CS1083 Assignment #7 – Fall 2021

Due: Wednesday, November 3rd by 11:00pm, submitted to the Desire2Learn submission folder (see submission instructions below).

The purpose of this assignment is to help develop your understanding of recursion.

This assignment is to be done individually. If you have questions, direct them to a tutor/assistant during a help session in the "Faculty of Computer Science Student Success Centre" team or to your course instructor.

Computation of A Recursive Sequence: three solutions

A recursive mathematical sequence is one in which each term is defined using the previous term. In the sequence 1, 3, 1, -5, -7, 3, 17, 11 … the first two terms are the seed terms, and each subsequent term is calculated using a recursive formula requiring the previous 2 terms.

More specifically, if seq(n) is term *n* of the sequence, then the sequence can be defined mathematically as follows:

```
    seq(0) = 1
    seq(1) = 3
    seq(n) = seq(n-1) - seq(n-2)*2, n>1
```

a) Because the sequence is defined recursively, it is natural to write a recursive method to determine term *n* in the sequence. The file `Seq.java` contains the skeleton for a class containing a method to compute the numbers in the sequence. Save this file to your directory. Following the specification above, fill in the code for method seqR so that it recursively computes and returns term *n* in the sequence.

   The program `TestSeq.java` contains a simple driver that asks the user for an integer and uses the seqR method to compute that element in the sequence. Save this file to your directory and use it to test your seqR method. Be sure to test for boundary cases such as the first two terms (0 and 1). Test other values of n such as 5, 10 and 45. What do you notice if you try to find the 50th sequence?

The problem is that the seqR method is making lots and lots of recursive calls. To see this, add a print statement at the beginning of your seqR method that indicates what call is being computed, e.g., “In seqR(3)” if the parameter is 3. Run `TestSeq` again and enter 5—you should get a number of messages from your print statement. Determine the sequence of calls that generated these messages by drawing the call tree. Although it is expected to see seqR(3) and seqR(4) in the printout since seqR(5) is seqR(4) - seqR(3)*2, there are actually two calls to seqR(3)! This is because both seqR(4) and seqR(5) need seqR(3), so they both need to compute it.

i. Add a method `seqM` to your `Seq` class. Like seqR, `seqM` should be static and should take an integer and return an integer. Add an ArrayList as a class variable.
ii. Inside `seqM`, check if the ArrayList has been initialized, if not then you will need to initialize it to the correct capacity, and add the first two elements of the sequence.
iii. Check if the value requested has already been calculated and stored in the cache (ArrayList). If so then it does not have to be recalculated, if not then the value will need to be calculated and stored in the appropriate index of the cache (ArrayList).
iv. Modify your `TestSeq` class so that it also calls `seqM` and prints the result. You should get the same answers.

b) Another way to compute term *n* in the sequence is to use an iterative approach and store the values in an array as they are computed. Proceed as follows:

i. Add a method `seqI` to your `Seq` class that is static and takes an integer as its parameter and returns an integer.
ii. Inside `seqI`, create an array of integers that you will use to store the values of the sequence. Using a loop, compute each element of the array as the difference of the two previous elements (except the first two elements which will be assigned their values) up to the value passed in. When the array is full, its last element is the element requested. Return this value.
iii. Modify your `TestSeq` class so that it calls seqR and prints that result, then calls `seqM` and prints the result, and lastly calls `seqI` and prints that result. You should get all the same answers.

c) To determine the time that each method takes to execute we can use the nanoTime() method from the System class and calculate the difference in the time before and after the method call. Produce a table of values of the average execution times by putting all three method calls and the printout of their execution times inside of a loop. You can use NumberFormat class to keep values lined up in the tab separated columns. Use the sample code below in your main method and add the remaining method calls and time calculations to the body of the for loop.

```
NumberFormat form = NumberFormat.getInstance();
form.setMaximumFractionDigits(7);
form.setMinimumFractionDigits(7);

System.out.println("Execution Times in Milliseconds (ms)");
System.out.println("n\tRecursive\tSeq(n)\tMemoization" +
       "\tSeq(n)\tItertive\tSeq(n)");

for(int i = 15; i <= 35; i+=10){
   long start = System.nanoTime();
   int seqA = Seq.seqR(i);
   long end = System.nanoTime();
   double time = (double)(end-start)/1000000;


   System.out.print(i + "\t" + form.format(time) + "\t" +
        seqA);
   //add remaining method calls and time calculations
 }
```

Note: Code must be appropriately commented (including Javadoc comments).

For this assignment, only an electronic submission is required.

Your electronic submission (submitted via Desire2Learn) will consist of two files:

i. a written report. This should begin with a title page. As always, your title page should include: the course (CS 1083), the assignment number (Assignment #7 in this case), your full name, and your UNB student number. That should be followed by each part clearly identified with a section heading. Include:
   a) the source code for Seq.java and TestSeq.java and sample output.

   This written report should be prepared using a word processor; we recommend using Microsoft Word (i.e. create a .docx file for your report). Copy & paste your java source code & required output into the report document. Add appropriate headings for each part. Fix up the formatting where necessary, adjusting line breaks & page breaks to ensure that your document is easy to read. Use a monospaced font for your code to maintain proper indentation.) Once the report is complete and you've checked it all over, save the .docx file for your own records, and then **save**

a second copy in pdf format for submission. (Note: Be sure to open that file in a pdf viewer to verify that the pdf was generated correctly.) The **SINGLE pdf file** containing your report will be submitted to the appropriate assignment submission folder on Desire2Learn. (It is important that you submit a pdf file and NOT the original Word document. This pdf will allow the marker to write comments directly on your work to give you better feedback.)

Note: Please name this report as follows:
YourName_As7_Report.pdf

ii. an archive file (.zip) that contains all your work for this assignment. Make sure that your archive includes **all source code** (.java files, input and output files - in case the marker wishes to compile & run your code). This archive should be submitted as a single file to the appropriate submission folder on Desire2Learn.

Note: Please name this archive file as follows:
YourName_As7_Archive.zip