Practical 3: Recursion

What am I doing today?
Today's practical focuses on 3 things:
***pre-requisite: Github classroom set up and commits.**

1. Quick questions about Recursion
2. Comparing an iterative fibonacci algorithm to a recursive one
3. Help the monks solve the Towers of Hanoi

Instructions
Try all the questions. Ask for help from the demonstrators if you get stuck.
Solutions will be posted afterward.
*You can re-use the helper classes from previous weeks.

*****Grading: Remember to commit your work to your repository and push to the main branch on the origin.**

Warm-up questions

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

The first principle of recursion is the base case and the second is dividing the problem into smaller problems and so getting closer to the base case.

2. Recursion is..

Answer	
	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 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?**

	Not including a base case
	Not reducing the value passed, not getting it closer to base case each time
	Very slow performance due to unnecessary re-computations
	If recursion does not terminate properly, you might run out of memory

10. **What method can be used to address recursive algorithms that excessively recompute?**

Memorizing the output of a function call can be used to drastically cut down the amount of time a recursive function will recompute. This will ask as the memory of the function so it doesn't have to start from scratch each time its called, therefore cutting down on the amount of time it will take to complete this recursive function.

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

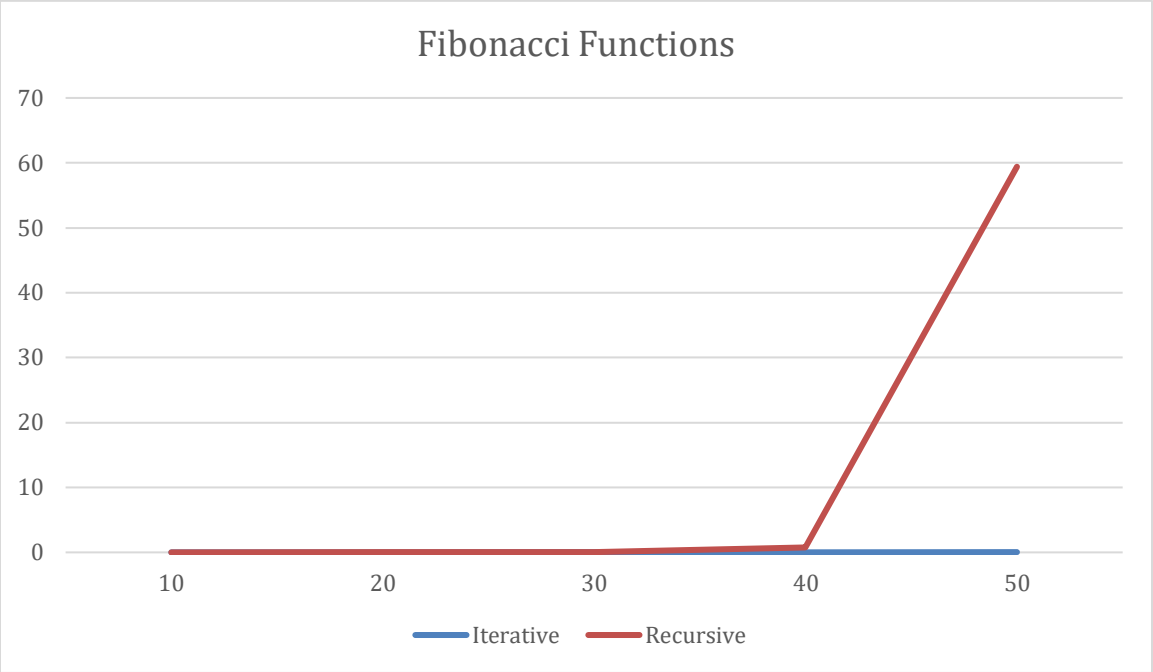
1. Below is an iterative algorithm that computes Fibonacci numbers. Write a recursive function to do the same.
2. Test both algorithms with various sizes of Ns. What do you find?
3. What is the time complexity of both functions?

