Quiz. 5

Problem 1

a) Write a Python/C++ program to generate **1M bytes** of cryptographically secure random numbers.

```python
import secrets
with open("random.bin", "wb") as file:
    file.write(secrets.token_bytes(1048576))
```

b) Run the NIST SP 800-22 statistical test on your **1M bytes** of binary cryptographically secure random numbers and analyze the test results to identify any deviations from the expected statistical properties of random numbers.

```
./assess 8388608
         G E N E R A T O R    S E L E C T I O N
         ---------------------------------------

  [0] Input File                 [1] Linear Congruential
  [2] Quadratic Congruential I   [3] Quadratic Congruential II
  [4] Cubic Congruential         [5] XOR
  [6] Modular Exponentiation     [7] Blum-Blum-Shub
  [8] Micali-Schnorr             [9] G Using SHA-1

 Enter Choice: 0


      User Prescribed Input File: random.bin

              S T A T I S T I C A L   T E S T S
              ---------------------------------

  [01] Frequency                       [02] Block Frequency
  [03] Cumulative Sums                 [04] Runs
  [05] Longest Run of Ones             [06] Rank
  [07] Discrete Fourier Transform      [08] Nonperiodic Template Matchings
  [09] Overlapping Template Matchings  [10] Universal Statistical
  [11] Approximate Entropy             [12] Random Excursions
  [13] Random Excursions Variant       [14] Serial
  [15] Linear Complexity

       INSTRUCTIONS
          Enter 0 if you DO NOT want to apply all of the
          statistical tests to each sequence and 1 if you DO.

 Enter Choice: 1

      P a r a m e t e r   A d j u s t m e n t s
      -----------------------------------------
  [1] Block Frequency Test - block length(M):         128
  [2] NonOverlapping Template Test - block length(m): 9
  [3] Overlapping Template Test - block length(m):    9
  [4] Approximate Entropy Test - block length(m):     10
  [5] Serial Test - block length(m):                  16
  [6] Linear Complexity Test - block length(M):       500
```

```
 Select Test (0 to continue): 1

 Enter Block Frequency Test block length: 65536

      P a r a m e t e r   A d j u s t m e n t s
      ----------------------------------------
  [1] Block Frequency Test - block length(M):         65536
  [2] NonOverlapping Template Test - block length(m): 9
  [3] Overlapping Template Test - block length(m):    9
  [4] Approximate Entropy Test - block length(m):     10
  [5] Serial Test - block length(m):                  16
  [6] Linear Complexity Test - block length(M):       500

 Select Test (0 to continue): 0

 How many bitstreams? 1

 Input File Format:
  [0] ASCII - A sequence of ASCII 0's and 1's
  [1] Binary - Each byte in data file contains 8 bits of data

 Select input mode:  1

   Statistical Testing In Progress.........

   Statistical Testing Complete!!!!!!!!!!!!

cat experiments/AlgorithmTesting/finalAnalysisReport.txt | grep 0/1
```

(a) **The frequency (monobit) test**.
The goal of this test is to see if the number of ones and zeros in a sequence is roughly the same as what would be expected from a random sequence.

(b) **Block-level frequency testing**
The goal of this test is to see if the frequency of ones in an M-bit block is about M/2, as would be predicted under the assumption of randomness.

(c) **The Run Test**
The runs test is used to check whether the number of runs of one and zeros of varying lengths is consistent with a random sequence. This test detects whether the oscillation between these zeros and ones is too fast or too slow.

(d) **Checks for the longest run-of-ones in a block**.
The goal of this test is to see if the length of the longest run of ones in the tested sequence matches the length of the longest run of ones that would be expected in a randomly generated sequence.

(e) **Binary Matrix Rank Test.**
This test checks for linear dependence among fixed-length substrings of the original sequence.

(f) **Discrete Fourier Transform (Spectral) Test**
The goal of this test is to identify periodic features (i.e., recurring patterns that are close together) in the tested sequence that would indicate a departure from the assumption of randomness.

(g) **The non-overlapping template matching test**.
The goal of this test is to identify generators that produce an excessive number of occurrences of a specific non-periodic (aperiodic) pattern.

(h) **Overlapping Template Matching Test**
The Overlapping Template Matching test focuses on the number of occurrences of pre-specified target strings. Both this test and the Non-overlapping Template Matching test in Section 2.7 use an m-bit window to look for a certain m-bit pattern.
The difference between this test and the one in Section 2.7 is that when the pattern is detected, the window only moves one bit before continuing the search.

(i) **Maurer's "universal statistical" test.**
The test determines whether the sequence can be greatly compressed without losing information.

(j) **Linear Complexity Test.**
The goal of this test is to see if the sequence is complicated enough to be deemed random.

(k) **The Serial Test**
The goal of this test is to see if the number of occurrences of the 2m m-bit overlapping patterns is roughly equal to what would be expected from a random sequence.

(l) **Approximate Entropy Test**.
The test compares the frequency of overlapping blocks of two consecutive/adjacent lengths (m and m+1) to the predicted result for a random sequence.

(m) **Cumulative Sum (Cusum) Test**
The test is designed to detect whether the cumulative sum of the partial sequences in the tested sequence is too large or too small in comparison to the expected behavior of that cumulative sum for random sequences.

(n) **Random Excursions Test**.
The goal of this test is to see if the number of trips to a specific state during a cycle differs from what one would expect from a random sequence.

(o) **Random Excursions Variant Test**.
Its goal is to detect deviations from the predicted behavior of a random sequence by counting the number of excursions to other states inside the sequence.

c) Extra credit: Find out a non-cryptographically secure random number generator, such as random (), to demonstrate its lack of safety. Then, propose modifications to enhance its security to generate cryptographically secure random numbers that meet the highest standards of security and reliability.

```
import random

with open("InsecureRandom.bin", "wb") as file:
    file.write(bytes(random.getrandbits(8) for _ in range(1048576)))
```

Result:

```
$ cat experiments/AlgorithmTesting/finalAnalysisReport.txt | grep 0/1
1 0 0 0 0 0 0 0 0 0 ---- 0/1 NonOverlappingTemplate
```

Modified:

```python
import random
import os

random_bytes = bytearray(random.randint(0, 255) for _ in range(1048576))
random_bytes += bytearray(os.urandom(1048576))

with open("secureRandom.bin", "wb") as file:
    file.write(bytes(random_bytes))
```

Result:
$ cat experiments/AlgorithmTesting/finalAnalysisReport.txt | grep 0/1