CPSC 331 — Solutions for Tutorial Exercise #5

The questions in this exercise concern an algorithm that solves the following problem — which extends the computational problem considered in Tutorial Exercise #3 in the same way as the computational problem considered during Lecture #4 extended the problem considered during Lecture #2:

Fibonacci Pair Computation — Safe Version

Precondition: An integer n is given as input.

Postcondition: If $n \geq 0$ then an array with length 2, whose first and second elements are the Fibonacci numbers F_n and F_{n+1} respectively, are returned as output. An `IllegalArgumentException` is thrown otherwise.

1. You were asked to modify the algorithm considered in Tutorial Exercise #3 (in a reasonably simple way) to produce an algorithm that solves the "Fibonacci Pair Computation — Safe Version" problem.

   Solution: Essentially the same changes can be made here as were made when modifying the algorithm from Lecture #2 to produce the algorithm considered in Lecture #4: First — if has not been done already — move the statement "`return F`" up, so that it appears at the end of each of the blocks currently being considered. Then replace the final `else` block with one incorporating a test to check whether n is positive, so that the completion for the case that $n \geq 0$ can be carried out, but the case that $n < 0$ can also be detected and handled. (The statement "`return F`" is then correctly handled, because it should be not be executed in this last case.)

   The resulting algorithm is as shown in Figure 1 on page 2.

2. You were asked whether the list of "unforeseen consequences" to be considered for this problem (and algorithm) is different from the problem and algorithm in Lecture #4.

   Answer: No, it is probably not.

   One notable difference is that "unforeseen consequences" concerning resource limitations might be easier to detect — for the problem and the algorithm in this tutorial exercise instead of the ones in the lecture: The algorithm in the lecture was so slow that it would more often be true that the computation would run for an unacceptable length of time, before being concluded, without any exceptions being thrown.

3. You were next asked to carry out ***Black Box Testing***, by describing a set of tests that should be included in a test plan for *any* implementation of an algorithm that solves the "Fibonacci Pair Computation — Safe Version" problem.

```
integer[] fibPair (integer n) {
 1. integer[] F := new integer[2]
 2. if (n == 0) {
 3.   F[0] := 0
 4.   F[1] := 1
 5.   return F
 6. } else if (n >= 1) {
 7.   integer[] oldF := fibPair(n − 1)
 8.   F[0] := oldF[1]
 9.   F[1] := oldF[0] + oldF[1]
10.   return F
    } else {
11.   throw an IllegalArgumentException
    }
}
```

Figure 1: An Algorithm for the "Fibonacci Pair Computation — Safe Version" Problem

Solution: This is almost the same as the set of tests described in Lecture #4 (when black box testing) because the input is the same and the output depends on the definition of Fibonacci numbers. That is, the number of tests listed and the inputs do not need to change. Of course, the outputs must change, because a different problem is being solved.

Input	**Expected Output**
$\mathtt{n} = -1$	Exception `IllegalArgumentException`
$\mathtt{n} = 0$	An integer array with length two and with entries $F_0 = 0$ and $F_1 = 1$
$\mathtt{n} = 1$	An integer array with length two and with entries $F_1 = 1$ and $F_2 = 1$
$\mathtt{n} = 2$	An integer array with length two and with entries $F_2 = 1$ and $F_3 = 2$

4. Finally, you were asked to carry out ***White Box Testing*** in order to design additional tests that should be included in a test plan for the algorithm designed when answering Question #1.

   Solution: The required tests will depend on how answered Question #1. However, if your answer was similar to the one in these solutions then you should find that every statement in the algorithm (and both the `if` and `else` part of the test) are executed out when the tests listed in the answer for Question #3 are carried out.

It is a good idea to add additional tests that will cause the body of the `while` loop in the algorithm to be executed various numbers of times.

There are lots of different tests you could choose! Tests corresponding to the "white box tests"' included in the example in Lecture #4 would be as follows.

Input	**Expected Output**
$n = 3$	An integer array with length two and entries $F_3 = 2$ and $F_4 = 3$
$n = 4$	An integer array with length two and entries $F_4 = 3$ and $F_5 = 5$
$n = 5$	An integer array with length two and entries $F_5 = 5$ and $F_6 = 8$
$n = 10$	An integer array with length two and entries $F_{10} = 55$ and $F_{11} = 89$
$n = 46$	An integer array with length two and entries $F_{46} = 1,836,311,903$ and $F_{47} = 2,971,215,073$
$n = 47$	An integer array with length two and entries $F_{47} = 2,971,215,073$ and $F_{48} = 4,807,526,976$