|w_d\rangle$ for $1 \le d \le 3$ is set to state $|1\rangle$.

In Listing 5.2, the program is in the backend that is *simulator* of Open QASM with *thirty-two* quantum bits in **IBM**'s quantum computer. The program is to find the order r of ($X = 2$) modulo ($N = 15$). Figure 5.14 is the corresponding quantum circuit of the program in Listing 5.2 and is to implement the quantum circuit of finding the order r of ($X = 2$) modulo ($N = 15$) in Figure 5.13.

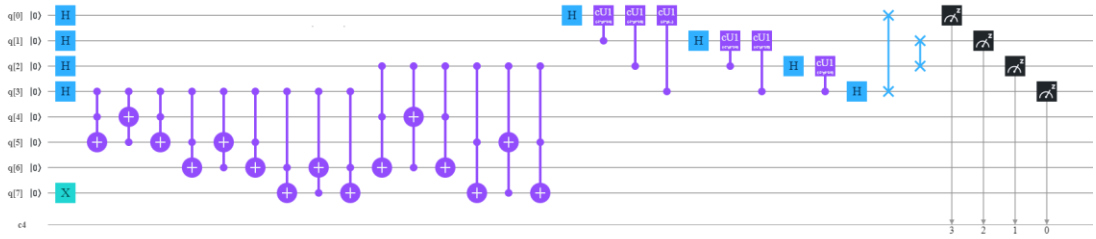

Figure 5.14: Implementing quantum circuits of finding the order r of ($X = 2$) modulo ($N = 15$) in Figure 5.13.

The statement "OPENQASM 2.0;" on line one of Listing 5.2 is to indicate that the program is written with version 2.0 of Open QASM. Then, the statement "include "qelib1.inc";" on line two of Listing 5.2 is to continue parsing the file "qelib1.inc" as if the contents of the file were pasted at the location of the include statement, where the file "qelib1.inc" is **Quantum Experience (QE) Standard Header** and the path is specified relative to the current working directory.

```
1. OPENQASM 2.0;
2. include "qelib1.inc";
3. qreg q[8];
4. creg c[4];
5. x q[7];
```

Listing 5.2: The program of finding the order r of ($X = 2$) modulo ($N = 15$).

Next, the statement "qreg q[8];" on line three of Listing 5.2 is to declare that in the program there are *eight* quantum bits. In the left top of Figure 5.14, eight quantum bits are subsequently q[0], q[1], q[2], q[3], q[4], q[5], q[6] and q[7]. The initial value of each quantum bit is set to state |0>. We use four quantum bits q[0], q[1], q[2] and q[3] to respectively encode four quantum bits $|p_1\rangle$, $|p_2\rangle$, $|p_3\rangle$ and $|p_4\rangle$ in Figure 5.13. We apply four quantum bits q[4], q[5], q[6] and q[7] to respectively encode four quantum bits $|w_1\rangle$, $|w_2\rangle$, $|w_3\rangle$ and $|w_4\rangle$ in Figure 5.13.

For the convenience of our explanation, $q[k]^0$ for $0 \le k \le 7$ is to represent the value 0 of q[k] and $q[k]^1$ for $0 \le k \le 7$ is to represent the value 1 of q[k]. Next, the statement "creg c[4];" on line four of Listing 5.2 is to declare that there are four classical bits in the program. In the left bottom of Figure 5.14, four classical bits are subsequently c[0], c[1], c[2] and c[3]. The initial value of each classical bit is set to zero (0). For the convenience of our explanation, $c[k]^0$ for $0 \le k \le 3$ is to represent the value 0 of c[k] and $c[k]^1$ for $0 \le k \le 3$ is to represent the value 1 of c[k]. The corresponding decimal value of the four initial classical bits $c[3]^0$ $c[2]^0$ $c[1]^0$ $c[0]^0$ is $2^3 \times c[3]^0 + 2^2 \times c[2]^0 + 2^1 \times c[1]^0 + 2^0 \times c[0]^0$. This indicates that classical bit $c[3]^0$ is the most significant bit and classical bit $c[0]^0$ is the least significant bit. Next, the statement "x q[7];" on line five of Listing 5.2 is to convert the state |0> of quantum bit |q[7]> into the state |1>. For the convenience of our explanation, an initial state vector of finding the order r of ($X = 2$) modulo ($N = 15$) is

$$|\Omega_0\rangle = |q[0]^0\rangle\,|q[1]^0\rangle\,|q[2]^0\rangle\,|q[3]^0\rangle\,|q[4]^0\rangle\,|q[5]^0\rangle\,|q[6]^0\rangle\,|q[7]^1\rangle. \qquad (5.47)$$

