Innovative Quantum Solutions Company

# QCI's Uniform probability distributed Quantum Random Number Generator (uQRNG)

Carrie Spear, David Haycraft, Lac Nguyen, Helen Zhang, Yong Meng Sua

April 2023, v1.0

Quantum Computing Inc
quantumcomputinginc.com
(703) 436-2161

# Contents

# 1 Introduction

Random number generation is the backbone of cryptography and an indispensable resource for a variety of applications, including the analysis of stochastic processes in the realms of physics, biology, finance, and as well as many other fields which require simulation and modeling. As the name implies, a random number is one that is selected by chance, or drawn from a set of numbers obeying a given distribution. A uniform random number must satisfy two requirements: the values are spread equally over a given range or set, and sample draws are independent of one another, meaning that the values of previous draws cannot be used to predict results of future samples. Pseudo-random number generators (PRNGs) produce random sequences based on mathematical algorithms and are therefore deterministic. In contrast, quantum random number generators (QRNGs) leverage the probabilistic nature of quantum mechanics, and can generate unbiased sequences of random numbers, from a non-deterministic process. Diverse physical phenomena have been utilized to implement QRNGs, including measurements of radioactive decay, vacuum noise, intrinsic phase fluctuation of lasers, energy fluctuation of stimulated Raman scattering, and superposition states of single photons. A novel photonic-based quantum random number generation method has been developed by Quantum Computing Inc. (QCI), which harvests randomness from the stochastic process of detecting single photons in superposition of time modes. QCI's, uQRNG product, generates randomness with a uniform probability distribution over a discrete range.

The randomness of a sequence of numbers can be evaluated through the detection of patterns and biases. Various statistical testing methods are available for this purpose, including TestU01, PractRand, Diehard, Dieharder, and NIST tests. Using the uQRNG API that connects to the device via AWS, multiple sets of QRNs were collected. There is no industry standard specifically designed for testing quantum randomness. In order to test the quality of the quantum random numbers two well-known randomness test suites designed for cryptography, NIST [3] and Dieharder [5] were used to conduct rigorous evaluation of the randomness of uQRNG. This paper provides a description of QCI's uQRNG, the conversion process of high-dimensional QRNs to binary, test results for QCI's QRNG, and discusses future research for QRNGs at QCI.

# 2 Methods

## 2.1 QRNG theory of operation

The conceptional illustration of the RNG is depicted in Figure 1. QCI's QRNG is quantum photonic based, the device harvests photons' time-energy degree of freedom. Taking advantage of the discrete nature of photons, a low intensity monochromatic laser with constant intensity is used to generate sparse photon stream. The photon detections will follow a Poissonian statistic.

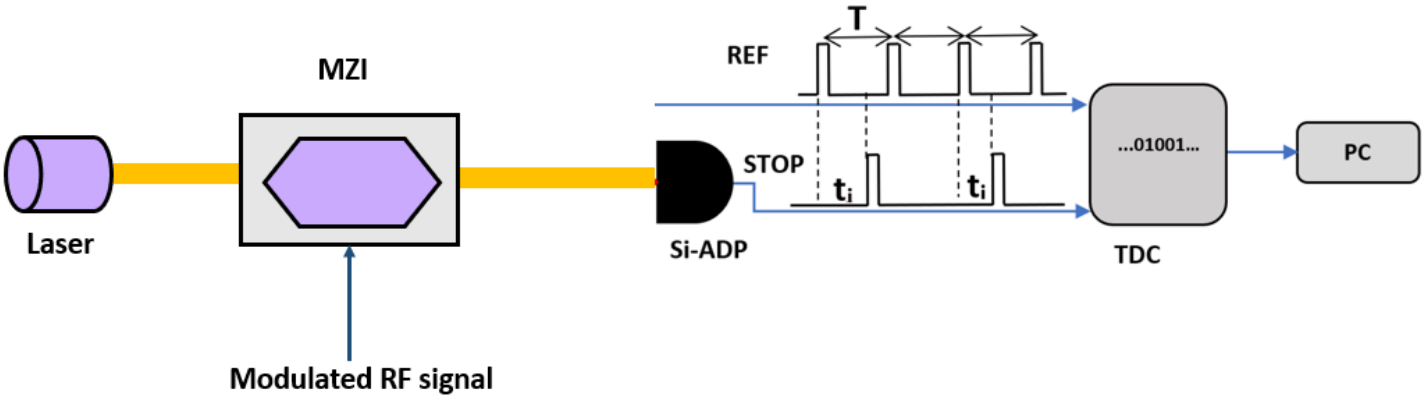


Figure 1: A low intensity visible continuous wave laser is modulated via Mach-Zehnder interferometer (MZI). Silion avalanche photodiode (Si-APD) is used to detect single photon stream. Time-to-Digital converter (TDC) returns the time difference between a signal triggered by the Si-APD and a reference clock. The TDC then sends the harvested data to external device.[4]

Before detection, a single photon is in superposition of the time modes, which means it is impossible to predict which state it will collapse into. The state vector of the photon before measurement is described as follows:

$$|\psi\rangle = c_1 |t_1\rangle + c_2 |t_2\rangle + c_3 |t_3\rangle + \ldots + c_n |t_n\rangle \tag{1}$$

$$\sum_{1}^{n} = (c_n)^2 = 1 \tag{2}$$

The true randomness of this quantum process can be obtained by measuring the stochastic arrival time of single photons compared to a reference clock. By modulating photon temporal waveform with customized shape using Mach-Zehnder interferometer (MZI), QCI's QRNG can produce random series with arbitrary probability distribution, thus, for the first time, provides a direct physical method to generate non-uniform randomness. [4]

## 2.2 uQRNG

A uniform random number (uRN) is a random number where each possible number is equally likely to be produced as any other possible number across an evenly divided range. The uniform random number

