Warm-up questions

1. What are the two principal characteristics of a recursive algorithm?

2. Recursion is..

Answer	**theoretically powerful and often used in algorithms that could benefit from recursive methods**
	theoretically interesting but rarely used in actual programs
	theoretically uninteresting and rarely used in programs
X	theoretically powerful and often used in algorithms that could benefit from recursive methods

3. True or false: All recursive functions can be implemented iteratively
   True

4. True or false: if a recursive algorithm does NOT have a base case, the compiler will detect this and throw a compile error?
   False

5. True or false: a recursive function must have a void return type.
   False

6. True or False: Recursive calls are usually contained within a loop.
   False

7. True or False: Infinite recursion can occur when a recursive algorithm does not contain a base case.
   True

8. **Which of these statements is true about the following code?**

```
int mystery(int n)
{
        if (n>0) return n + mystery(n-1);
        return 0;
}
```

Your answer	**The base case for this recursive function is an argument with the value zero.**
	The base case for this recursive method is an argument with any value which is greater than zero.
X	The base case for this recursive function is an argument with the value zero.
	There is no base case.

9. **List common bugs associated with recursion?**

1.	Not including a base case
2.	Not making the function smaller to reach the base case
3.	Making an infinite loop
4	Not calling the function recursively

10. **What method can be used to address recursive algorithms that excessively recompute?**
    Use a base case to prevent the recursive algorithms that excessively recompute and make sure the base case is reached by making the algorithm smaller. Also use many base cases to prevent excessive re-computation.

Fibonacci

The Fibonacci numbers are a sequence of integers in which the first two elements are 0 and 1, and each following element is the sum of the two preceding elements:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, and so on…

The Nth Fibonacci number is output with the following function:

$$fib(n) = fib(n-1) + fib(n-2) \rightarrow \text{for } n > 1$$

$$fib(n) = 1 \rightarrow \text{for } n = 0, 1$$

The first two terms of the series are 0, 1.

For example: fib(0) = 0, fib(1) = 1, fib(2) = 1

Exercises

1. Below is an iterative algorithm that computes Fibonacci numbers. Write a recursive function to do the same.
2. Test both algorithms with various sizes of Ns. What do you find?
3. What is the time complexity of both functions?

Iterative Fibonacci

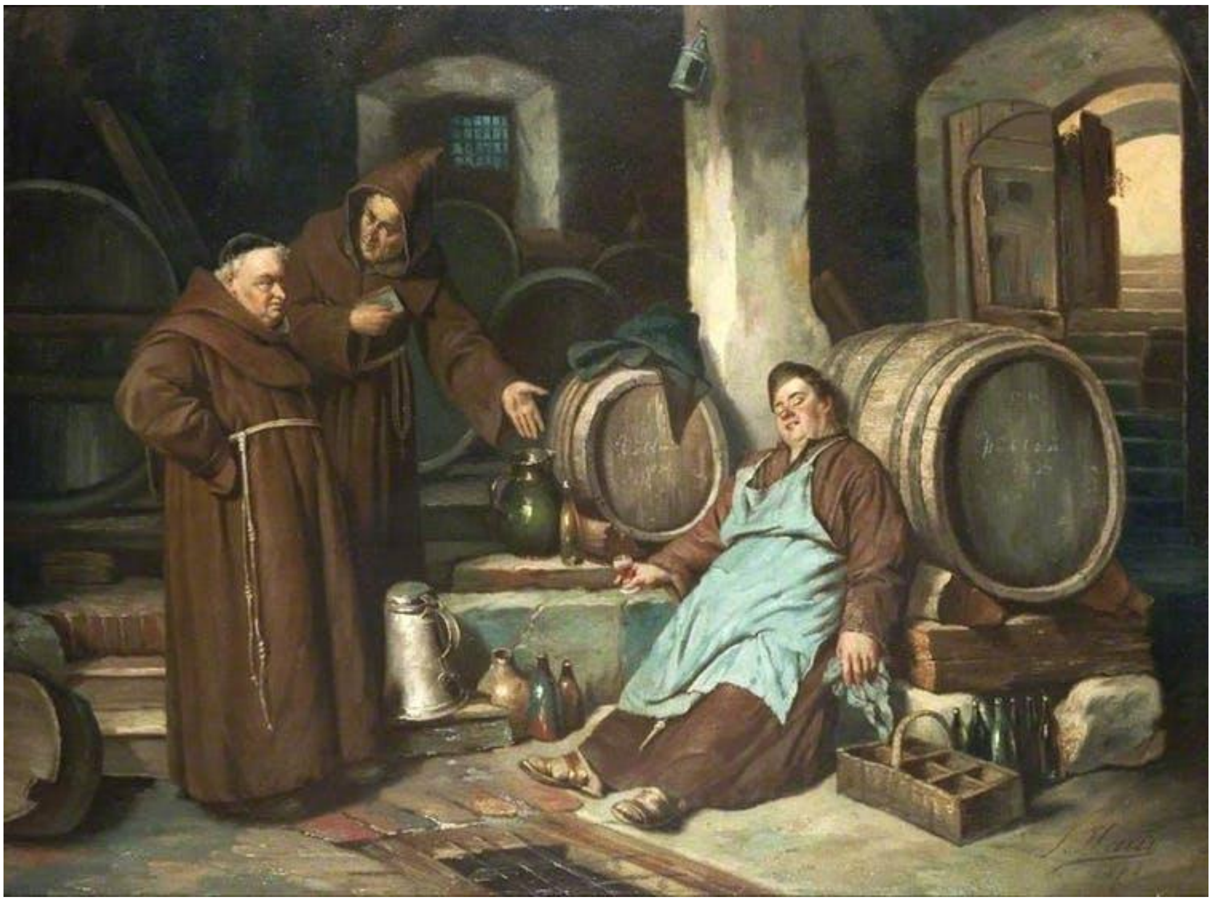
```
static int fibonacciIterative(int n){
  if (n<=1)
      return 1;

  int fib = 1;
  int prevFib =  1;

  for (int i = 2; i < n; i++) {
   int temp = fib;
   fib = fib + prevFib;
   prevFib = temp;
  }
  return fib;
 }

 public static void main (String args[])
    {
    int n = 9;
    System.out.println(fibonacciIterative(n));
    }
```

Hanoi - The Monks need your help!



Convert the pseudo-code into java and add your own output instructions so junior monks can learn how to perform the legal moves in the Tower of Hanoi so they can end the world.

There are two rules:

- Move only one disc at a time.
- Never place a larger disc on a smaller one.

Tasks:

1. Implement Hanoi in java
2. Test with various size disks
3. Output the moves for the monks as step-by-step instructions so the monks can end the world

Pseudocode for Hanoi

```
towersOfHanoi(disk, source, dest, auxiliary):
IF n == 0, THEN:
    move disk from source to dest
ELSE:
   towersOfHanoi(disk - 1, source, auxiliary, dest)
   towersOfHanoir(disk - 1, auxiliary, dest, source)
```

```
END IF
```

*alternative Palindrome or Factorial (iterative and recursive solution)