5.14.6 Quantum Superposition to Compute 2^P (mod 15)

In the initial state vector $|\Omega_0\rangle$ in (5.47), quantum bits $|q[0]^0\rangle$ $|q[1]^0\rangle$ $|q[2]^0\rangle$ $|q[3]^0\rangle$ encode four quantum bits $|p_1{}^0\rangle$, $|p_2{}^0\rangle$, $|p_3{}^0\rangle$ and $|p_4{}^0\rangle$ of the first register in Figure 5.13 and are the *precision* register. We use the precision register to represent the values of P that we shall pass to modular exponentiation 2^P (mod 15). We shall make use of quantum superposition to evaluate modular exponentiation 2^P (mod 15) for multiple values of P in parallel, so we use four statements "h q[0];", "h q[1];", "h q[2];" and "h q[3];" from line *six* through line *nine* in Listing 5.2 to place the *precision* register into

Listing 5.2 continued…

```
6.  h q[0];
7.  h q[1];
```

```
8. h q[2];
9. h q[3];
```

a superposition of all possible values. Therefore, we have the following new state vector

$$|\Omega_1\rangle = (\tfrac{1}{\sqrt{2}}\ (|q[0]^0\rangle + |q[0]^1\rangle))\ (\tfrac{1}{\sqrt{2}}\ (|q[1]^0\rangle + |q[1]^1\rangle))\ (\tfrac{1}{\sqrt{2}}\ (|q[2]^0\rangle + |q[2]^1\rangle))$$

$$(\tfrac{1}{\sqrt{2}}\ (|q[3]^0\rangle + |q[3]^1\rangle))\ (|q[4]^0\rangle\ |q[5]^0\rangle\ |q[6]^0\rangle\ |q[7]^1\rangle)$$

$$= \tfrac{1}{\sqrt{2^4}}\ (\textstyle\sum_{P=0}^{2^4-1}|P\rangle)\ (|q[4]^0\rangle\ |q[5]^0\rangle\ |q[6]^0\rangle\ |q[7]^1\rangle). \qquad (5.48)$$

This way makes each state ($|P\rangle\ |q[4]^0\rangle\ |q[5]^0\rangle\ |q[6]^0\rangle\ |q[7]^1\rangle$) in the new state vector ($|\Omega_1\rangle$) in (5.48) to be ready to be treated as a separate input to a parallel computation.

5.14.7 Implementing Conditional Multiply-by-2 for Computing 2^P (mod 15)

In the initial state vector $|\Omega_0\rangle$ in (5.47), quantum bits $|q[4]^0\rangle\ |q[5]^0\rangle\ |q[6]^0\rangle\ |q[7]^1\rangle$ encode four quantum bits $|w_1{}^0\rangle$, $|w_2{}^0\rangle$, $|w_3{}^0\rangle$ and $|w_4{}^1\rangle$ of the second register in Figure 5.13 and are the *auxiliary* register. We now would like to complete modular exponentiation 2^P (mod 15) = $(2^{p_3\times 2^1}\ (\text{mod } 15)) \times_{15} (2^{p_4\times 2^0}\ (\text{mod } 15))$ in (5.46) on the superposition of inputs we have within the precision register, and we shall apply the auxiliary register to hold and to store the results. We use three ***CSWAP*** gates to implement the instruction $(2^{p_4\times 2^0}\ (\text{mod } 15))$ that is conditional multiply-by-2. Three statements "ccx q[3],q[4],q[5];", "ccx q[3],q[5],q[4];" and "ccx q[3],q[4],q[5];" from line ten through line twelve in Listing 5.2 are three ***CCNOT*** gates to implement the *first* ***CSWAP*** gate in Figure 5.13. In the first ***CSWAP*** gate, quantum bit q[3] is its controlled bit and quantum bits q[4] and q[5] are its target bits. In the ***CCNOT*** instruction, the first operand and the second operand are its two controlled bits and the third operand is its target bit.