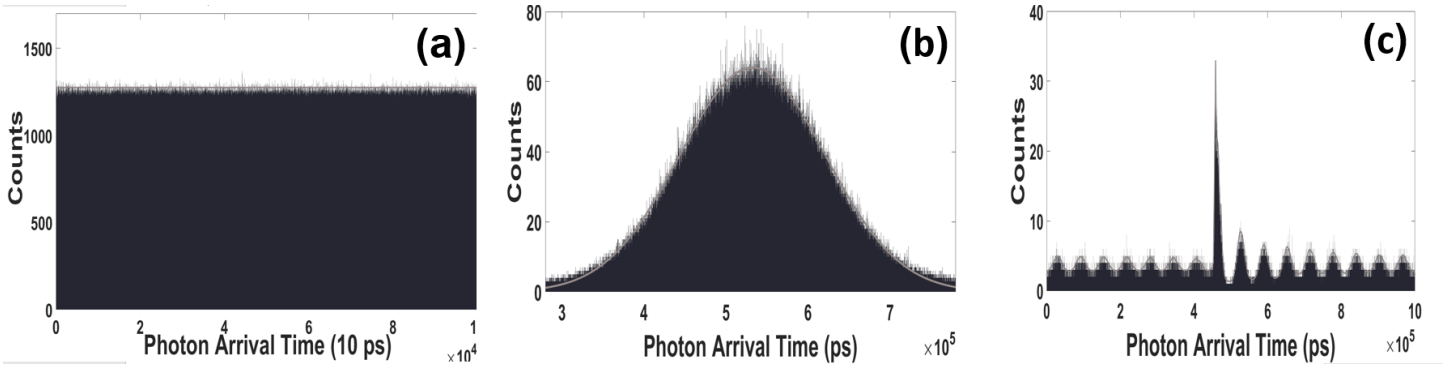


Figure 2: Histograms of typical QRN collecting by customizing probability distribution to be uniform (a), Gaussian (b), and modified Bessel function

distribution generates numbers with equal probability within a given range. For example, uRNs are widely used in simulation-based studies to generate random inputs to test the performance of a system or to simulate a process. For instance, in a Monte Carlo simulation, uRNs are used to simulate the randomness of a process, such as the movement of particles in a system. Besides, uRN are used as entropy source in cryptography for cryptographic protocols. uRNs show their significance in optimization algorithms, where randomly generated candidate solutions should be unbiased and unpredictable.

QCI's uQRNG operates without the modulation signal in order maintain uniformity. Given a $t_i$ time bins during a period, each photon detection is equally likely to fall in any bin, thus creating high-dimensional, uniform RN. Probability amplitude of each mode in equation (2) become

$$c_1 = c_2 = c_3 = \ldots = c_n = \frac{1}{\sqrt{n}} \tag{3}$$

# 3 Randomness study

## 3.1 Test suites

NIST Statistical Testing Suite (STS) version and Dieharder version 3.31.1 were performed to assess the uQRNG. Below are descriptions of each test.

Table 1: NIST test names paired with descriptions [3]

| Test Name | Description |
|---|---|
| **Frequency** | This test determines whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence (0s and 1s should each have a fraction of roughly 1/2) |
| **Block Frequency** | Tests that the proportion of zeroes and ones within M-bit blocks are close to M/2. |
| **Runs** | Tests whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. |
| **Longest Run** | Tests sequence to determine if longest run is consistent with the length that would be expected in a random sequence. |
| **Rank** | This test checks for linear dependence among fixed length substrings of the original sequence. |
| **FFT** | Tests the spectral density of a sequence |
| **Non-overlapping Template** | Tests the frequency of non-overlapping substrings |
| **Overlapping Template** | Tests the frequency of overlapping substrings |
| **Maurer's Universal** | Tests whether or not the sequence can be significantly compressed without loss of information. Too much compression indicates lack of randomness |
| **Cumulative Sums** | Tests for deviations from the mean in the cumulative sum of the sequence |
| **Linear Complexity** | Tests the complexity of a sequence, useful for detecting linear dependence which indicates non-randomness |

*Continued on next page*

Table 1 – *Continued from previous page*

| Test Name | Description |
|---|---|
| **Serial** | Tests whether the number of occurrences of the $2m$ m-bit overlapping patterns is approximately the same as would be expected for a random sequence |
| **Approximate Entropy** | Compares the frequency of overlapping blocks of two consecutive/adjacent lengths ($m$ and $m+1$) against the expected result for a random sequence |
| **Cumulative Sums** | Determines whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences (bits are mapped to $-1$ and $+1$). If sums stray too far from zero is considered non-random |
| **Random Excursions** | Determines if the number of visits to a state within a random walk exceeds what one would expect for a random sequence (bits are mapped to $-1$ and $+1$). |
| **Random Excursions Variant** | Detect deviations from the expected number of occurrences of various states in the random walk. |

Table 2: DieHarder test names paired with descriptions [5]

| Test Name | Description |
|---|---|
| **Diehard Birthdays** | Each test determines the number of matching intervals from 512 "birthdays" (by default) drawn on a 24-bit "year" (by default). |
| **Diehard OPERM5** | This test looks at a sequence of one million 32-bit random integers. Each set of five consecutive integers can be in one of 120 states, for the 5! possible orderings of five numbers. Thus the 5th, 6th, 7th, ... numbers each provide a state. As many thousands of state transitions are observed, cumulative counts are made of the number of occurrences of each state. Then the quadratic form in the weak inverse of the $120 \times 120$ covariance matrix yields a test equivalent to the likelihood ratio test that the 120 cell counts came from the specified (asymptotically) normal distribution with the specified $120 \times 120$ covariance matrix (with rank 99). |
| **Diehard $32 \times 32$ Binary Rank** | A random $32 \times 32$ binary matrix is formed, each row a 32-bit random integer. The rank is determined. That rank can be from 0 to 32, ranks less than 29 are rare, and their counts are pooled with those for rank 29. Ranks are found for 40,000 such random matrices and a chisquare test is performed on counts for ranks 32, 31, 30 and $\leq 29$. |

*Continued on next page*

Table 2 – *Continued from previous page*

| Test Name | Description |
|---|---|
| **Diehard $6 \times 8$ Binary Rank** | From each of six random 32-bit integers from the generator under test, a specified byte is chosen, and the resulting six bytes form a $6 \times 8$ binary matrix whose rank is determined. That rank can be from 0 to 6, but ranks 0, 1, 2, 3 are rare; their counts are pooled with those for rank 4. Ranks are found for 100,000 random matrices, and a chi-square test is performed on counts for ranks 6, 5 and $\leq 4$. |
| **Diehard Bitstream** | The file under test is viewed as a stream of bits. Call them $b_1$, $b_2$, .... Consider an alphabet with two "letters", 0 and 1 and think of the stream of bits as a succession of 20-letter "words", overlapping. Thus the first word is $b_1 b_2 \ldots b_{20}$, the second is $b_2 b_3 \ldots b_{21}$, and so on. The bitstream test counts the number of missing 20-letter (20-bit) words in a string of 2 to the 21 overlapping 20-letter words. |
| **Diehard OPSO** | Diehard Overlapping Pairs Sparse Occupance (OPSO) The OPSO test considers 2-letter words from an alphabet of 1024 letters. Each letter is determined by a specified ten bits from a 32-bit integer in the sequence to be tested. |
| **Diehard OQSO** | Similar to OPSO except that it considers 4-letter words from an alphabet of 32 letters, each letter determined by a designated string of 5 consecutive bits from the test file, elements of which are assumed 32-bit random integers. |
| **Diehard DNA** | The DNA test considers an alphabet of 4 letters:: C,G,A,T, determined by two designated bits in the sequence of random integers being tested. |
| **Diehard Count the 1s (stream)** | Consider the file under test as a stream of bytes (four per 32 bit integer). Each byte can contain from 0 to 8 1's, with probabilities 1, 8, 28, 56, 70, 56, 28, 8, 1 over 256. Now let the stream of bytes provide a string of overlapping 5-letter words, each "letter" taking values A, B, C, D, E. The letters are determined by the number of 1's in a byte:: 0, 1, or 2 yield A, 3 yields B, 4 yields C, 5 yields D and 6, 7 or 8 yield E. |
| **Diehard Count the 1s** | Consider the file under test as a stream of 32-bit integers. From each integer, a specific byte is chosen , say the left most:: bits 1 to 8. Each byte can contain from 0 to 8 1's, with probabilities 1, 8, 28, 56, 70, 56, 28, 8, 1 over 256. Now let the specified bytes from successive integers provide a string of (overlapping) 5-letter words, each "letter" taking values A,B,C,D,E. The letters are determined by the number of 1's, in that byte:: 0, 1, or 2 $\rightarrow$ A, 3 $\rightarrow$ B, 4 $\rightarrow$ C, 5 $\rightarrow$ D, and 6, 7 or 8 $\rightarrow$ E. |
| **Diehard Parking Lot** | This tests the distribution of attempts to randomly park a square car of length 1 on a 100x100 parking lot without crashing. |

