Name – Nithyapriya S
Roll No - CS21M510

ASSIGNMENT - 2

1. a. What is the maximum period obtainable from the following generator?
Xn+1 = (aXn) mod 2^4
b. What should be the value of a?
c. What restrictions are required on the seed?

With a as 1, the sequence is [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

With a as 2, the sequence is [1, 2, 4, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

With a as 3, the sequence is [1, 3, 9, 11, 1, 3, 9, 11, 1, 3, 9, 11, 1, 3, 9, 11, 1]

With a as 4, the sequence is [1, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

With a as 5, the sequence is [1, 5, 9, 13, 1, 5, 9, 13, 1, 5, 9, 13, 1, 5, 9, 13, 1]

With a as 6, the sequence is [1, 6, 4, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

With a as 7, the sequence is [1, 7, 1, 7, 1, 7, 1, 7, 1, 7, 1, 7, 1, 7, 1, 7, 1]

With a as 8, the sequence is [1, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

With a as 9, the sequence is [1, 9, 1, 9, 1, 9, 1, 9, 1, 9, 1, 9, 1, 9, 1, 9, 1]

With a as 10, the sequence is [1, 10, 4, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

With a as 11, the sequence is [1, 11, 9, 3, 1, 11, 9, 3, 1, 11, 9, 3, 1, 11, 9, 3, 1]

With a as 12, the sequence is [1, 12, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

With a as 13, the sequence is [1, 13, 9, 5, 1, 13, 9, 5, 1, 13, 9, 5, 1, 13, 9, 5, 1]

With a as 14, the sequence is [1, 14, 4, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

With a as 15, the sequence is [1, 15, 1, 15, 1, 15, 1, 15, 1, 15, 1, 15, 1, 15, 1, 15, 1]

a) From the above result, Maximum period is **$2^{4-2} = 4$**
b) From the above result, with a as **3,5,11**, the maximum period is achieved.
c) From the above result, with even seed, the sequence ends with 1 cycle and 0 is seen for the rest of the values. **So, seed must be odd**.

2. With the linear congruential algorithm, a choice of parameters that provides a full period does not necessarily provide a good randomization. For example, consider the following two generators:

$$X_{n+1} = (11X_n) \bmod 13$$
$$X_{n+1} = (2X_n) \bmod 13$$

Write out the two sequences to show that both are full period. Which one appears more random to you?

Let us start with an initial seed of 1. The first generator yields the sequence:

With a as 11, the sequence is [1, 11, 4, 5, 3, 7, 12, 2, 9, 8, 10, 6, 1, 11]

The second generator yields the sequence:

With a as 2, the sequence is [1, 2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1, 2]

Because of the patterns evident in the second half of the latter sequence, most people would consider it to be less random than the first sequence.

3. What RC4 key value will leave S unchanged during initialization? That is, after the initial permutation of S, the entries of S will be equal to the values from 0 through 255 in ascending order.

```
/* Initial Permutation of S */
j = 0;
for i = 0 to 255 do
        j = (j + S[i] + K[i]) mod 256;
        Swap (S[i], S[j]);
```

Here, S is {0,1,2,3,4,…}

If K[0] = 0, S[0] = 0, (j=0, K[0] =0, S[0] = 0, j=0+0+0)

If K[1] = 0, S[1] = 1, (j=0, K[1] =0, S[1] = 1, j=0+1+0=1)

If K[2] = 255, S[2] = 2, (j=1, K[2] =255, S[2] = 2, j=1+255+2=258 mod 256 = 2)

If K[3] = 254, S[3] = 3, (j=2, K[2] =254, S[3] = 3, j=2+254+3=259 mod 256 = 3)

…

…

…

And goes on

The first two bytes are zero; that is K[0] = K[1] = 0.

Thereafter, we have:

K[0] = 0;

K[1] = 0;

K[2] = 255;

K[3] = 254;

K[4] = 253;

…

K[254]= 3;

K[255]= 2;

4. RC4 has a secret internal state which is a permutation of all the possible values of the vector S and the two indices *i* and *j*.
a. Using a straightforward scheme to store the internal state, how many bits are used?
b. Suppose we think of it from the point of view of how much information is represented by the state. In that case, we need to determine how may different states there are, then take the log to base 2 to find out how many bits of information this represents. Using this approach, how many bits would be needed to represent the state?

a. Simply store i, j, and S
which requires $8 + 8 + (256 \times 8) = 2064$ bits

b. The number of states is $[256! \times 256^2] \approx 2^{1700}$.
Therefore, 1700 bits are required.