Listing 5.2 continued…

```
//Implement the first CSWAP gate.
10. ccx q[3],q[4],q[5];
11. ccx q[3],q[5],q[4];
12. ccx q[3],q[4],q[5];
```

```
//Implement the second CSWAP gate.
13. ccx q[3],q[5],q[6];
14. ccx q[3],q[6],q[5];
15. ccx q[3],q[5],q[6];

//Implement the third CSWAP gate.
16. ccx q[3],q[6],q[7];
17. ccx q[3],q[7],q[6];
18. ccx q[3],q[6],q[7];
```

The three ***CCNOT*** gates exchange the quantum bit at the weighted position (2^3) of the work register with the quantum bit at the weighted position (2^2) of the work register. This means that the new state vector is

$$|\Omega_2\rangle = \frac{1}{\sqrt{2^4}} (|q[0]^0\rangle + |q[0]^1\rangle) (|q[1]^0\rangle + |q[1]^1\rangle) (|q[2]^0\rangle + |q[2]^1\rangle) (|q[3]^0\rangle |q[4]^0\rangle |q[5]^0\rangle |q[6]^0\rangle |q[7]^1\rangle + |q[3]^1\rangle |q[5]^0\rangle |q[4]^0\rangle |q[6]^0\rangle |q[7]^1\rangle). \quad (5.49)$$

Next, in the second ***CSWAP*** gate in Figure 5.13, quantum bit q[3] is its controlled bit and quantum bits q[5] and q[6] are its target bits. Three ***CCNOT*** gates "ccx q[3],q[5],q[6];", "ccx q[3],q[6],q[5];" and "ccx q[3],q[5],q[6];" from line thirteen through line fifteen in Listing 5.2 are to implement the *second* ***CSWAP*** gate in Figure 5.13. This indicates that the new state vector is

$$|\Omega_3\rangle = \frac{1}{\sqrt{2^4}} (|q[0]^0\rangle + |q[0]^1\rangle) (|q[1]^0\rangle + |q[1]^1\rangle) (|q[2]^0\rangle + |q[2]^1\rangle) (|q[3]^0\rangle |q[4]^0\rangle |q[5]^0\rangle |q[6]^0\rangle |q[7]^1\rangle + |q[3]^1\rangle |q[5]^0\rangle |q[6]^0\rangle |q[4]^0\rangle |q[7]^1\rangle). \quad (5.50)$$

Then, in the third ***CSWAP*** gate in Figure 5.13, quantum bit q[3] is its controlled bit and quantum bits q[6] and q[7] are its target bits. Three ***CCNOT*** gates "ccx q[3],q[6],q[7];", "ccx q[3],q[7],q[6];" and "ccx q[3],q[6],q[7];" from line sixteen through line eighteen in Listing 5.2 are to implement the *third* ***CSWAP*** gate in Figure 5.13. This implies that the new state vector is

$$|\Omega_4\rangle = \frac{1}{\sqrt{2^4}} (|q[0]^0\rangle + |q[0]^1\rangle) (|q[1]^0\rangle + |q[1]^1\rangle) (|q[2]^0\rangle + |q[2]^1\rangle) (|q[3]^0\rangle |q[4]^0\rangle |q[5]^0\rangle |q[6]^0\rangle |q[7]^1\rangle + |q[3]^1\rangle |q[5]^0\rangle |q[6]^0\rangle |q[7]^1\rangle |q[4]^0\rangle). \quad (5.51)$$

In the new state vector $|\Omega_4>$ in (5.51), if the least significant quantum bit q[3] in the precision register is the |1> state, then the value of its work register is two (2). Otherwise, the value of its work register is not changed and is still one (1).