*Continued on next page*

Table 2 – *Continued from previous page*

| Test Name | Description |
|---|---|
| **Diehard Minimum Distance (2d Circle)** | This test does this 100 times: choose $n = 8000$ random points in a square of side 10000. Find $d$, the minimum distance between the "$(n^2 - n)/2$" pairs of points. |
| **Diehard 3d Sphere (Minimum Distance)** | Choose 4000 random points in a cube of edge 1000. At each point, center a sphere large enough to reach the next closest point. Then the volume of the smallest such sphere is (very close to) exponentially distributed with mean $120\pi/3$. Thus the radius cubed is exponential with mean 30. (The mean is obtained by extensive simulation). |
| **Diehard Squeeze** | Random integers are floated to get uniforms on $[0, 1)$. Start-ing with $k = 2147483647$, the test finds $j$, the number of iterations necessary to reduce $k$ to 1, using the reduction $k = \text{ceiling}(k \times U)$, with $U$ provided by floating integers from the file being tested |
| **Diehard Sums** | Integers are floated to get a sequence $U(1), U(2), \ldots$ of uniform $[0, 1)$ variables. Then overlapping sums, $S(1) = U(1) + \ldots + U(100)$, $S2 = U(2) + \ldots + U(101), \ldots$ are formed. The $S$'s are virtually normal with a certain covariance matrix. A linear transformation of the $S$'s converts them to a sequence of independent standard normals, which are converted to uniform variables for a KSTEST. The $p$-values from ten KSTESTs are given still another KSTEST. |
| **Diehard Runs** | This test counts runs up, and runs down, in a sequence of uniform $[0, 1)$ variables, obtained by floating the 32-bit integers in the specified file. This example shows how runs are counted: .123, .357, .789, .425, .224, .416, .95 contains an up-run of length 3, a down-run of length 2 and an up-run of (at least) 2, depending on the next values. |
| **Diehard Craps** | This test plays 200,000 games of craps, finds the number of wins and the number of throws necessary to end each game. The number of wins should be (very close to) a normal with mean $200000p$ and variance $200000p(1 - p)$, with $p = 244/495$. |
| **Marsaglia and Tsang GCD** | 10 to the 7 tsamples (default) of uint rands $u$, $v$ are generated and two statistics are generated: their greatest common divisor (GCD) ($w$) and the number of steps of Euclid's Method required to find it ($k$). |
| **STS Monobit** | This test counts the 1 bits in a long string of random uints. Compares to expected number, generates a $p$-value directly from `erfc()`. |
| **STS Runs** | This test counts the total number of 0 runs + total number of runs across a sample of bits. |

*Continued on next page*

Table 2 – *Continued from previous page*

| Test Name | Description |
|---|---|
| **STS Serial** | This test accumulates the frequencies of overlapping $n$-tuples of bits drawn from a source of random integers. |
| **RGB Bit Distribution** | This test accumulates the frequencies of all $n$ tuples of bits in a list of random integers and compares the distribution thus generated with the theoretical (binomial) histogram, forming chisq and the associated $p$-value. |
| **RGB Generalized Minimum Distance** | This is the generalized minimum distance test, based on the paper of M. Fischler in the doc directory and private communications. This test utilizes correction terms that are essential in order for the test not to fail for large numbers of trials. It replaces both "`diehard_2dsphere.c`" and "`diehard_3dsphere.c`", and generalizes the test itself so that it can be run for any $d = 2, 3, 4, 5$. |
| **RGB Permutations** | This is a non-overlapping test that simply counts order permutations of random numbers, pulled out $n$ at a time. There are $n!$ permutations and all are equally likely. |
| **RGB Lagged Sum** | Test for lagged correlations, the possibility that the RNG has a bitlevel correlation after some fixed number of intervening bits. |
| **RGB Kolmogorov-Smirnov Test** | This test generates a vector of tsamples uniform deviates from the selected rng, then applies an Anderson-Darling or Kuiper KS test to it to directly test for uniformity. |

## 3.2 Data preparation

The data for the testing suite was obtained from the QRNG Application Programming Interface (API), which pulls from a cache of numbers that were created by QCI's photonic uQRNG. Data samples consist of uniform bits, meaning that each bit has an equal probability of being 0 or 1. 845 million 32-bit samples were obtained by making requests using Python through uQRNG API version 0.0.1[1]. For more information on QCI's uQRNG API see Appendix A.

NIST STS version 2.1.2 and Dieharder version 3.31.1 were performed to assess the uQRNG. The same set of 845 million 32-bit samples was used for both testing suites. The NIST STS is a set of 15 statistical tests that are meant to detect randomness in binary sequences. In this test suite, each of the individual tests was executed 10 times. 10-bit streams were used for all tests to determine the $p$-values. The assess value, which is defined as "the number of bits to test in one test run," was set to 387840. Each of NISTs tests is considered to have passed if conforms to $0.01 < p < 0.99$. The Dieharder suite consists of 19 tests and demands a larger dataset than NIST to carry out a comprehensive evaluation. The Kolmogorov Smirnov (KS) test is employed by Dieharder to assess the $p$-value distribution, which is highly sensitive to lack of uniformity. Uniformly high $p$-values or low $p$-values both indicate bias in the testing results for a given RNG. Each test is run 100 times to ensure that the resolution is high enough to assess lack of uniformity in the $p$-values. The default threshold value of $0.000001 < p \leq 0.005$ defines a WEAK

result and $p \leq 0.000001$ determines a FAILED result. Although the Dieharder documentation does not provide guidelines for sample size for analyzing random numbers, according to Hurley-Smith [2], their work indicates, a minimum of 228 GB of 32-bit data is necessary to avoid data rewinding during testing. Due to the unavailability of sufficient data during the data collection period, Dieharder was run with rewinds of the random numbers to obtain the results.

