Objective:

The goal is to clean and improve the randomness quality of QRNG (Quantum Random Number Generator) data generated from IBM's quantum computers. This is achieved by applying preprocessing and post-processing techniques to maximize entropy and ensure the generated numbers are truly random. Tests of randomness such as the DFT test and the Linear Complexity test are used to verify the quality of the randomness.

---

1. Discrete Fourier Transform (DFT) Test

Purpose:

The DFT test checks for the presence of periodic structures within the bit strings. Periodicity in random data indicates a lack of randomness, which undermines the utility of quantum random numbers for applications like cryptography. The DFT test is based on Fourier analysis, which transforms a sequence into its frequency domain to detect any repeating patterns.

Implementation:

The code uses the Fast Fourier Transform (FFT) algorithm to analyze the frequency components of the QRNG bit strings. It then calculates the number of peaks above a certain threshold and compares it to the expected number for truly random data.

```
# DFT Test Implementation
def dft_test(bit_strings):
    n = len(bit_strings)
    fft_result = np.abs(fft.fft(bit_strings))[:n // 2]  # Use only
half of the FFT output (real part)

    # Threshold for peaks
    threshold = np.sqrt(np.log(1 / 0.05) * n)

    # Compute expected number of peaks for random data
    expected_peaks = 0.95 * (n / 2)

    # Check if the sequence passes the DFT test
    passes_dft = np.sum(fft_result > threshold) <= expected_peaks
    return passes_dft, fft_result
```

Passing Criteria:

- **Pass**: If the number of peaks in the FFT result is below the expected threshold, the data passes the DFT test, indicating that the bit strings do not exhibit significant periodicity and can be considered random.

Results:

In this case, the QRNG data **passed the DFT test**.

```
print(f"DFT Test Passed: {passes_dft}")
```

Interpretation:

The DFT test confirms that the QRNG bit strings do not contain periodic patterns, which is an important property for randomness. Passing this test ensures that the QRNG data can be used in sensitive applications where periodicity could undermine security, such as encryption.

---

2. Linear Complexity Test

Purpose:

The linear complexity test evaluates the unpredictability of a sequence by determining the minimum length of a linear feedback shift register (LFSR) that can generate the sequence. A high linear complexity indicates that the sequence is difficult to predict, which is desirable for cryptographic randomness. Low complexity implies that the sequence is predictable and not truly random.

Implementation:

The code uses the **Berlekamp-Massey algorithm** to compute the linear complexity of each bit string. This algorithm determines how many previous bits are required to predict the next one in the sequence.

```
# Linear Complexity Test Implementation
def berlekamp_massey_algorithm(bit_string):
    n = len(bit_string)
    c = [0] * n
    b = [0] * n
    c[0], b[0] = 1, 1
    l, m, i = 0, -1, 0

    for n in range(n):
        discrepancy = (bit_string[n] + sum([c[j] * bit_string[n - j -
1] for j in range(1, l + 1)])) % 2
        if discrepancy == 1:
            temp = c[:]
            for j in range(n - m):
                c[n - m + j] = (c[n - m + j] + b[j]) % 2
            if l <= n // 2:
                l = n + 1 - l
                m = n
                b = temp
    return l
```

Passing Criteria:

- **Pass**: The mean linear complexity across all bit strings is greater than a certain threshold. In this case, the threshold is set at 1% of the bit string length, ensuring that the sequences are sufficiently unpredictable.

```
# Check if the sequences pass the linear complexity test
def linear_complexity_test(bit_strings, threshold=0.01):
    complexities = [berlekamp_massey_algorithm(bit_string) for
bit_string in bit_strings]
    mean_complexity = np.mean(complexities)
    passes_complexity_test = mean_complexity > len(bit_strings[0]) *
threshold
    return passes_complexity_test, mean_complexity, complexities
```

Results:

In this case, the QRNG data **passed the linear complexity test**.

```
print(f"Linear Complexity Test Passed: {passes_lc}")
print(f"Mean Linear Complexity: {mean_complexity}")
```

Interpretation:

Passing the linear complexity test means that the QRNG sequences are sufficiently complex and unpredictable. This is crucial for ensuring that the QRNG data can be trusted for applications such as cryptography, where predictability could lead to vulnerabilities.

Conclusion:

By passing both the **DFT test** and the **Linear Complexity test**, the QRNG data generated on IBM's quantum computers demonstrates high entropy and randomness. These properties are critical for practical applications such as encryption protocols, where the security and reliability of random number generation are paramount.