5.14.8 Implementing Conditional Multiply-by-4 for Computing 2^P (mod 15)

Because the work register in the new state vector $|\Omega_4>$ in (5.51) holds and stores the result of computing ($2^{p_4 \times 2^0}$ (mod 15)), next we want to complete the instruction ($2^{p_3 \times 2^1}$ (mod 15)). If the value of quantum bit |q[2]> that encodes quantum bit $|p_3>$ in the precision register is one (1), then that indicates to complete the instruction ($2^{p_3 \times 2^1}$ (mod 15)) to require another two multiplications by two (2) on the work register. We use two ***CSWAP*** gates to implement the instruction ($2^{p_3 \times 2^1}$ (mod 15)) that is conditional multiply-by-4. In Figure 5.13, the fourth ***CSWAP*** gate is to exchange the quantum bit at the weighted position (2^3) of the work register with another quantum bit at the weighted position (2^1) of the work register if the value of the quantum bit at the weighted position (2^1) of the precision register is state |1>. Next, in Figure 5.13, the fifth ***CSWAP*** gate is to exchange the quantum bit at the weighted position (2^2) of the work register with another quantum bit at the weighted position (2^0) of the work register if the value of the quantum bit at the weighted position (2^1) of the precision register is state |1>.

Three statements "ccx q[2],q[4],q[6];", "ccx q[2],q[6],q[4];" and "ccx q[2],q[4],q[6];" from line nineteen through line twenty-one in Listing 5.2 are three ***CCNOT*** gates to implement the *fourth* ***CSWAP*** gate in Figure 5.13. In the fourth ***CSWAP*** gate, quantum bit q[2] is its controlled bit and quantum bits q[4] and q[6] are its target bits. In the ***CCNOT*** instruction, the first operand and the second operand are its two controlled bits and the third operand is its target bit.

Listing 5.2 continued…

```
//Implement the fourth CSWAP gate.
19. ccx q[2],q[4],q[6];
20. ccx q[2],q[6],q[4];
21. ccx q[2],q[4],q[6];

//Implement the fifth CSWAP gate.
```

```
22. ccx q[2],q[5],q[7];
23. ccx q[2],q[7],q[5];
24. ccx q[2],q[5],q[7];
```

The three ***CCNOT*** gates exchange the quantum bit at the weighted position (2^3) of the work register with the quantum bit at the weighted position (2^1) of the work register. This means that the new state vector is

$$|\Omega_5\rangle = \frac{1}{\sqrt{2^4}} (|q[0]^0\rangle + |q[0]^1\rangle) (|q[1]^0\rangle + |q[1]^1\rangle) (|q[2]^0\rangle |q[3]^0\rangle |q[4]^0\rangle |q[5]^0\rangle |q[6]^0\rangle$$
$$|q[7]^1\rangle + |q[2]^0\rangle |q[3]^1\rangle |q[5]^0\rangle |q[6]^0\rangle |q[7]^1\rangle |q[4]^0\rangle + (|q[2]^1\rangle |q[3]^0\rangle |q[6]^0\rangle |q[5]^0\rangle$$
$$|q[4]^0\rangle |q[7]^1\rangle + |q[2]^1\rangle |q[3]^1\rangle |q[7]^1\rangle |q[6]^0\rangle |q[5]^0\rangle |q[4]^0\rangle). \quad (5.52)$$

Next, in the fifth ***CSWAP*** gate in Figure 5.13, quantum bit q[2] is its controlled bit and quantum bits q[5] and q[7] are its target bits. Three ***CCNOT*** gates "ccx q[2],q[5],q[7];", "ccx q[2],q[7],q[5];" and "ccx q[2],q[5],q[7];" from line twenty-two through line twenty-four in Listing 5.2 are to implement the *fifth* ***CSWAP*** gate in Figure 5.13. This means that the new state vector is

$$|\Omega_6\rangle = \frac{1}{\sqrt{2^4}} (|q[0]^0\rangle + |q[0]^1\rangle) (|q[1]^0\rangle + |q[1]^1\rangle) (|q[2]^0\rangle |q[3]^0\rangle |q[4]^0\rangle |q[5]^0\rangle |q[6]^0\rangle$$
$$|q[7]^1\rangle + |q[2]^0\rangle |q[3]^1\rangle |q[5]^0\rangle |q[6]^0\rangle |q[7]^1\rangle |q[4]^0\rangle + (|q[2]^1\rangle |q[3]^0\rangle |q[6]^0\rangle |q[7]^1\rangle$$
$$|q[4]^0\rangle |q[5]^0\rangle + |q[2]^1\rangle |q[3]^1\rangle |q[7]^1\rangle |q[4]^0\rangle |q[5]^0\rangle |q[6]^0\rangle). \quad (5.53)$$