## 3.3 Results

Using the test settings as described above each NIST test all 15 tests passed with $p$-values greater than 0.01. The tests included in the suite are designed to detect a wide range of deviations from randomness, such as biases, patterns, and correlations, and the generator demonstrated robustness against all of these potential weaknesses.

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <32_bit_binary_800M.txt>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  2   2   1   0   1   1   2   1   0   0  0.739918     10/10      Frequency
  1   0   0   2   1   1   0   1   4   0  0.122325      9/10      BlockFrequency
  3   2   2   0   1   1   0   0   0   1  0.350485     10/10      CumulativeSums
  4   0   1   1   0   0   1   2   1   0  0.122325     10/10      CumulativeSums
  2   1   2   0   0   2   0   2   0   1  0.534146     10/10      Runs
  2   0   1   1   1   1   2   1   0   1  0.911413     10/10      LongestRun
  0   3   0   0   1   1   3   1   1   0  0.213309     10/10      Rank
  1   1   1   1   1   1   1   1   2   0  0.991468      9/10      FFT
  1   2   0   0   1   1   1   2   1   1  0.911413     10/10      NonOverlappingTemplate
  1   0   1   2   1   1   0   0   2   2  0.739918     10/10      NonOverlappingTemplate
  6   0   0   0   1   0   0   0   0   3  0.000040 *    9/10      NonOverlappingTemplate
  1   1   2   0   1   2   0   2   0   1  0.739918     10/10      NonOverlappingTemplate
  0   2   0   1   1   0   2   1   1   2  0.739918     10/10      NonOverlappingTemplate
  4   1   1   0   1   0   2   0   1   0  0.122325     10/10      NonOverlappingTemplate
  0   1   2   1   0   1   2   1   2   0  0.739918     10/10      NonOverlappingTemplate
  0   0   1   1   0   3   1   3   0   1  0.213309     10/10      NonOverlappingTemplate
  1   1   0   1   0   2   1   1   2   1  0.911413     10/10      NonOverlappingTemplate
  1   2   1   1   0   0   0   0   4   1  0.122325     10/10      NonOverlappingTemplate
  1   1   0   1   3   2   0   1   0   1  0.534146     10/10      NonOverlappingTemplate
  1   2   2   0   2   1   0   0   1   1  0.739918     10/10      NonOverlappingTemplate
  2   0   0   0   1   3   0   1   1   2  0.350485     10/10      NonOverlappingTemplate
  1   1   2   0   0   0   0   3   1   2  0.350485     10/10      NonOverlappingTemplate
  4   1   1   1   0   2   0   0   1   0  0.122325      9/10      NonOverlappingTemplate
  2   3   2   2   0   0   0   0   1   0  0.213309     10/10      NonOverlappingTemplate
  1   0   1   1   1   2   1   1   1   1  0.991468     10/10      NonOverlappingTemplate
  0   0   1   0   1   2   2   2   2   0  0.534146     10/10      NonOverlappingTemplate
  0   0   2   2   3   1   0   1   0   1  0.350485     10/10      NonOverlappingTemplate
  1   1   2   1   1   1   1   1   0   1  0.991468     10/10      NonOverlappingTemplate
  0   1   3   0   1   1   1   0   1   2  0.534146     10/10      NonOverlappingTemplate
  0   0   2   1   2   0   2   2   1   0  0.534146     10/10      NonOverlappingTemplate
```

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 3 | 1 | 1 | 3 | 0 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 0 | 0 | 2 | 3 | 1 | 1 | 0 | 2 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 2 | 1 | 1 | 1 | 0 | 3 | 0 | 0 | 1 | 1 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 1 | 1 | 0 | 3 | 1 | 0 | 1 | 2 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 2 | 0 | 2 | 0 | 1 | 1 | 2 | 0 | 0.534146 | 9/10 | NonOverlappingTemplate |
| 4 | 2 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 1 | 0.066882 | 9/10 | NonOverlappingTemplate |
| 0 | 0 | 2 | 0 | 1 | 2 | 2 | 0 | 1 | 2 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 1 | 0 | 1 | 0 | 1 | 3 | 0 | 2 | 0.350485 | 9/10 | NonOverlappingTemplate |
| 0 | 1 | 2 | 1 | 2 | 0 | 1 | 1 | 2 | 0 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 1 | 0 | 0 | 2 | 1 | 0 | 2 | 2 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 2 | 1 | 3 | 0 | 0 | 1 | 0 | 1 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 1 | 1 | 1 | 1 | 2 | 0 | 2 | 0 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 3 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 2 | 3 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 3 | 0 | 0 | 1 | 2 | 2 | 0 | 1 | 0 | 1 | 0.350485 | 9/10 | NonOverlappingTemplate |
| 2 | 1 | 0 | 1 | 0 | 0 | 2 | 2 | 1 | 1 | 0.739918 | 9/10 | NonOverlappingTemplate |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 1 | 0.991468 | 9/10 | NonOverlappingTemplate |
| 0 | 2 | 2 | 0 | 0 | 3 | 1 | 0 | 0 | 2 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 0 | 2 | 0 | 3 | 0 | 1 | 1 | 1 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 2 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 3 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 1 | 2 | 0 | 3 | 2 | 1 | 0 | 1 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 2 | 1 | 1 | 0 | 0 | 2 | 3 | 0 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 0 | 3 | 1 | 0 | 0 | 2 | 1 | 2 | 1 | 0 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 1 | 2 | 2 | 0 | 0 | 2 | 0 | 0 | 3 | 0 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 2 | 0 | 0 | 1 | 0 | 1 | 2 | 4 | 0.066882 | 10/10 | NonOverlappingTemplate |
| 1 | 0 | 1 | 4 | 2 | 0 | 2 | 0 | 0 | 0 | 0.066882 | 10/10 | NonOverlappingTemplate |
| 3 | 1 | 1 | 2 | 1 | 0 | 1 | 0 | 0 | 1 | 0.534146 | 9/10 | NonOverlappingTemplate |
| 3 | 0 | 1 | 0 | 2 | 0 | 2 | 1 | 0 | 1 | 0.350485 | 9/10 | NonOverlappingTemplate |
| 0 | 2 | 1 | 0 | 1 | 0 | 4 | 0 | 2 | 0 | 0.066882 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 1 | 2 | 1 | 1 | 0 | 1 | 2 | 2 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 2 | 0 | 2 | 1 | 3 | 1 | 0 | 0 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 4 | 1 | 0 | 1 | 0 | 0 | 0 | 2 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 2 | 1 | 2 | 1 | 1 | 0 | 2 | 1 | 0 | 0 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 0 | 2 | 1 | 1 | 0 | 1 | 0 | 3 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 0 | 1 | 4 | 2 | 0 | 1 | 0 | 1 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 3 | 0 | 1 | 0 | 3 | 0 | 1 | 2 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 1 | 1 | 0 | 1 | 1 | 2 | 2 | 0 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 3 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 3 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 0 | 2 | 1 | 1 | 3 | 3 | 0 | 0 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 3 | 2 | 0.534146 | 9/10 | NonOverlappingTemplate |
| 0 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 0.991468 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 0 | 3 | 1 | 0 | 1 | 1 | 1 | 1 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 1 | 0 | 2 | 3 | 1 | 1 | 0 | 1 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 2 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 0.739918 | 9/10 | NonOverlappingTemplate |
| 3 | 0 | 1 | 1 | 2 | 1 | 1 | 0 | 1 | 0 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 0 | 1 | 2 | 1 | 2 | 4 | 0 | 0 | 0.066882 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 2 | 2 | 2 | 0 | 0 | 1 | 1 | 1 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 0 | 2 | 1 | 1 | 1 | 1 | 2 | 0 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 2 | 1 | 1 | 0 | 1 | 0 | 2 | 2 | 1 | 0 | 0.739918 | 9/10 | NonOverlappingTemplate |
