CS2030 Programming Methodology
Semester 2 2021/2022

30 & 31 March 2022
Problem Set #9

1. Consider two following two methods `foo` and `bar`:

```
Optional<Integer> foo(Integer x) {
  if (x == null) {
    return null;
  }
  return Optional.ofNullable(x*2);
}

Maybe<Integer> bar(Integer x) {
  if (x == null) {
    return null;
  }
  return Maybe.some(x*2);
}
```

   Using these methods, test if `Maybe` and `Optional` obey the three Monad laws.

   i. Left Identity Law: `Monad.of(x).flatMap(y -> f(y))` is equivalent to `f(x)`
   ii. Right Identity Law: `monad.flatMap(y -> Monad.of(y))` is equivalent to `monad`
   iii. Associative Law: `monad.flatMap(x -> f(x)).flatMap(x -> g(x))` is equivalent to `monad.flatMap(x -> f(x).flatMap(x -> g(x)))`

   Now test if `Maybe` and `Optional` obey the two Functor laws.

   i. Preserving Identity: `functor.map(x -> x)` is equivalent to `functor`
   ii. Preserving Composition: `functor.map(x -> f(x)).map(x -> g(x))` is equivalent to `functor.map(x -> g(f(x))`

   From this, what can we say about `Maybe` and `Optional`?

2. What is the outcome of the following stream pipeline?

```
Stream.of(1, 2, 3, 4)
    .reduce(0, (result, x) -> result * 2 + x);
```

   What happens if we parallelize the stream? Explain.

3. By now you should be familiar with the Fibonacci sequence where the first two terms are defined by $f_1 = 1$ and $f_2 = 1$, and generation of each subsequent term is based upon the sum of the previous two terms. In this question, we shall attempt to parallelize the generation of the sequence.

   Suppose we are given the first $k = 4$ values of the sequence f_1 to f_4, i.e. $1, 1, 2, 3$.

   To generate the next $k - 1$ values, we observe the following:

$$\begin{array}{rclcrcl} f_5 & = & f_3 + f_4 & = & f_3 + f_4 & = & f_1 \cdot f_3 + f_2 \cdot f_4 \\ f_6 & = & f_4 + f_5 & = & f_3 + 2f_4 & = & f_2 \cdot f_3 + f_3 \cdot f_4 \\ f_7 & = & f_5 + f_6 & = & 2f_3 + 3f_4 & = & f_3 \cdot f_3 + f_4 \cdot f_4 \end{array}$$

   Notice that generating each of the terms f_5 to f_7 only depends on the terms of the given sequence. This actually means that generating the terms can now be done in parallel! In addition, repeated application of the above results in an exponential growth of the Fibonacci sequence.

   You are now given the following program fragment:

```
static BigInteger findFibTerm(int n) {
    List<BigInteger> fibList = new ArrayList<>();
    fibList.add(BigInteger.ONE);
    fibList.add(BigInteger.ONE);

    while (fibList.size() < n) {
        generateFib(fibList);
    }
    return fibList.get(n-1);
}
```

   (a) Using Java parallel streams, complete the `generateFib` method such that each method call takes in an initial sequence of k terms and fills it with an additional $k - 1$ terms. The `findFibTerm` method calls `generateFib` repeatedly until the n^{th} term is generated and returned.

   (b) Find out the time it takes to complete the sequential and parallel generations of the Fibonacci sequence for $n = 50000$.