In the new state vector $|\Omega_6\rangle$ in (5.53), it shows how we have now managed to compute $2^P \pmod{15} = (2^{p_3 \times 2^1} \pmod{15}) \times_{15} (2^{p_4 \times 2^0} \pmod{15})$ in (5.46) on every value of P from the precision register in superposition.

5.14.9 Implementing Inverse Quantum Fourier Transform of Four Quantum Bits

In Figure 5.13, by completing an inverse quantum Fourier transform on the precision register, it effectively transform the precision register state into a superposition of the *periodic* signal's component frequencies. Twelve statements from

Listing 5.2 continued…

```
// Implement an inverse quantum Fourier transform.
```

```
25. h q[0];
26. cu1(-2*pi*1/4) q[1],q[0];
27. cu1(-2*pi*1/8) q[2],q[0];
28. cu1(-2*pi*1/16) q[3],q[0];

29. h q[1];
30. cu1(-2*pi*1/4) q[2],q[1];
31. cu1(-2*pi*1/8) q[3],q[1];

32. h q[2];
33. cu1(-2*pi*1/4) q[3],q[2];

34. h q[3];
35. swap q[0],q[3];
36. swap q[1],q[2];
```

line twenty-five through line thirty-six in Listing 5.2 implement an inverse quantum Fourier transform on the precision register. They take the new state vector $|\Omega_6\rangle$ in (5.53) as their input state vector. They produce the following new state vector

$$|\Omega_7\rangle = (\frac{1}{\sqrt{2^2}}\ (|q[0]^0\rangle\,|q[1]^0\rangle\,|q[2]^0\rangle\,|q[3]^0\rangle) + \frac{1}{\sqrt{2^2}}\ (|q[0]^0\rangle\,|q[1]^1\rangle\,|q[2]^0\rangle\,|q[3]^0\rangle)$$

$$+ \frac{1}{\sqrt{2^2}}\ (|q[0]^1\rangle\,|q[1]^0\rangle\,|q[2]^0\rangle\,|q[3]^0\rangle) + \frac{1}{\sqrt{2^2}}\ (|q[0]^1\rangle\,|q[1]^1\rangle\,|q[2]^0\rangle\,|q[3]^0\rangle))$$

$$(\ |q[4]^0\rangle\,|q[5]^0\rangle\,|q[6]^0\rangle\,|q[7]^1\rangle + |q[5]^0\rangle\,|q[6]^0\rangle\,|q[7]^1\rangle\,|q[4]^0\rangle + |q[6]^0\rangle\,|q[7]^1\rangle$$
$$|q[4]^0\rangle\,|q[5]^0\rangle + |q[7]^1\rangle\,|q[4]^0\rangle\,|q[5]^0\rangle\,|q[6]^0\rangle). \qquad (5.54)$$

5.14.10 Read the Quantum Result

Finally, four statements “measure q[0] -> c[3];”, “measure q[1] -> c[2];”, “measure q[2] -> c[1];” and “measure q[3] -> c[0];” from line thirty-seven through line forty in Listing 5.2 implement the measurement on the precision register in Figure 5.13. They measure four quantum bits q[0], q[1], q[2] and q[3] of the precision register. They record the measurement outcome by overwriting four classical bits c[3], c[2], c[1] and c[0].

