Part 1

Warm-up questions

1. **What are the two principal characteristics of a recursive algorithm?**
   A. 1. A base case
      2. A recursive call i.e. a call to itself.

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**
   A. 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?**
   A. False

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

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

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

1.	Infinite recursion if there is no base case
2.	The recursive call might not actually make the problem smaller and therefore never catch up with the base case
3.	Might use up too much memory
4.	Might use up too much time because of too many recomputations

10. **What method can be used to address recursive algorithms that excessively recompute?**
    A. We can store the value of frequent recursive calls.

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?

Exercise 1)

See Fibonacci.java

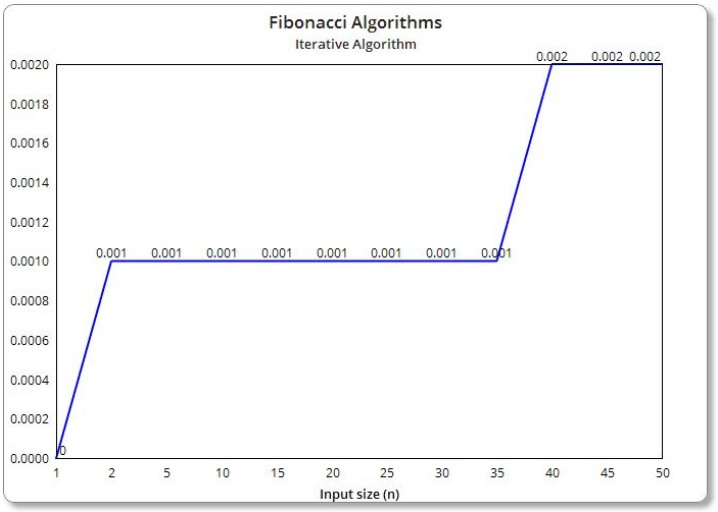
Exercise 2)

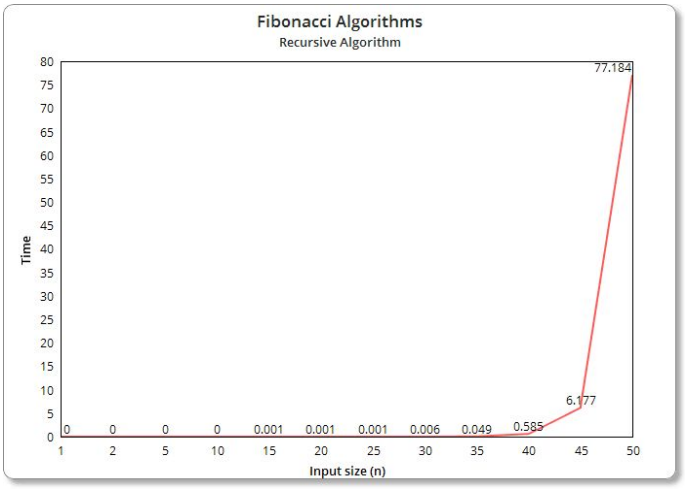
Testing for Fibonacci Iterative and Recursive

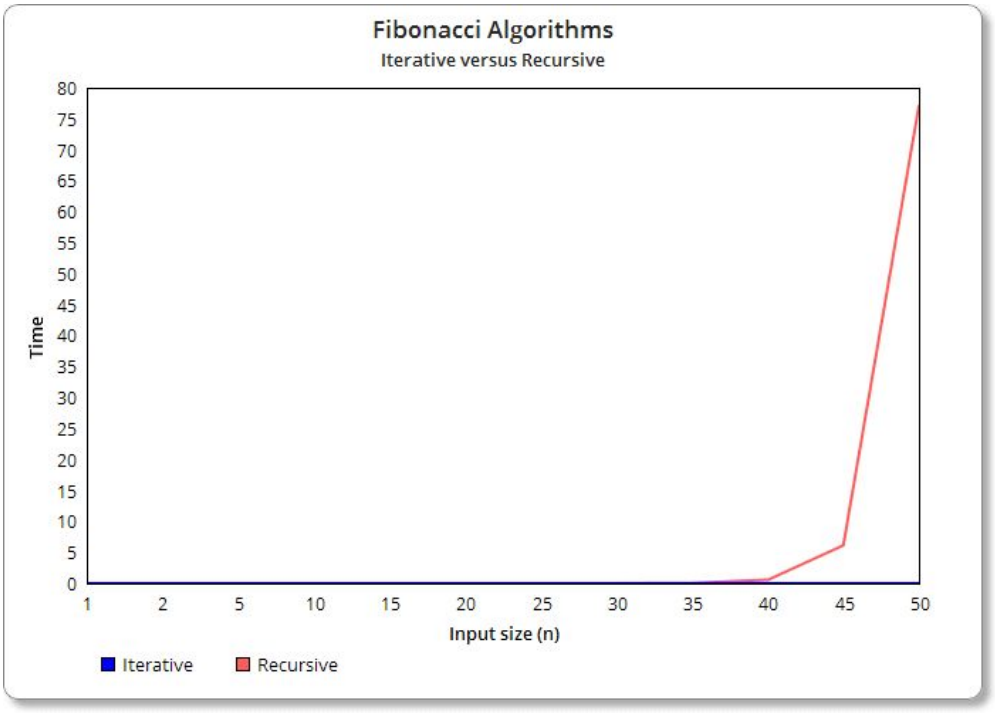
Input size (n)	Time (Iterative)	Time (Recursive)	Result
1	0.0	0.0	1
2	0.001	0.0	1
5	0.001	0.0	5
10	0.001	0.0	55
15	0.001	0.001	610
20	0.001	0.001	6765
25	0.001	0.001	75025
30	0.001	0.006	832040
35	0.001	0.049	9227465
40	0.001	0.585	102334155

