Kerry Ojakian's CSI 33 Class
Due Date: Tuesday September 24 in class

HW #1

General Instructions: Write problems 1 to 5 on paper to hand in. Problem 6 is a program - put it in your Dropbox in the folder HomeWork, in a new folder called HW01.
Remember: Do not copy your work from someone or from some online source.

The Assignment

1. (a) If a computer is capable of performing one billion operations per second, how long would it take to execute an algorithm that requires 2^n operations for an input of $n = 100$ elements?

   (b) If a computer is capable of performing one billion operations per second, how long would it take to execute an algorithm that requires n^2 operations on an input of $n = 1,000,000$. How long would it take if the algorithm requires n^3 operations?

2. Textbook Chapter 1, page 36 - Do exercise 8.

3. Consider the following list: $[3, 7, 8, 10, 11, 15, 20]$.

   Carry out the Binary Search algoirthm on the list, first when searching for the number 7, then when searching for the number 18. In both cases, show every step (i.e. the result of every recursive call to Binary Search).

   Note: Use the same convention we did in class - when finding the middle element of an even length list, choose the left position from the 2 possible middle positions

4. Consider the following 5 functions:

   $$(0.01)(1.2)^n, \quad 10n^2, \quad 1000n, \quad 100\log_2(n), \quad n^3.$$

   Put them in order, in terms of Big O growth rate. Justify your answer by finding a single number n for which their values follow this order when this number is plugged in (justifying your answer).

5. Show that any function which is $O(100n)$ is also $O(n)$, and vice versa. Show that any function which is $O(n)$ is also $O(n^2)$, but show that the vice-versa is false. (While I do not require full blown proofs, these should be precise and convincing explanations)

6. Consider the following Python class

   (USE JUYPTER NOTEBOOK FOR THIS PROBLEM):

```
class Polynomial:
    def __init__(self, c, e):
        self.coeff = c
        self.exp   = e

    def evaluate(self, x):
        return self.coeff * (x ** self.exp)
```

   The class Polynomial is expected to take two positive integers (c and e) as input. It then represents the simple polynomial cx^e. The `evaluate` method returns the value of the polynomial at the given input. Write a Python function (must be called exactly `BigO`) which takes as input $p1, p2, m$, where $p1$ and $p2$ are of type Polynomial, and m is a positive integer. The function should return either a positive integer or `None`, as follows: If there is a least positive integer n such that $p1(x) \leq p2(x)$ for all integers from $n \leq x \leq m$, then n is returned; otherwise (if there is no such integer) `None` is returned.

   Example: Suppose p1 = Polynomial(3, 1) and p2 = Polynomial(1, 2), so p1 represents the polynomial $3x$ and p2 represents the polynomial x^2. Suppose we call `BigO`(p1, p2, 7). Notice that $p1(1) = 3 > 1 = p2(1)$ and $p1(2) = 6 > 4 = p2(2)$, however $p1(3) = 9 \leq 9 = p2(3)$, and for integers $x \geq 3$, $p1(x) \leq p2(x)$. Thus, $n = 3$ is the least positive integer such that $p1(x) \leq p2(x)$ for all integers from 3 to 7, so 3 is returned.

   Example: With the same p1, p2, and m, if we **reverse** p1 and p2, to call `BigO`(p2, p1, 7), then the function should return `None`. The reason for this is that $p2(x) > p1(x)$ when x is at least 4, so we can find no suitable n that works up till $m = 7$.

   The Point: This is a simple "Big O" checker: In the above examples, p1 is O(p2) but p2 is not O(p1).