Practical 3: Recursion

What am I doing today?
Today's practical focuses on 3 things:

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.

***Grading: Remember** if you complete the practical, add the code to your GitHub repo which needs to be submitted at the end of the course **for an extra 5%**

Warm-up questions

1. What are the two principal characteristics of a recursive algorithm?
   There use to be 2 parts: a base case and inductive 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	
False, the base case is when n <= 0	The base case for this recursive method is an argument with any value which is greater than zero.
Partially true, it will be any	The base case for this recursive function is an argument with the

argument with a value equal or less than 0.	value zero.
False	There is no base case.

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

	Stack Overflow. We run out of memory for large problems.
	Infinite Recursion, there is no base case.
	Non-termination.

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

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

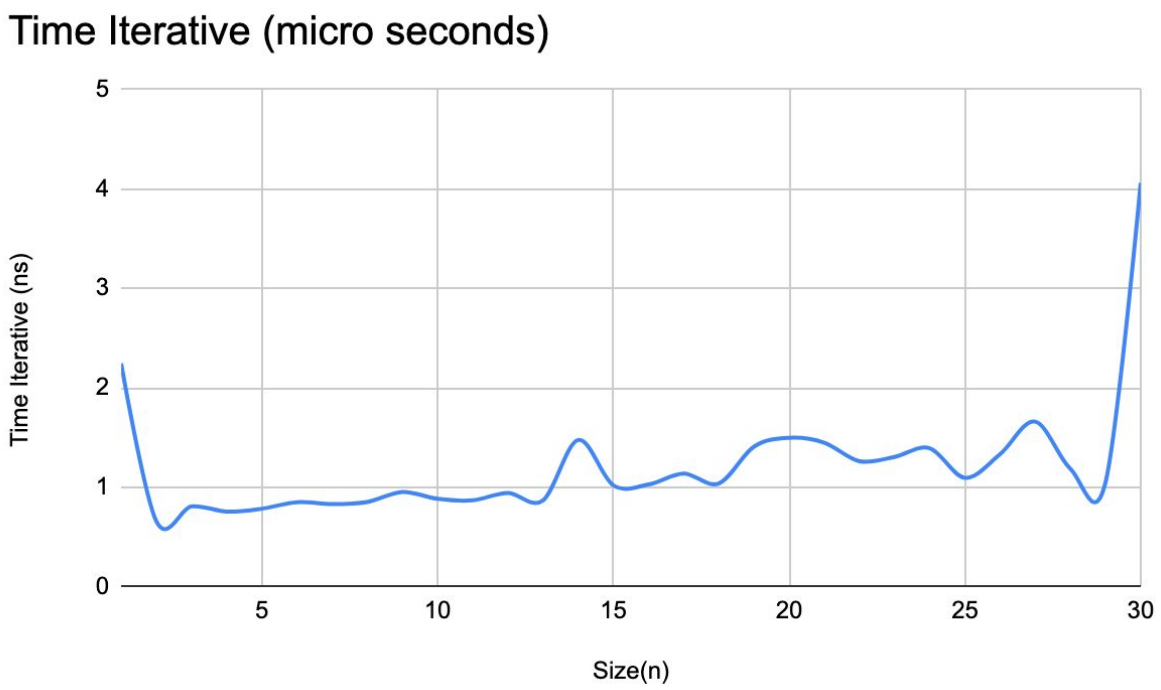
1. Below is an iterative algorithm that computes Fibonacci numbers. Write a recursive function to do the same.
2. Test both algorithms with various sizes of Ns. What do you find?
3. What is the time complexity of both functions?

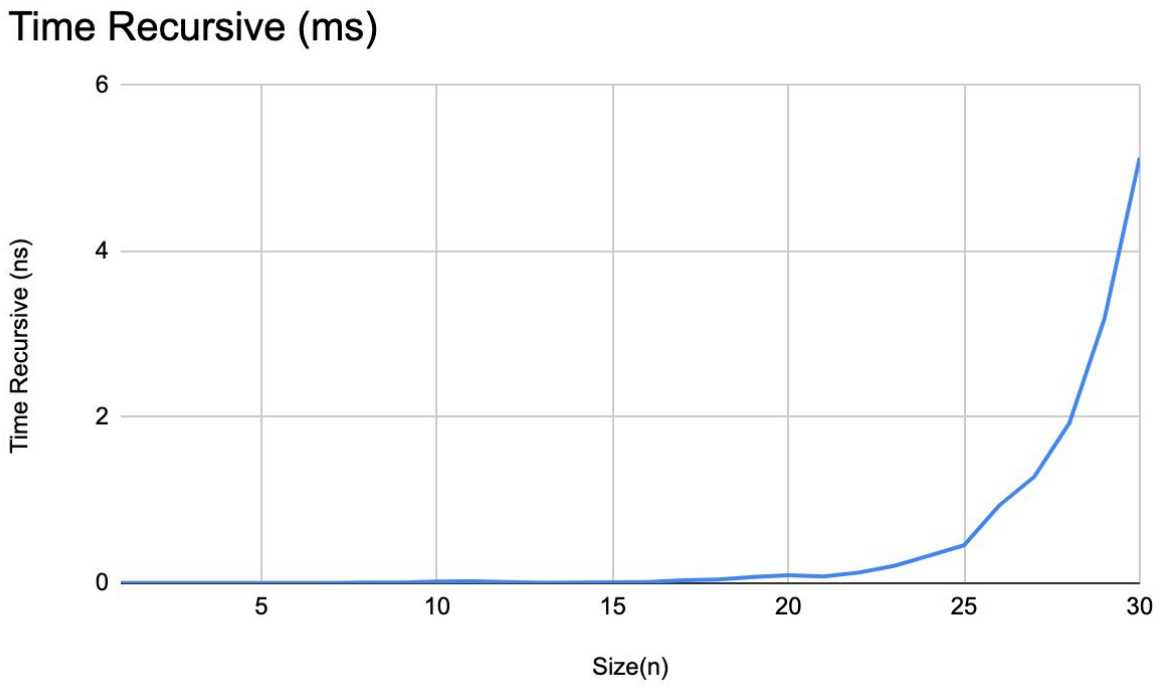
Iterative Fibonacci

```
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));
    }
```

Time Analyses Fibonacci (Experimental)

Size(n)	Result	Time Iterative (ns)	Time Recursive (ns)
1	1	2245	1024
2	1	658	447
3	2	807	685
4	3	755	945
5	5	784	1510
6	8	849	2228
7	13	830	3236
8	21	851	6733
9	34	953	9036
10	55	883	19639
11	89	868	24101
12	144	943	13843
13	233	868	4066
14	377	1475	8870
15	610	1019	10208
16	987	1028	15274
17	1597	1138	35506
18	2584	1038	44696
19	4181	1408	74470
20	6765	1497	97216
21	10946	1448	80921
22	17711	1262	127443
23	28657	1305	207481
24	46368	1392	329898
25	75025	1095	456809
26	121393	1332	934949
27	196418	1660	1283429
28	317811	1186	1929555
29	514229	1049	3191047
30	832040	4060	5128985





We can observe an exponential increase in time with the increase of the input size. Also, we can see that the recursive implementation is much more inefficient (around 1000 times). This can be produced by optimizations in the compiler or different factors in the machine executed. The important here is to observe that in both cases the tendency of the time is to increase exponentially with the size of the input.

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
```