| 2 | 0 | 1 | 1 | 2 | 1 | 0 | 1 | 1 | 1 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 2 | 1 | 0 | 1 | 1 | 2 | 1 | 0 | 0 | 2 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 1 | 2 | 0 | 0 | 1 | 1 | 1 | 2 | 1 | 1 | 0.911413 | 10/10 | NonOverlappingTemplate |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 2 | 0 | 2 | 2 | 0 | 1 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 1 | 0 | 0 | 2 | 3 | 1 | 0 | 2 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 0 | 1 | 3 | 0 | 2 | 1 | 1 | 0 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 0 | 0 | 2 | 2 | 0 | 2 | 0 | 3 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 1 | 0 | 3 | 1 | 0 | 2 | 2 | 1 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 2 | 1 | 0 | 1 | 2 | 1 | 1 | 1 | 0 | 1 | 0.911413 | 8/10 | NonOverlappingTemplate |
| 1 | 3 | 1 | 1 | 0 | 0 | 1 | 0 | 2 | 1 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 1 | 3 | 0 | 2 | 0 | 1 | 0 | 2 | 1 | 0 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 0 | 1 | 2 | 0 | 0 | 3 | 2 | 0 | 0.350485 | 9/10 | NonOverlappingTemplate |
| 0 | 3 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 0 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 1 | 0 | 2 | 3 | 0 | 0 | 1 | 1 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 3 | 2 | 1 | 1 | 0 | 0 | 2 | 1 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 1 | 0 | 1 | 1 | 2 | 0 | 2 | 0 | 0 | 3 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 4 | 2 | 0 | 1 | 0 | 2 | 1 | 0 | 0.066882 | 10/10 | NonOverlappingTemplate |
| 3 | 0 | 1 | 2 | 0 | 1 | 0 | 1 | 1 | 1 | 0.534146 | 9/10 | NonOverlappingTemplate |
| 1 | 0 | 1 | 1 | 3 | 1 | 1 | 1 | 0 | 1 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 1 | 0 | 0 | 1 | 0 | 2 | 2 | 1 | 2 | 1 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 1 | 1 | 1 | 2 | 0 | 2 | 0 | 1 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 3 | 2 | 2 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 1 | 0 | 2 | 2 | 0 | 1 | 0 | 2 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 1 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 1 | 3 | 1 | 1 | 0 | 0 | 2 | 1 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 0 | 1 | 0 | 2 | 1 | 2 | 2 | 0 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 1 | 3 | 2 | 0 | 0 | 2 | 1 | 0 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 0 | 1 | 0 | 0 | 3 | 3 | 1 | 0 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 3 | 0 | 1 | 2 | 1 | 1 | 0 | 1 | 0 | 1 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 1 | 1 | 1 | 3 | 1 | 0 | 0 | 1 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 0 | 0 | 3 | 1 | 3 | 1 | 1 | 0 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 2 | 1 | 1 | 1 | 0 | 1 | 0 | 2 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 0 | 4 | 0 | 1 | 1 | 1 | 1 | 2 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 0 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 0.911413 | 9/10 | NonOverlappingTemplate |
| 1 | 1 | 0 | 0 | 2 | 1 | 2 | 1 | 0 | 2 | 0.739918 | 9/10 | NonOverlappingTemplate |
| 4 | 1 | 1 | 2 | 0 | 0 | 1 | 1 | 0 | 0 | 0.122325 | 9/10 | NonOverlappingTemplate |
| 2 | 1 | 1 | 1 | 2 | 0 | 0 | 1 | 2 | 0 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 2 | 0 | 1 | 2 | 0 | 2 | 1 | 0 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 1 | 2 | 0 | 2 | 1 | 3 | 1 | 0 | 0 | 0 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 2 | 2 | 1 | 1 | 1 | 1 | 2 | 0 | 0 | 0 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 0 | 1 | 3 | 1 | 1 | 1 | 1 | 0 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 3 | 0 | 0 | 0 | 1 | 0 | 2 | 2 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 1 | 0 | 3 | 2 | 0 | 0 | 2 | 0 | 2 | 0 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 0 | 4 | 1 | 1 | 2 | 0 | 0 | 0 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 1 | 2 | 1 | 2 | 1 | 0 | 0 | 1 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 0 | 3 | 1 | 2 | 1 | 0 | 0 | 1 | 2 | 0 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 4 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 1 | 0.991468 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 1 | 1 | 1 | 3 | 0 | 2 | 0 | 0 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 2 | 1 | 0 | 2 | 3 | 0 | 0 | 1 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 2 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 4 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 1 | 2 | 0 | 0 | 3 | 1 | 0 | 1 | 1 | 1 | 0.534146 | 9/10 | NonOverlappingTemplate |
| 1 | 1 | 0 | 0 | 2 | 2 | 1 | 2 | 1 | 0 | 0.739918 | 9/10 | NonOverlappingTemplate |
| 2 | 1 | 1 | 1 | 1 | 1 | 2 | 0 | 1 | 0 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 1 | 3 | 3 | 0 | 1 | 0 | 1 | 0 | 0.213309 | 10/10 | NonOverlappingTemplate |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 1 | 2 | 1 | 1 | 0 | 1 | 1 | 1 | 2 | 0 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 1 | 1 | 2 | 1 | 1 | 3 | 1 | 0 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 2 | 2 | 1 | 0.739918 | 9/10 | NonOverlappingTemplate |
| 1 | 1 | 2 | 0 | 1 | 1 | 0 | 2 | 1 | 1 | 0.911413 | 10/10 | NonOverlappingTemplate |
| 1 | 2 | 1 | 0 | 0 | 1 | 1 | 2 | 2 | 0 | 0.739918 | 9/10 | NonOverlappingTemplate |
| 3 | 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0.035174 | 9/10 | NonOverlappingTemplate |
| 1 | 2 | 0 | 2 | 0 | 0 | 2 | 1 | 2 | 0 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 3 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 2 | 2 | 1 | 0 | 0 | 0 | 1 | 2 | 1 | 1 | 0.739918 | 9/10 | NonOverlappingTemplate |
| 0 | 1 | 3 | 0 | 2 | 0 | 0 | 1 | 1 | 2 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 2 | 2 | 1 | 1 | 1 | 2 | 0 | 0 | 0 | 1 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 1 | 0 | 3 | 1 | 1 | 0 | 2 | 1 | 1 | 0 | 0.534146 | 10/10 | NonOverlappingTemplate |
| 1 | 2 | 2 | 2 | 1 | 1 | 0 | 0 | 0 | 1 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 0 | 2 | 3 | 0 | 1 | 1 | 1 | 0 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 0 | 0 | 1 | 3 | 1 | 1 | 0 | 1 | 0 | 3 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 1 | 2 | 1 | 1 | 0 | 0 | 0 | 2 | 0 | 3 | 0.350485 | 10/10 | NonOverlappingTemplate |
| 0 | 2 | 2 | 0 | 3 | 1 | 0 | 0 | 2 | 0 | 0.213309 | 10/10 | NonOverlappingTemplate |
| 1 | 3 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 3 | 0.122325 | 10/10 | NonOverlappingTemplate |
| 2 | 1 | 0 | 1 | 1 | 2 | 1 | 0 | 0 | 2 | 0.739918 | 10/10 | NonOverlappingTemplate |
| 1 | 1 | 2 | 2 | 0 | 0 | 0 | 2 | 1 | 1 | 0.739918 | 9/10 | OverlappingTemplate |
| 1 | 1 | 2 | 1 | 0 | 0 | 1 | 0 | 3 | 1 | 0.534146 | 10/10 | Universal |
| 1 | 2 | 1 | 1 | 0 | 0 | 0 | 2 | 1 | 2 | 0.739918 | 10/10 | ApproximateEntropy |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursions |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursions |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | ---- | 2/2 | RandomExcursions |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ---- | 2/2 | RandomExcursions |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ---- | 2/2 | RandomExcursions |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursions |
| 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursions |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursions |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | ---- | 2/2 | RandomExcursionsVariant |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | ---- | 2/2 | RandomExcursionsVariant |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | ---- | 2/2 | RandomExcursionsVariant |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 2 | 2 | 0.534146 | 10/10 | Serial |
| 0 | 1 | 1 | 2 | 1 | 0 | 1 | 1 | 3 | 0 | 0.534146 | 10/10 | Serial |
| 0 | 3 | 3 | 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0.122325 | 10/10 | LinearComplexity |