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));
    }
```

Answers:

Q2.

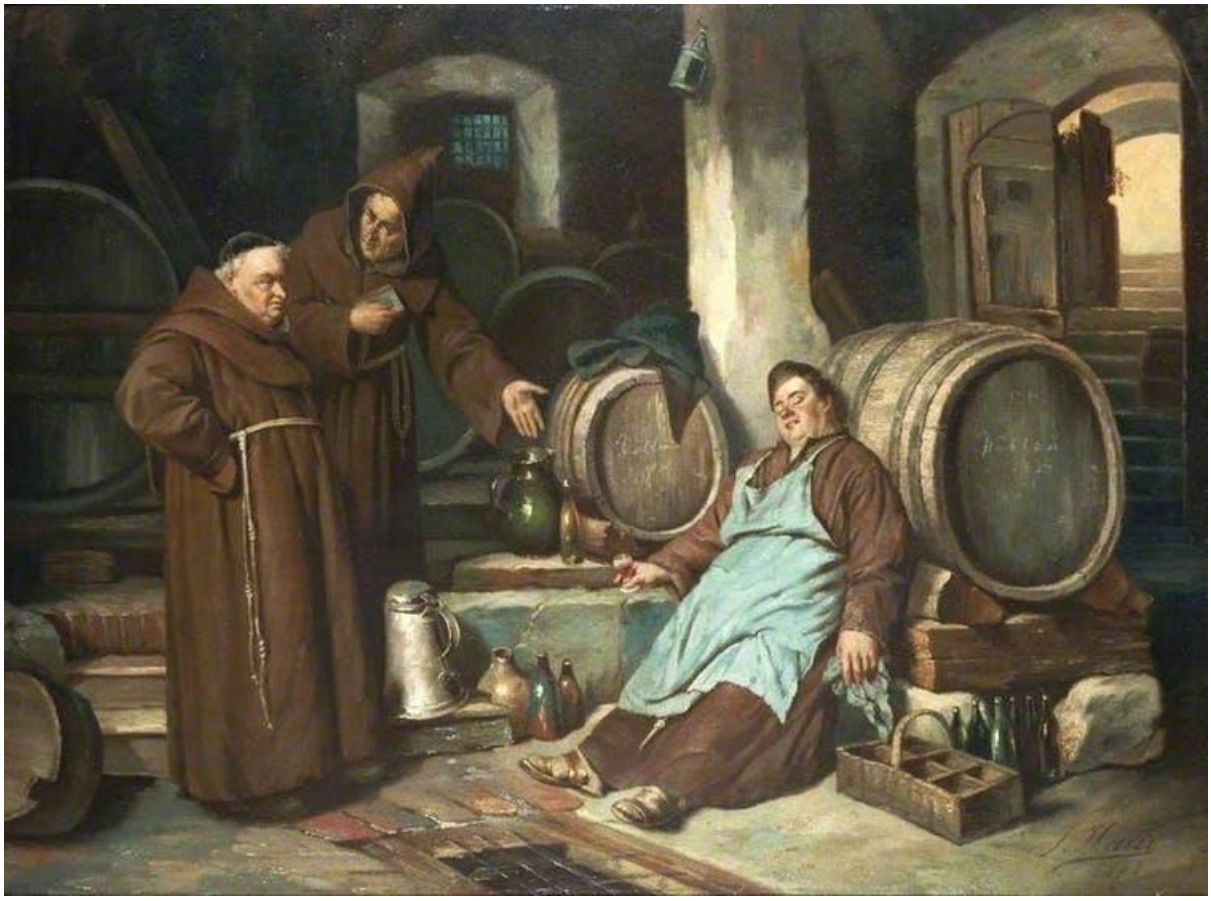


During my testing I found that the recursive function performed well for the first 40 integers then it took an exponentially more time for bigger numbers. The iterative function however, did really well and was very efficient as it never got close to taking more than a second. These results surprised me as I thought the recursive function would take considerably less time.

Q3. Iterative function = O(n) , as it only uses one for loops with incremented values.

Recursive function = O(2^n) , time complexity is exponential as calling two functions and adding them together on smaller numbers means that its incredibly inefficient as the previous result is not remembered.

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)

Answer:
Q2.
Time complexity is O(2^n), exponential.