Listing 5.2 continued…

```
// Implement one measurement on the precision register
37. measure q[0] -> c[3];
38. measure q[1] -> c[2];
39. measure q[2] -> c[1];
40. measure q[3] -> c[0];
```

In the backend *simulator* with thirty-two quantum bits in **IBM**'s quantum computers, we use the command "run" to execute the program in Listing 5.2. Figure 5.15 shows the measured result. From Figure 5.15, we obtain that a computational basis state 0000 (c[3] = 0 = q[0] = |0>, c[2] = 0 = q[1] = |0>, c[1] = 0 = q[2] = |0> and c[0] = 0 = q[3] = |0>) has the probability 24.512% (0.24512). On the other hand, we get that a computational basis state 0100 (c[3] = 0 = q[0] = |0>, c[2] = 1 = q[1] = |1>, c[1] = 0 = q[2] = |0> and c[0] = 0 = q[3] = |0>) has the probability 23.145% (0.23145). Alternatively, we gain that a computational basis state 1000 (c[3] = 1 = q[0] = |1>, c[2] = 0 = q[1] = |0>, c[1] = 0 = q[2] = |0> and c[0] = 0 = q[3] = |0>) has the probability 27.148% (0.27148). On the other hand, we obtain that a computational basis state 1100 (c[3] = 1 = q[0] = |1>, c[2] = 1 = q[1] = |1>, c[1] = 0 = q[2] = |0> and c[0] = 0 = q[3] = |0>) has the probability 25.195% (0.25195).

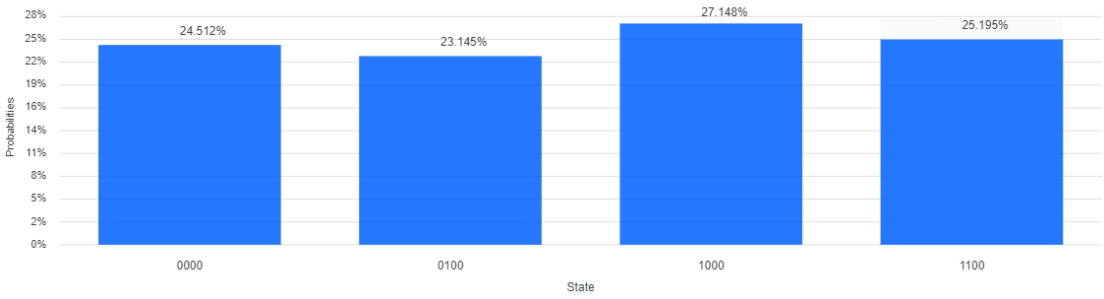

Figure 5.15: A computational basis state 0000 has the probability 24.512% (0.24512), a computational basis state 0100 has the probability 23.145% (0.23145), a computational basis state 1000 has the probability 27.148% (0.27148) and a computational basis state 1100 has the probability 25.195% (0.25195).

We select the computational basis state 0100 (c[3] = 0 = q[0] = |0>, c[2] = 1 = q[1] = |1>, c[1] = 0 = q[2] = |0> and c[0] = 0 = q[3] = |0>) with the probability 23.145% (0.23145) as the measured result. Because the decimal value of the computational basis state 1000 is four (4) and (2^4 / 4) is a rational number, we use the continued fractional algorithm in Figure 5.2 to determine the continued fractional representation of (c / d) if $c = 2^4 = 16$ and $d = 4$ and the corresponding convergent. From the first execution of

statement S_0 through statement S_2, it gets $i = 1$, $q[1] = c / d = 16 / 4 = 4$ and $r = 16 \pmod 4) = 0$. This is to split (16 / 4) into its integer and fractional part and not to invert its fractional part,

$$\frac{16}{4} = 4 + \frac{0}{4} = 4. \tag{5.55}$$

Since the value of r is equal to 0, from the first execution of statement S_3, it returns to a *true*. Thus, next, from the first execution of statement S_4, the answer is to the continued fractional representation of (16 / 4)

$$\frac{16}{4} = (q[1] = 4) = 4. \tag{5.56}$$

Next, from the first execution of Statement S_5, it terminates the execution of the continued fractional algorithm. For a rational number (16 / 4), the first convergent is $(q[1]) = 4 = \frac{4}{1}$ and is the closest one to $(\frac{16}{4})$ with numerator less than 15. Therefore, we check $2^4 \pmod{15}$ which equals one (1) and we find the order r to be four (4). Because the order r is even, from **Lemma 5-2**, we use Euclid's algorithm to compute gcd(15, $2^{\frac{4}{2}} + 1$) and gcd(15, $2^{\frac{4}{2}} - 1$). This implies that two nontrivial factors for $N = 15$ are respectively 5 and 3. Therefore, the prime factorization for $N = 15$ is $N = 5 \times 3$.