```
#=============================================================================#
#            dieharder version 3.31.1 Copyright 2003 Robert G. Brown          #
#=============================================================================#
   rng_name    |           filename             |rands/second|
     file_input|         32_bit_integer_800M.txt|  8.78e+06  |
#=============================================================================#
        test_name   |ntup| tsamples |psamples|  p-value |Assessment
#=============================================================================#
   diehard_birthdays|   0|       100|     100|0.44663328|  PASSED
      diehard_operm5|   0|   1000000|     100|0.26256205|  PASSED
  diehard_rank_32x32|   0|     40000|     100|0.64559216|  PASSED
    diehard_rank_6x8|   0|    100000|     100|0.17869747|  PASSED
   diehard_bitstream|   0|   2097152|     100|0.64172738|  PASSED
        diehard_opso|   0|   2097152|     100|0.41406637|  PASSED
        diehard_oqso|   0|   2097152|     100|0.25648563|  PASSED
         diehard_dna|   0|   2097152|     100|0.67515442|  PASSED
diehard_count_1s_str|   0|    256000|     100|0.47840758|  PASSED
# The file file_input was rewound 1 times
diehard_count_1s_byt|   0|    256000|     100|0.74422823|  PASSED
# The file file_input was rewound 1 times
 diehard_parking_lot|   0|     12000|     100|0.77335420|  PASSED
# The file file_input was rewound 1 times
    diehard_2dsphere|   2|      8000|     100|0.40146872|  PASSED
# The file file_input was rewound 1 times
    diehard_3dsphere|   3|      4000|     100|0.16962659|  PASSED
# The file file_input was rewound 1 times
     diehard_squeeze|   0|    100000|     100|0.84715316|  PASSED
# The file file_input was rewound 1 times
        diehard_sums|   0|       100|     100|0.41052706|  PASSED
# The file file_input was rewound 1 times
        diehard_runs|   0|    100000|     100|0.50385166|  PASSED
        diehard_runs|   0|    100000|     100|0.09392926|  PASSED
# The file file_input was rewound 1 times
       diehard_craps|   0|    200000|     100|0.04273389|  PASSED
       diehard_craps|   0|    200000|     100|0.97702765|  PASSED
# The file file_input was rewound 3 times
 marsaglia_tsang_gcd|   0|  10000000|     100|0.68173086|  PASSED
 marsaglia_tsang_gcd|   0|  10000000|     100|0.86101261|  PASSED
# The file file_input was rewound 3 times
         sts_monobit|   1|    100000|     100|0.66452769|  PASSED
# The file file_input was rewound 3 times
            sts_runs|   2|    100000|     100|0.70050564|  PASSED
# The file file_input was rewound 3 times
          sts_serial|   1|    100000|     100|0.36749601|  PASSED
          sts_serial|   2|    100000|     100|0.20622078|  PASSED
          sts_serial|   3|    100000|     100|0.99746823|   WEAK
          sts_serial|   3|    100000|     100|0.52995769|  PASSED
          sts_serial|   4|    100000|     100|0.10023768|  PASSED
          sts_serial|   4|    100000|     100|0.07602888|  PASSED
          sts_serial|   5|    100000|     100|0.13184663|  PASSED
          sts_serial|   5|    100000|     100|0.56652842|  PASSED
          sts_serial|   6|    100000|     100|0.10385636|  PASSED
```

