CIS*2460 F13 Assignment 2:

Questions

1. a) Write a function `my.rand.generator(m,a,c,seed)` that produces a LCG function that takes an integer `n`.

   If `n > 0` the function produces *n* random integers between 0 and m – 1.
   If `n = -1`, the function returns the last random number that was returned when called last.
   If `n = -2`, the function returns `list(m, a, c)`.

   Test it out with various small parameter values (i.e. `m` should be between 10 and 200).

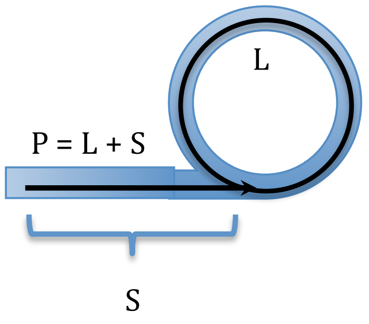
   For example:

```
> my.rand <- my.rand.generator(a = 15, c = 7, m = 128, seed = 13)
> my.rand(5)
[1]  74  93 122  45  42
> my.rand(7)
[1] 125  90  77  10  29  58 109
> foo.rand <- my.rand.generator(a = 15, c = 7, m = 128, seed = 13)
> foo.rand(-1)
[1] 13
> foo.rand(5)
[1]  74  93 122  45  42
> my.rand(7)
[1] 106  61  26  13  74  93 122
```

   b) Write a function that takes the LCG produced by `my.rand.generator(a,c,m,seed)` from 1a and calculates the periodicity (P) of the LCG as well as breaking it down into the loop length (L) and lead-in sequence length (S), where the loop length is number of random numbers that are repeated if the LCG is allowed to cycle and the lead-in sequence length is the number of random numbers produced before it goes into the loop. Note that P = L + S.
   Return the three values as a vector c(P, L, S).

   Example:

```
> my.rand <- my.rand.generator(14, 39, 100, 8)
> periodicity(my.rand)
  [1] 12, 10, 2
> my.rand <- my.rand.generator(14, 39, 100, 8)
> c(8, my.rand(12))
  [1] 8 51 53 81 73 61 93 41 13 21 33 1 53
```



   Hint: *There are many ways to solve this problem. The most obvious way is to store each random number produced, and see if it is repeated at each step. This is efficient if the loop is small and has a short lead-in sequence. However, if the random number generator has maximal periodicity it would take approximately $m^2/2$ comparisons with m storage.*

   *A faster approach for generators with large periodicity (although slower for those with short periodicity) also exists and is just as simple. Produce 2*m random numbers using the LCG. Use the frequency count function from Lab Quiz 1 to count how many times an integer (between 0 and m – 1) was produced. From this P, L and S can be calculated. You may implement either of these two algorithms.*

2. a) Using `my.rand.generator` from 1a, create a LCG of type " mod 2^b ", where b equals 15, that has maximal periodicity. It may start at a seed of your choice.
   Store the function produced in `my.rand` to answer further parts of this question.
   Notes: You must discuss your choice of a, c and m
   You must demonstrate that it has maximum periodicity (hint: use the function from 1b).

   b) Using the LCG from 2a stored in `my.rand`, write
   `my.runif(n, min = 0, max = 1, rand = my.rand)` that behaves the same as `runif()` except that it uses your random number generator.
   Notes: Remember to demonstrate that my.runif() has the correct behavior – don't just produce code.

3. a) Using the LCG from 2a, write a function `rbinaryDigits(n, rand = my.rand)` that produces random binary numbers by returning a 0 if the random integer is even and 1 if it is odd.

   Demonstrate that `rbinaryDigits()` cycles with a period of 2 (i.e. produces either 0, 1, 0, 1, 0, … or 1, 0, 1, 0, 1, 0, …) .

   Does this happen with the example LCG from 1a? Explain.
   Note: demonstrate the behavior don't just say "yes" or "no"

   Finally, write a version of `rbinaryDigits()` that does not have this behavior for the LCG of either 1a or 2a.

   b) Using the LCG from 2a demonstrate that bit k cycles after at most 2^k random number calls for k = 1, 2, 3, …, 7.
   Note: submit any code you used to perform the tests.

4. a) Implement the 7-bit and 14 bit Fibonacci linear feedback shift register that is discussed in the "Fibonacci LFSRs" and "Some polynomials for maximal LFSRs" sections of the Wikipedia article on linear feedback shift registers. It does not have to be efficient; it just has to work.

   What is its precision for each?
   Demonstrate that they have maximum periodicity in the same way you did for question1b.

   b) Try out a 14-bit Fibonacci linear feedback shift register with two different sets of four taps (keep the top tap as the top bit). Try them out with different seeds. What periodicity do you get?

Bonus. a) Design a 4-bit <u>combined</u> linear congruential generator using 3 regular 4-bit LCG (plus a fourth one for seed creation). Choose your three modulo values for maximum periodicity. In the write-up provide detailed explanations for the constants used.
What is its precision? What is its periodicity?

b) Compare periodicity and precision of this with the 7-bit FLFS generator from question 4. Which would you prefer to use and why?

Mark Breakdown

	Q1a	Q1b	Q2a	Q2b	Q3a	Q3b	Q4a	Q4b	Bonus	Total
Correct Answer	3	5	5	5	3	3	3	3	3	33
Code	3	5	1	3	2	5	3	3	3	25
Testing	5	3	3	3	3	5	5	5	3	32
Discussion	0	0	3	0	2	3	3	2	3	13
Total	11	13	12	11	10	16	14	13	12	100

Instructions

You must create a document actually answering the questions.

- Do not assume that if you hand in the code, that the TA will run it and answer the questions for you.

Submission Instructions

Hand in the R code (including function calls) you used to determine the answers.

- Your code must use the Monte Carlo simulator used in class, provided on the course moodle site.
- Probability proofs may be used to check the validity of your simulators, but cannot be used as the answer to the question; a simulation must be done.

All of your write-up (except program source code) should be either a Word document (.doc or .docx), a PDF document (.pdf), a text file (.txt) or a rich text file (.rtf).

All answer files and R files should be bundled together into a single file using a standard archiving or compression program (such as .zip, .tar, .tgz, etc.) and submitted via Moodle.