45	0.002	6.177	1134903170
50	0.002	77.184	298632863

While the recursive algorithm for the Fibonacci sequence is adequate for smaller inputs, it becomes unmanageable as n gets larger. This is most likely due to the excessive recomputations. The iterative algorithm is therefore the preferable algorithm in this case. This can be seen in the following graphs:







Exercise 3)

- Iterative Fibonacci Time Complexity → O(n) (single for loop)
- Recursive Fibonacci Time Complexity → $O(2^n)$ (two recursive calls)

Hanoi

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)

Exercise 1

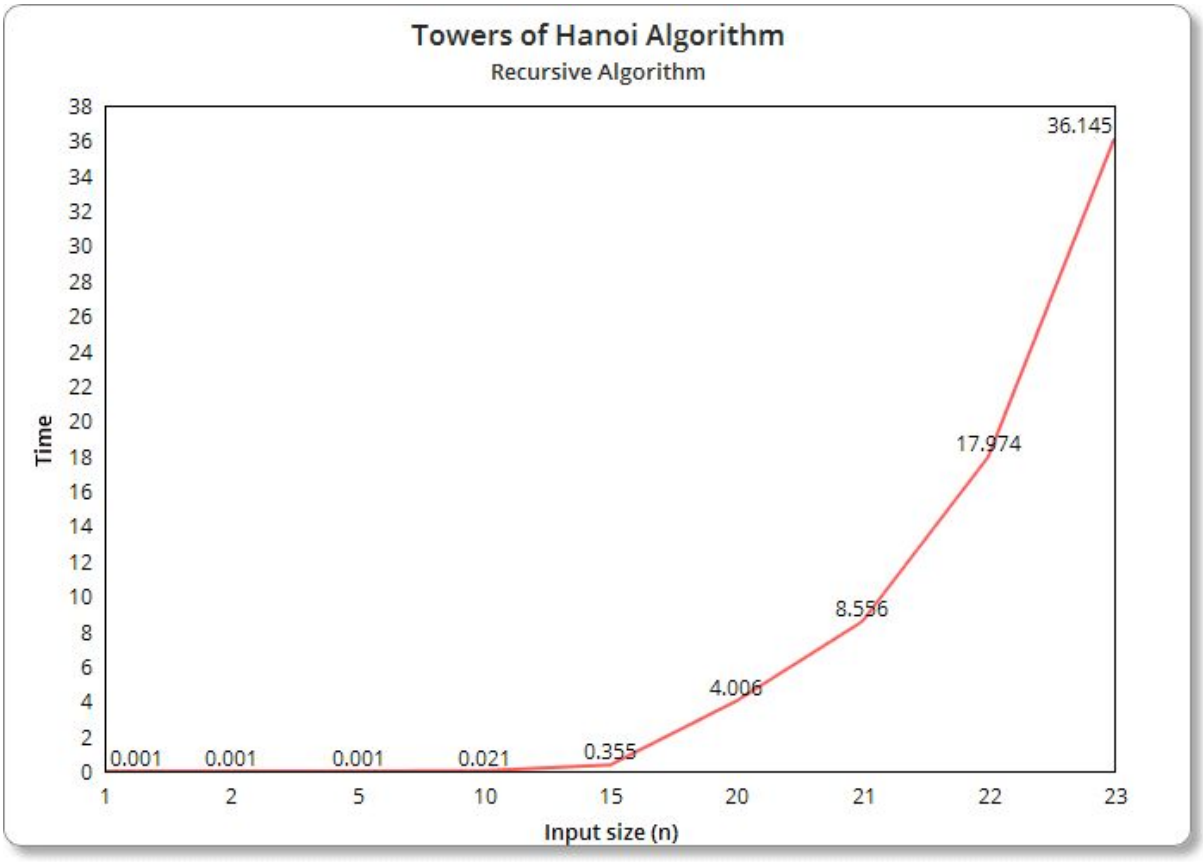
See Hanoi.java

Exercise 2

Towers of Hanoi algorithm testing

Input size (n)	Time
1	0.001
2	0.001
5	0.001
10	0.021
15	0.355
20	4.006

21	8.556
22	17.974
23	



It's easy to see from the chart above that the time complexity of this algorithm is exponential.

Exercise 3)

The step by step instructions for n = 3 (where A is the source rod, B is the auxiliary rod and C is the dest) are:

- Move disk 1 from A to C
- Move disk 2 from A to B
- Move disk 1 from C to B
- Move disk 3 from A to C
- Move disk 1 from B to A
- Move disk 2 from B to C
- Move disk 1 from A to C