It has 2 parts.
Classical and Quantum.

Classical: How to factor a number?
Let’s say we want to factorize N.
Let’s take a random number a which is less than N.
For e.g.
N = 21, a = 2
Now keep computing a^x mod N until it reaches 1.
Once it reaches 1, then it will start repeating, hence that will give us the period of the function.

2^1 mod 21 = 2
2^2 mod 21 = 4
2^3 mod 21 = 8
2^4 mod 21 = 16
2^5 mod 21 = 11
2^6 mod 21 = 1

So period r = 6.
If r is odd, repeat with a different ‘a’ until you find an even ‘r’.
Once we have an even r, it’s highly likely that factors of N are:
gcd(N,a^(r/2) - 1) and gcd(N,a^(r/2) + 1)
a^(r/2) = 2^3 = 8
gcd(21,8-1) = gcd(21,7) = 7
gcd(21,8+1) = gcd(21,9) = 3
So factors are 7,3.

Why does this work?
a^r = 1 mod N
=> a^r - 1 = 0 mod N
=> a^r - 1 = k.N = (a^(r/2) - 1).(a^(r/2) + 1) since r is even.
Let a^(r/2) - 1 = t1 and a^(r/2) + 1 = t2.
So t1.t2 = k.N
Now, ideally N shouldn’t divide t1 or t2 entirely.
Else gcd(t1,N) or gcd(t2,N) will simply return N and we are back to square 1.

In our case:
7*9 = 3*21
So factors of 21 which are 3,7 are distributed across both terms(7 and 9), and it works as intended.

Quantum Part:
Coming to the quantum part. Had it been a classical computer, here is what we would do:
1. Write a function which takes inputs as a,N and try to find ‘r’ using loops. If that doesn’t work,

try with another ‘a’ in an outer loop.

But in quantum, we hardcode ‘a’ and ‘N’ in the circuit and if that doesn’t work, we recompile the circuit with different a,N and rerun the algorithm.

So ‘a’,’N’ are part of the circuit architecture, and not some sort of input which we give it.

Though we have 2 input registers here.

Register 1 is for holding all the possible values of ‘x’, i.e. the exponent of ‘a’. And it should have enough qbits to hold all values of x such that a^x = N^2. In our case where N = 21 and a = 2 we need x to be 9 so that it can hold 441(21^2) and 2^8 is only 256.

Register 2 is for holding all possible values of the computation a^x mod N which can be from 0 to N-1.
So here 2^5 = 32 > 21 so 5 qubits for Register 2.

Here is the precise reason why we need N^2 states:
It is not about the value of a^x.
It is about Resolution (Precision).
We are trying to find the period ‘r’.
To find ‘r’ accurately using the Quantum Fourier Transform (QFT), we need to sample the function over a domain that is much larger than the period itself.
The math of continued fractions (which we use in the final step) requires the total number of states (Q) to be roughly N^2 (specifically N^2 <= Q < 2N^2).
If we used fewer qubits (say, just enough to cover N), the "peaks" in our quantum probability result would be too blurry/wide, and we might fail to recover the correct period.

Modular exponentiation:
a = 2, N = 21,M=2^9 = 512
Now we first apply Hadamard gate on Register 1. So Register 1 holds all the values from 0 to M-1 in superposition. Next, we apply a gate which computes a^x mod N and output goes to Register 2. So register 2 holds all the values resulting from the computation.
In our case those values will be: 2,4,8,16,11,1

So the state of our quantum computer is now a giant superposition of pairs:
|0>|1> + |1>|2> + |2>|4> + |3>|8> + |4>|16> + |5>|11> + |6>|1> + …… + |511>|2>

And now we have “entanglement” between Register 1 and Register 2 which simply means that their states are tightly coupled. If we measure Register 1 right now, the wavefunction will collapse and give us the linked value in Register 2 as well.

Finding the period. Quantum Fourier Transform.
Now, we come back to what we wanted to do in the beginning. Finding the period ‘r’.

The collapse:
Let's say we measure Register 2 now and we find the value 2, Register 1 will collapse and hold the values 1,7,13 etc. So Register 1 is now a superposition of all those 'x' values which produced the remainder 2 when divided by 21.
|1> + |7> + |13> …
So the period r = 6 is now encoded in this pattern.
Now if we measure Register 1 right now we will get a single value like 1,7,13 etc and not the period r = 6.
To solve this we use **Quantum Fourier Transform(QFT).**

The **QFT** is like a prism. If you shine a light wave made of a specific repeating frequency into a prism, it splits it and points to exactly what that frequency is. Similarly, the QFT takes a state with repeating spikes (like 1,7,13,19) and transforms it into a state where the probability peaks at multiples of the frequency.

Basically, the QFT transforms the period (spacing in data) into a frequency (measurable peaks).

Final measurement:
If we apply QFT to Register 1 and measure the output we get a number 'y' such that:
y/M = k/r
Where:
y is the number we just measured.
M is the total size of Register 1(in our case 512)
k is some random integer.
r is the period we are desperate to find.
So
y = kM/r => y will be a multiple of 512/6 = 85.33
But y has to be an integer, so it will be 85 or 86(probability will split between these two),170 or 171, 256….
Let's say we got y = 85
So y/M = 85/512 = 0.16…
Now we use this to find 'r'.
We(a classical computer) try some simple fractions for k/r = ¼,⅕, ⅙ etc…
We notice that ⅙ is very close to y/M so r = 6 could be a possible solution.
So we compute a^x mod N and see that it does indeed repeat in cycles of 6.
So r = 6 is the correct answer.

Had we got any other y, we wouldn't get a correct r and we will redo the algorithm run again.

So, this is how we find 'r' using Shor's algorithm.