5. Alice and Bob agree to communicate privately via email using a scheme based on RC4, but they want to avoid using a new secret key for each transmission. Alice and Bob privately agree on a 128-bit key k. To encrypt a message m, consisting of a string of bits, the following procedure is used.

> **1. Choose a random 64-bit value v**
>
> **2. Generate the ciphertext c = RC4(v || k) ⊕ m**
>
> **3. Send the bit string (v || c)**

a. Suppose Alice uses this procedure to send a message m to Bob. Describe how Bob can recover the message m from (v || c) using k.

b. If an adversary observes several values (v1 || c1), (v2 || c2), transmitted between Alice and Bob, how can he/she determine when the same key stream has been used to encrypt two messages?

c. Approximately how many messages can Alice expect to send before the same key stream will be used twice? Use the result from the birthday paradox described in Appendix U.

d. What does this imply about the lifetime of the key k (i.e., the number of messages that can be encrypted using k)?

a. By taking the first 80 bits of v || c, we obtain the initialization vector, v.
Since v, c, k are known, the message can be recovered (i.e., decrypted) by computing

m = RC4(v || k) ⊕ c.

b. If the adversary observes that vi = vj for distinct i, j then he/she knows that the same key stream was used to encrypt both mi and mj. In this case, the messages mi and mj may be vulnerable to the type of cryptanalysis carried out in part (a).

c. Since the key is fixed, the key stream varies with the choice of the 80-bit v, which is selected randomly. Thus, after approximately   messages are sent, we expect the same v, and hence the same key stream, to be used more than once.

d. The key k should be changed sometime before 2^{40} messages are sent

6. In any use of pseudorandom numbers, whether for encryption, simulation, or statistical design, it is dangerous to trust blindly the random number generator that happens to be available in your computer's system library. [PARK88] found that many contemporary textbooks and programming packages make use of flawed algorithms for pseudorandom number generation. This exercise will enable you to test your system. The test is based on a theorem attributed to Ernesto Cesaro (see [KNUT98] for a proof), which states the following: Given two randomly chosen integers, *x* and *y*, the probability that gcd(*x*, *y*) = 1 is 6/p2. Use this theorem in a program to determine statistically the value of p. The main program should call three subprograms: the random number generator from the system library to generate the random integers; a subprogram to calculate the greatest common divisor of two integers using Euclid's Algorithm; and a subprogram that calculates square roots. If these latter two programs are not available, you will have to write them as well. The main program should loop through a large number of random numbers to give an estimate of the aforementioned probability. From this, it is a simple matter to solve for your estimate of p. If the result is close to 3.14, congratulations! If not, then the result is probably low, usually a value of around 2.7. Why would such an inferior result be obtained?

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <bits/stdc++.h>

#define NUM 1000000

//Random number generator program
long int Random_Number_Generator()
{
    long int rand1 = std::rand();
    return rand1;
}

//GCD using Euclids algorithm
long int GCD_Euclids_Algorithm(long int a, long int b)
{
    // Everything divides 0
    if (a == 0)
       return b;
    if (b == 0)
       return a;

    // base case
    if (a == b)
        return a;
    // a is greater
    if (a > b)
        return GCD_Euclids_Algorithm(a % b, b);
    return GCD_Euclids_Algorithm(b % a, a);
}

int main()
{
    long int rand1,rand2, gcd;
    long long result=0;
    double probability, pi;
    std::srand(static_cast<unsigned int>(std::time(NULL)));

    for (long long int i=1; i <= NUM; i++)
    {
        rand1= Random_Number_Generator();
        rand2=Random_Number_Generator();
        gcd = GCD_Euclids_Algorithm(rand1,rand2);
        if (gcd == long(1))
        {
            result++;
        }
     }

     probability = float(result) / NUM;
     pi = sqrt(6.0 /probability);

     printf("\nTotal samples Taken                    : %lu", NUM);
     printf("\nRandom numbers generated with GCD 1  : %lu", result);
     printf("\nProbability                            : %f", probability);
     printf("\nPi value                               : %f", pi);
     return 0;
}
```