Homework 1

Problem 1 - 15 points

Define a `Rectangle` class (in `Rectangle.java`) that implements `RectangleInterface`. In addition, you should implement the `Comparable` interface, such that `Rectangle` objects can be compared by their **perimeter**. That is, your class signature should be:

```
public class Rectangle implements RectangleInterface, Comparable<Rectangle>
```

You must have this class signature for full credit.

Finally, you must have a **constructor** that takes in doubles of the length and width as parameters:

```
public Rectangle(double length, double width)
```

As is implied by the parameters to the constructor, your instance variables should also be doubles.

Problem 2 - 20 points

Write a `GenericMethods` class (in `GenericMethods.java`) that implements `GenericMethodsInterface`.
There are two methods you must implement:

1. `linearSearch`: Iterate through the array linearly and search for a value equal (again, in the `Comparable` sense) to `x`. This must run in O(n) time. If the value is not found in the array, return -1. Else, return the index in the array where the value was found.
2. `binarySearch`: Implement a **recursive** binary search to find a value equal to `x`. Hint: The public `binarySearch` method, itself, should not be recursive. The private helper method should be. This private helper method with additional parameters should be called from the public `binarySearch` method. This must run in O(log n) time. If the value is not found in the array, return -1. Else, return the index in the array where the value was found.

To test your code, you might try creating a file with a main method that does the following:

- Build an array of `Rectangle` objects
- Demonstrate `linearSearch` functionality on the array
- Sort the array with `Arrays.sort` (remember, input to `binarySearch` must be sorted)
- Demonstrate `binarySearch` functionality on the array

This is *not* required, but will be useful if you're looking for ways to guarantee your implementation is correct.
Your `GenericMethods` class must implement the interface for full credit.

Problem 3 - 15 points

In a file called `BigO.java`, implement `BigOInterface` and write methods that have the following runtime requirements:

- `cubic` must be O(n^3)
- `exp` must be O(2^n)
- `constant` must be O(1)

Where n is an integer which is passed into the function. The methods can contain any code fragments of your choice. However, in order to receive any credit, the runtime requirements must be satisfied. As in the previous two problems, you must implement the interface to receive full credit.
In addition to writing the code fragments, we will explore their actual runtimes, to observe big-O in action in the real world. In a file called `Problem3.java` write a `main` method which runs the code with various values of `n` and measures their runtime. Then, discuss the results you observed briefly in a file called `Problem3.txt`.
Please run each of your methods with multiple different values of n and include the elapsed time for each of these runs and the corresponding value of n in `Problem3.txt`.
To properly time code runtime in Java, we must disable compiler optimizations. We do this by running the code with the `-Xint` flag, for example: `java -Xint Problem3`. The easiest way to time your run is to wrap each fragment with the following code:

```
long startTime = System.nanoTime();
// YOUR CODE HERE
long endTime = System.nanoTime();
```

The elapsed time is the difference between these two variables.
Note also that you may see slightly erratic results due to noise and memory allocation delays. This may be one of the factors you discuss in addressing outliers.