5.15 Assessment to Complexity of Shor's Order-Finding Algorithm

In Figure 5.11, the precision register (the first register or the upper register) with n quantum bits represents the order r of X modulo N. Because the value of r is less than or equal to the value of N, the value of n is the smallest integer greater than or equal to $\log_2 N$ and we can write it as $n = \lceil \log_2 N \rceil$. In Figure 5.11, the work register (the second register or the lower register) with L quantum bits is to store the result of computing $X^P \pmod N$ for $0 \le p \le 2^n - 1$. Because the result of computing $X^P \pmod N$ for $0 \le p \le 2^n - 1$ is less than the value of N, the value of L is the smallest integer greater than or equal to $\log_2 N$ and we can write it as $L = \lceil \log_2 N \rceil$. The value of X is less than the value of N, so we can use L bits to represent the value of X and the value of N.

In Figure 5.11, in parallel model, it simultaneously computes each modular exponentiation $X^P \pmod N$ to $0 \le P \le 2^n - 1$. In Figure 5.11, it applies a quantum gate

$|X^P \bmod N\rangle = |X^{p_1\times 2^{n-1} + p_2\times 2^{n-2} + \cdots + p_n\times 2^{n-n}} \bmod N\rangle$ operating on the precision register and the work register. The method to calculate the modular exponentiation X^P (mod N) contains two stages. The first stage uses modular multiplication to compute X^2 (mod N), by squaring X (mod N). Then, it figures out X^4 (mod N) by squaring X^2 (mod N). Then, it calculates X^8 (mod N) by squaring X^4 (mod N). In this way, it continues to figure out X^{2^k} (mod N) for all k up to $(n - 1)$.

Since $n = \lceil \log_2 N \rceil$ and $L = \lceil \log_2 N \rceil$, we apply $n = O(L)$. Therefore, in the first stage it completes a total of $(n - 1) = O(L)$ squaring operations. A squaring operation consists of a multiplication instruction and a modular instruction. Multiplicand and multiplier in the multiplication instruction are both X and they are L bits long. Product in the multiplication instruction is $(2 \times L)$ bits long and it is dividend in the modular instruction. Divisor in the modular instruction is N and it is $n = O(L)$ bits long. The number of the auxiliary carry bit in the multiplication instruction is $(2 \times L + 1)$ bits and the number of the auxiliary borrow bit in the modular instruction is $(2 \times L + 1)$ bits.

Because the cost of the circuit to implement one multiplication instruction is $O(L^2)$ digital logic gates and the cost of the circuit to implement one modular instruction is $O(L^2)$ digital logic gates, the cost of the quantum circuit to implement one squaring operation is $O(L^2)$ quantum gates. Therefore, the cost to complete $(n - 1) = O(L)$ squaring operations is $O(L^3)$ quantum gates and a total cost of implementing the first stage is $O(L^3)$ quantum gates.

Next, the second stage of the method is to complete X^P (mod N) = ($X^{p_1\times 2^{n-1}}$ (mod N)) $\times_N$ ($X^{p_2\times 2^{n-2}}$ (mod N)) $\times_N \ldots \times_N$ ($X^{p_n\times 2^{n-n}}$ (mod N)). It implements $(n - 1) = O(L)$ modular multiplications. Each modular multiplication consists of a multiplication instruction and a modular instruction. Multiplicand and multiplier in the multiplication instruction are L bits long. Product in the multiplication instruction is $(2 \times L)$ bits long and it is dividend in the modular instruction. Divisor in the modular instruction is N and it is $n = O(L)$ bits long. The number of the auxiliary carry bit in the multiplication instruction is $(2 \times L + 1)$ bits and the number of the auxiliary borrow bit in the modular instruction is $(2 \times L + 1)$ bits.

