```
import QuantumRingsLib
from QuantumRingsLib import QuantumRegister, ClassicalRegister, QuantumCircuit
from QuantumRingsLib import QuantumRingsProvider, job_monitor
from QuantumRingsLib import JobStatus
import numpy as np
import math
from fractions import Fraction

def gcd(a, b):
    """Compute the greatest common divisor (GCD) using Euclidean algorithm."""
    while b:
        a, b = b, a % b
    return a

def qft(qc, q, n):
    """Applies the Quantum Fourier Transform to the quantum register."""
    for i in range(n):
        qc.h(q[i])
        for j in range(i + 1, n):
            qc.cu1(np.pi / float(2 ** (j - i)), q[j], q[i])

def iqft(qc, q, n):
    """Implements the Inverse Quantum Fourier Transform (IQFT)."""
    for i in range(n - 1, -1, -1):
        qc.h(q[i])
        for j in range(i):
            qc.cu1(-np.pi / float(2 ** (i - j)), q[j], q[i])

def quantum_period_finding(N, a):
    """Implements quantum phase estimation to find period r of a^r mod N."""
    num_qubits = int(math.log2(N)) + 2  # Enough qubits for accuracy
    q = QuantumRegister(num_qubits, 'q')
    c = ClassicalRegister(num_qubits - 1, 'c')
    qc = QuantumCircuit(q, c)

    # Apply Hadamard to the first qubits
    for i in range(num_qubits - 1):
        qc.h(q[i])
 # Modular exponentiation: simulate a^x mod N (simplified here, should be more robust)
    qc.x(q[num_qubits - 1])
    qc.barrier()

    # Apply QFT instead of just inverse QFT
    qft(qc, q, num_qubits - 1)
```

```
    # Perform Inverse QFT to extract period r
    iqft(qc, q, num_qubits - 1)

    # Measure all qubits except the last one
    for i in range(num_qubits - 1):
        qc.measure(q[i], c[i])

    return qc

def run_shors(N, a):
    """Runs Shor’s Algorithm on Quantum Rings to factor N."""
    provider = QuantumRingsProvider(
        token='rings-200.sjCanK2skqL4YIzImD4QLXGtqRZuXAzR',
        name='jessicaryuzaki@gmail.com'
    )
    backend = provider.get_backend("scarlet_quantum_rings")

    # Build quantum circuit
    qc = quantum_period_finding(N, a)

    # Execute the circuit
    job = backend.run(qc, shots=1024)
    job_monitor(job)
    result = job.result()
    counts = result.get_counts()
  # Extract the most probable measurement outcome
    measured_value = max(counts, key=counts.get)  # Binary string result
    decimal_value = int(measured_value, 2)

    # Convert to fraction to estimate period r
    r = Fraction(decimal_value, 2**(qc.num_qubits - 1)).denominator
    print(f"Estimated period r: {r}")

    if r % 2 == 1:
        print("Odd period found, retry with a different a")
        return None

    # Classical post-processing
    factor1 = gcd(N, pow(a, r // 2, N) - 1)
    factor2 = N // factor1

    if factor1 * factor2 == N:
        print(f"Factors of {N}: {factor1}, {factor2}")
```

```
        return factor1, factor2
    else:
        print("Failed to factorize, retry with different a")
        return None

# Example run (Factorizing a semiprime N = 143 using base 7)
run_shors(N=345, a=6)  # Replace with different semiprimes and a values to test
```