```
          sts_serial|   6|    100000|     100|0.79059996|  PASSED
          sts_serial|   7|    100000|     100|0.34624310|  PASSED
          sts_serial|   7|    100000|     100|0.71858090|  PASSED
          sts_serial|   8|    100000|     100|0.89735037|  PASSED
          sts_serial|   8|    100000|     100|0.01789872|  PASSED
          sts_serial|   9|    100000|     100|0.96758826|  PASSED
          sts_serial|   9|    100000|     100|0.88403874|  PASSED
          sts_serial|  10|    100000|     100|0.98654226|  PASSED
          sts_serial|  10|    100000|     100|0.71163422|  PASSED
          sts_serial|  11|    100000|     100|0.62224236|  PASSED
          sts_serial|  11|    100000|     100|0.43084437|  PASSED
          sts_serial|  12|    100000|     100|0.88525455|  PASSED
          sts_serial|  12|    100000|     100|0.22004571|  PASSED
          sts_serial|  13|    100000|     100|0.98207576|  PASSED
          sts_serial|  13|    100000|     100|0.99914386|   WEAK
          sts_serial|  14|    100000|     100|0.66350655|  PASSED
          sts_serial|  14|    100000|     100|0.92970201|  PASSED
          sts_serial|  15|    100000|     100|0.33400213|  PASSED
          sts_serial|  15|    100000|     100|0.10592566|  PASSED
          sts_serial|  16|    100000|     100|0.86196857|  PASSED
          sts_serial|  16|    100000|     100|0.99936798|   WEAK
# The file file_input was rewound 3 times
         rgb_bitdist|   1|    100000|     100|0.51511542|  PASSED
# The file file_input was rewound 3 times
         rgb_bitdist|   2|    100000|     100|0.26396122|  PASSED
# The file file_input was rewound 4 times
         rgb_bitdist|   3|    100000|     100|0.82852015|  PASSED
# The file file_input was rewound 4 times
         rgb_bitdist|   4|    100000|     100|0.74598084|  PASSED
# The file file_input was rewound 4 times
         rgb_bitdist|   5|    100000|     100|0.99263637|  PASSED
# The file file_input was rewound 4 times
         rgb_bitdist|   6|    100000|     100|0.44968481|  PASSED
# The file file_input was rewound 4 times
         rgb_bitdist|   7|    100000|     100|0.98078560|  PASSED
# The file file_input was rewound 4 times
         rgb_bitdist|   8|    100000|     100|0.01132557|  PASSED
# The file file_input was rewound 4 times
         rgb_bitdist|   9|    100000|     100|0.57425840|  PASSED
# The file file_input was rewound 5 times
         rgb_bitdist|  10|    100000|     100|0.90516242|  PASSED
# The file file_input was rewound 5 times
         rgb_bitdist|  11|    100000|     100|0.55928194|  PASSED
# The file file_input was rewound 5 times
         rgb_bitdist|  12|    100000|     100|0.69490668|  PASSED
# The file file_input was rewound 5 times
rgb_minimum_distance|   2|     10000|    1000|0.61814302|  PASSED
# The file file_input was rewound 5 times
rgb_minimum_distance|   3|     10000|    1000|0.84779611|  PASSED
# The file file_input was rewound 5 times
rgb_minimum_distance|   4|     10000|    1000|0.31647075|  PASSED
# The file file_input was rewound 5 times
rgb_minimum_distance|   5|     10000|    1000|0.06032514|  PASSED
```

```
# The file file_input was rewound 5 times
    rgb_permutations|   2|    100000|     100|0.80529402|  PASSED
# The file file_input was rewound 5 times
    rgb_permutations|   3|    100000|     100|0.03425501|  PASSED
# The file file_input was rewound 6 times
    rgb_permutations|   4|    100000|     100|0.49196648|  PASSED
# The file file_input was rewound 6 times
    rgb_permutations|   5|    100000|     100|0.62286315|  PASSED
# The file file_input was rewound 6 times
      rgb_lagged_sum|   0|   1000000|     100|0.08032407|  PASSED
# The file file_input was rewound 6 times
      rgb_lagged_sum|   1|   1000000|     100|0.77861190|  PASSED
# The file file_input was rewound 6 times
      rgb_lagged_sum|   2|   1000000|     100|0.57394603|  PASSED
# The file file_input was rewound 7 times
      rgb_lagged_sum|   3|   1000000|     100|0.99697086|   WEAK
# The file file_input was rewound 7 times
      rgb_lagged_sum|   4|   1000000|     100|0.90135260|  PASSED
# The file file_input was rewound 8 times
      rgb_lagged_sum|   5|   1000000|     100|0.81760955|  PASSED
# The file file_input was rewound 9 times
      rgb_lagged_sum|   6|   1000000|     100|0.14199120|  PASSED
# The file file_input was rewound 10 times
      rgb_lagged_sum|   7|   1000000|     100|0.17575320|  PASSED
# The file file_input was rewound 11 times
      rgb_lagged_sum|   8|   1000000|     100|0.06310426|  PASSED
# The file file_input was rewound 12 times
      rgb_lagged_sum|   9|   1000000|     100|0.16816091|  PASSED
# The file file_input was rewound 13 times
      rgb_lagged_sum|  10|   1000000|     100|0.38480770|  PASSED
# The file file_input was rewound 15 times
      rgb_lagged_sum|  11|   1000000|     100|0.55162562|  PASSED
# The file file_input was rewound 16 times
      rgb_lagged_sum|  12|   1000000|     100|0.39563913|  PASSED
# The file file_input was rewound 18 times
      rgb_lagged_sum|  13|   1000000|     100|0.99318675|  PASSED
# The file file_input was rewound 20 times
      rgb_lagged_sum|  14|   1000000|     100|0.07681033|  PASSED
# The file file_input was rewound 22 times
      rgb_lagged_sum|  15|   1000000|     100|0.60752620|  PASSED
# The file file_input was rewound 24 times
      rgb_lagged_sum|  16|   1000000|     100|0.27987095|  PASSED
# The file file_input was rewound 26 times
      rgb_lagged_sum|  17|   1000000|     100|0.27201659|  PASSED
# The file file_input was rewound 28 times
      rgb_lagged_sum|  18|   1000000|     100|0.83955936|  PASSED
# The file file_input was rewound 30 times
      rgb_lagged_sum|  19|   1000000|     100|0.71743152|  PASSED
# The file file_input was rewound 33 times
      rgb_lagged_sum|  20|   1000000|     100|0.83752136|  PASSED
# The file file_input was rewound 35 times
      rgb_lagged_sum|  21|   1000000|     100|0.86037210|  PASSED
# The file file_input was rewound 38 times
```