Since the cost of the circuit to implement one multiplication instruction is $O(L^2)$ digital logic gates and the cost of the circuit to implement one modular instruction is $O(L^2)$ digital logic gates, the cost of the quantum circuit to implement one modular multiplication is $O(L^2)$ quantum gates. Hence, the cost to complete $(n - 1) = O(L)$ modular multiplication is $O(L^3)$ quantum gates and a total cost of implementing the

second stage is $O(L^3)$ quantum gates. This means that the cost of the quantum circuit to implement $|X^P \bmod N\rangle = |X^{p_1\times 2^{n-1} + p_2\times 2^{n-2} + \cdots + p_n\times 2^{n-n}} \bmod N\rangle$ operating on the precision register and the work register is $O(L^3)$ quantum gates.

Next, in Figure 5.11, it completes one inverse quantum Fourier transform. The cost of implementing one inverse quantum Fourier transform is $O(L^2)$ quantum gates. Finally, in Figure 5.11, it completes a measurement on the precision register. Thus, in Shor's order-finding algorithm, the cost to compute the order r of X modulo N is $O(L)$ quantum bits and $O(L^3)$ quantum gates.

5.16 Summary

In this chapter, we gave an introduction of fundamental number theory. Next, we described **Euclid's** algorithm. We also introduced quadratic congruence. We then illustrated continued fractions. We also introduced the two problems of order finding and factoring. Next, we described how to compute the order of 2 modulo 15 and the prime factorization for 15. We also illustrated how to calculate the order of 2 modulo 21 and the prime factorization for 21. Next, we introduced how to calculate the order of 2 modulo 35 and the prime factorization for 35. We also introduced how to figure out the order of 5 modulo 33 and the prime factorization for 33. We then described the possibility of finding the even order of X modulo N. We also illustrated public key cryptography and the RSA cryptosystem. Next, we introduced how to implement the ***controlled-swap*** gate of three quantum bits. We also described Shor's order-finding algorithm. We then illustrated how to design quantum circuits of factoring 15. We also gave assessment of complexity of Shor's order-finding algorithm.

5.17 Bibliographical Notes

In this chapter for more details about an introduction of fundamental and advanced knowledge of number theory, the recommended books are [Hardy and Wright 1979; Nielsen and Chuang 2000; Imre and Balazs 2005; Lipton and Regan 2014; Silva 2018; Johnston et al 2019]. For a more detailed description to Shor's order-finding algorithm, the recommended article and books are [Shor 1994; Nielsen and Chuang 2000; Imre and Balazs 2005; Lipton and Regan 2014; Silva 2018; Johnston et al 2019]. A good introduction to the instructions of Open QASM is the famous article in [Cross et al 2017].

5.18 Exercises

5.1 Let c and d be integer, and let r be the remainder when c is divided by d. Then provided $r \neq 0$, please prove the equation $\gcd(c, d) = \gcd(d, r)$.

5.2 (**Chinese remainder theorem**) we assume that $y_1, \ldots, y_n$ are positive integers such that any pair y_i and y_j ($i \neq j$) are co-prime. Then the system of equations

$$\begin{aligned} z &= c_1 \pmod{y_1} \\ z &= c_2 \pmod{y_2} \\ &\ldots \\ z &= c_n \pmod{y_n} \end{aligned}$$

has a solution. Moreover, any two solutions to this system of equations are equal modulo $Y = y_1\, y_2 \ldots y_n$. Please prove **Chinese remainder theorem**.

5.3 We assume that p and k are integers and p is a prime in the range 1 to $p - 1$. Then prime p divides $\binom{p}{k}$. Please prove it.

5.4 (**Fermat's little theorem**) we assume that p is a prime, and a is an integer. Then $a^p = a \pmod{p}$. If integer a is not divisible by prime p then $a^{p-1} = 1 \pmod{p}$. Please prove them.

5.5 The *Euler* function $\varphi(n)$ is defined to be the number of positive integers which are less than n and are co-prime to n. We assume that a is co-prime to n. Then $a^{\varphi(n)} = 1 \pmod{n}$.

5.6 If $(i / 2^n)$ is a rational fraction and z and r are positive integers that satisfy $|(z / r) - (i / 2^n)| \leq (1 / (2 \times r^2))$, then (z / r) is a convergent of the continued fraction of $(i / 2^n)$.

5.7 Prove that $|1 - e^{\sqrt{-1} \times \theta}|^2 = 4 \times \sin^2(\theta / 2)$.