COMP610 — Data Structures and Algorithms

Lab 02

What to do

Do all the tasks, and answer all the questions. For code, make sure you follow the Code Laws. You can work individually or in pairs. Record non-code answers, as you may have to give open-class feedback on them.

Background

A (finite) *sequence* is a list of increasing numbers (called *terms*). For example, $1, 2, 3, 4, 5, 6, 7$ is a sequence. Every sequence has an *initial term*, which is the number that comes first; in the example sequence, that is 1. Sequences also have a *length*, which is the number of terms in the sequence; our example sequence has length 7.

There are many types of sequence. We will be working with the following three sequence types:

Odd sequence: Given an initial odd term a, an odd sequence of length n is $a, a+2, a+4, \ldots, a+2(n-1)$.

Arithmetic sequence: Given an initial term a and a *common difference* d, an arithmetic sequence of length n is $a, a+d, a+2d, \ldots, a+(n-1)d$.

Geometric sequence: Given an initial term a and a *scaling factor* $r \neq 1$, a geometric sequence of length n is $a, ar, ar^2, \ldots, ar^{n-1}$.

Our sequences will be on positive terms only.

A *series* is the sum of a sequence; the series of the example sequence is 28. To compute a series from a sequence, we have two choices:

- Add up the terms in a loop
- Use a *closed form*

These methods will both give the same answer, but have very different performance, which we will investigate.

Question 1

Download the Eclipse project for this lab, unzip it, and open it in Eclipse. Inside, you should find a `Sequence` interface, along with a set of tests, a `Profiler` class, and an `Odd` class. The `Sequence` interface requires two methods:

`public abstract long seriesLoop ():` Computes this sequence's series by looping.

`public abstract long seriesClosedForm ():` Computes this sequence's series using a closed form.

`Odd` implements the `Sequence` interface. The closed form that it uses states that an odd sequence of length n that starts with a has the series

$$n(a + n + 1)$$

1. Rename the packages to fit the Code Laws.

2. Run `OddTest`; ensure all tests pass.
3. Have a look at `Odd`'s implementations of `seriesLoop` and `seriesClosedForm`, then answer the following questions:
   (a) What is the time complexity of `seriesLoop` (with respect to n)? What about `seriesClosedForm`?
   (b) Which one do you think will be faster in practice? Why?
4. Read `Profiler`. Ensure that you understand what it is doing (you might have to look up `System.nanoTime`). Then, run it, and examine its table of results. Do they support your answers to the previous questions? If not, how are they different?

Question 2

We will now implement `Arithmetic`, which represents arithmetic sequences. For an arithmetic sequence, where a_1 is the initial term, a_n is the last term, and n is the number of terms in the sequence, that sequence will have the series

$$n(\frac{a_1 + a_n}{2})$$

1. Create an `Arithmetic` class, which implements `Sequence`. Use `Odd` as a guide.
2. Ensure all tests in `ArithmeticTest` pass.
3. Have a look at `Arithmetic`'s implementations of `seriesLoop` and `seriesClosedForm`, then answer the following questions:
   (a) What is the time complexity of `seriesLoop` (with respect to n)? What about `seriesClosedForm`?
   (b) Which one do you think will be faster in practice? Why?
4. Modify `Profiler` to test the performance of the `Sequence` methods of `Arithmetic`. Then, run it, and examine its table of results. Do they support your answers to the previous questions? If not, how are they different?

Question 3

We will now implement `Geometric`, which represents geometric sequences. For a geometric sequence, where a is the initial term, r is the scaling factor, and n is the number of terms in the sequence, that sequence will have the series

$$a(\frac{r^n - 1}{r - 1})$$

Take *extra* care implementing this one and answering these questions — it's trickier than it looks!

1. Create a `Geometric` class, which implements `Sequence`. Use `Arithmetic` as a guide.
2. Ensure all tests in `GeometricTest` pass.
3. Have a look at `Geometric`'s implementations of `seriesLoop` and `seriesClosedForm`, then answer the following questions:
   (a) What is the time complexity of `seriesLoop` (with respect to n)? What about `seriesClosedForm`?
   (b) Which one do you think will be faster in practice? Why?
4. Modify `Profiler` to test the performance of the `Sequence` methods of `Geometric`. Then, run it, and examine its table of results. Do they support your answers to the previous questions? If not, how are they different?