```
      rgb_lagged_sum|  22|   1000000|     100|0.85230898|  PASSED
# The file file_input was rewound 41 times
      rgb_lagged_sum|  23|   1000000|     100|0.09290434|  PASSED
# The file file_input was rewound 44 times
      rgb_lagged_sum|  24|   1000000|     100|0.88811465|  PASSED
# The file file_input was rewound 47 times
      rgb_lagged_sum|  25|   1000000|     100|0.86622083|  PASSED
# The file file_input was rewound 50 times
      rgb_lagged_sum|  26|   1000000|     100|0.04172537|  PASSED
# The file file_input was rewound 54 times
      rgb_lagged_sum|  27|   1000000|     100|0.55598233|  PASSED
# The file file_input was rewound 57 times
      rgb_lagged_sum|  28|   1000000|     100|0.97069922|  PASSED
# The file file_input was rewound 61 times
      rgb_lagged_sum|  29|   1000000|     100|0.85878129|  PASSED
# The file file_input was rewound 64 times
      rgb_lagged_sum|  30|   1000000|     100|0.75900465|  PASSED
# The file file_input was rewound 68 times
      rgb_lagged_sum|  31|   1000000|     100|0.13147276|  PASSED
# The file file_input was rewound 72 times
      rgb_lagged_sum|  32|   1000000|     100|0.40309133|  PASSED
# The file file_input was rewound 72 times
     rgb_kstest_test|   0|     10000|    1000|0.06357618|  PASSED
# The file file_input was rewound 72 times
     dab_bytedistrib|   0|  51200000|       1|0.32398090|  PASSED
# The file file_input was rewound 72 times
             dab_dct| 256|     50000|       1|0.74044676|  PASSED
Preparing to run test 207.  ntuple = 0
# The file file_input was rewound 72 times
        dab_filltree|  32|  15000000|       1|0.28111781|  PASSED
        dab_filltree|  32|  15000000|       1|0.27236169|  PASSED
Preparing to run test 208.  ntuple = 0
# The file file_input was rewound 72 times
       dab_filltree2|   0|   5000000|       1|0.08949423|  PASSED
       dab_filltree2|   1|   5000000|       1|0.98239434|  PASSED
Preparing to run test 209.  ntuple = 0
# The file file_input was rewound 72 times
        dab_monobit2|  12|  65000000|       1|0.96928433|  PASSED
```

Dieharder similarly passed all 19 tests without any significant failures. However, some runs did have WEAK individual $p$-values. STS serial had 3 WEAK $p$-values over the course of its 29 runs. RGB lagged sum had one WEAK $p$-value across the 33 runs. Conducting Dieharder with more samples and higher total number of tests may provide more insight into these result. This may be considered to be part of the random fluctuations in the underlying test statistic and stronger weight should be given to the passing results on this harder battery of tests indicating the high quality of the random numbers from the uQRNG.

# 4 Discussion

In conducting randomness tests, it is not uncommon to observe some failures due to random chance. To mitigate this phenomenon, adjustments such as analyzing multiple tests and observing the distribution of $p$-values were employed [6]. The sample size used in this study was adequate to pass the randomness testing suites, as all tests yielded $p$-values above the significance threshold. However, it is worth noting that for the Dieharder test suite, a larger sample size may have been preferable to reduce the need for rewinding during testing. This would have provided a more accurate assessment of the generator's randomness properties. Nonetheless, the sample size used in this study was still deemed sufficient to provide a reliable evaluation of the generator's performance. While true randomness cannot be replicated exactly, similar test settings should yield comparable results.

These methods employ a battery of statistical tests to analyze the given sequence. Nevertheless, these tests are not infallible, and it is possible to construct a set of numbers that could deliberately pass these tests and give the impression of being random.

Quantum Computing Inc's uQRNG device generates high-quality numbers which can pass the most stringent tests available without any randomness extraction to alleviate bias or randomness amplification to increase entropy. Given the true randomness inherent in the underlying quantum mechanical processes, the QRNG used in this study is well-suited for entropy source of cryptographic protocols. Furthermore, the lack of bias exhibited by the generator may also prove advantageous for generating random numbers for other applications such as experiments and simulations that require unbiased and truly random results. Future research will explore the potential benefits of using this generator for real-world experimentation in modeling, simulation, and cryptographic applications.

DieHarder and NIST are two of many possible randomness tests. Testing using other suites may be desirable to further validate the generator's performance. QCI is committed to conducting all necessary statistical tests and standards to support existing and future customer applications.The future development of QRNG devices by QCI includes random numbers certified by non-local correlations of quantum states by the means of violation of Bell's inequality beyond standard statistical randomness test suite. The violation of Bell's inequality guarantees that the correlated random numbers generated from quantum entangled systems possess intrinsic randomness.

# References


[1] Quantum Computing Inc. (QCI). *QCI QRNG API Guide*. https://quantumcomputinginc.com/qrng/api-guide/. Accessed on April 18, 2023.

[2] Darren Hurley-Smith and Julio Hernandez-Castro. "Challenges in Certifying Small-Scale (IoT) Hardware Random Number Generators". In: *Security of Ubiquitous Computing Systems: Selected Topics*. Ed. by Gildas Avoine and Julio Hernandez-Castro. Cham: Springer International Publishing, 2021, pp. 165–181. ISBN: 978-3-030-10591-4. DOI: 10.1007/978-3-030-10591-4_10. URL: https://doi.org/10.1007/978-3-030-10591-4_10.

[3] National Institute of Standards and Technology. *Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software. Accessed: March 31, 2023. 2010.

[4] Lac Nguyen et al. "Programmable quantum random number generator without postprocessing". In: *Opt. Lett.* 43.4 (Feb. 2018), pp. 631–634. DOI: 10.1364/OL.43.000631. URL: https://opg.optica.org/ol/abstract.cfm?URI=ol-43-4-631.

[5] Robert G. Brown. *Dieharder: A Random Number Test Suite*. https://webhome.phy.duke.edu/~rgb/General/dieharder.php. Accessed: March 31, 2023. 2023.

[6] Andrew Rukhin et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Tech. rep. SP 800-22 Rev. 1a. National Institute of Standards and Technology (NIST), 2010. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf.

# Appendix A: QCI's QRNG API

The QCI QRNG API supports a variety of formats and distributions. Samples are read directly from the device, and the raw output is processed via time-binning and ranging to create a specified number of bits. These bits are then cached until a request is made that uses them, after which they are deleted to prevent reuse.

To use the API, registration with QCI is required, and an access token is needed to access the service. Each request is limited to acquiring one million numbers, which can be returned as either numbers or binary strings. For testing purposes, binary strings were used to provide the appropriate input format for NIST and Dieharder. Consecutive samples were concatenated together to produce 32 bit samples.

```
import requests

BEARER_TOKEN = "YOUR_BEARER_TOKEN"
QRNG_URL = "https://api.qci-next.com/qrng/random_numbers"
headers = {"Authorization": f"Bearer {BEARER_TOKEN}"}
result = requests.post(f"{ip_address}/random_numbers",
                           json = {"distribution": "uniform_discrete",
                                   "n_samples": 10**6,
                                   "n_bits": 16,
                                   "output_type": "binary"},
                             headers = headers)
```

For further information on API please see API Quickstart Guide[1].