INTRODUCTION TO COMPUTER SCIENCE I
Fall 2014
LAB 4
Finding things in lists

We now know how to make a *list*, so let's write some functions to search through them. We'll work on lists of *integers* only, just because they're easy to work with.

1 Writing search functions

In lab, I wrote the following `main()` function to begin a new module named `search.py`:

```
def main ():
   n = int(input('Enter the desired list length: '))
   l = makeRandomList(n)
   l.sort()
   v = random.randrange(1, 10000)
   p = linearSearch(l, v)
   q = linearSearch(l, v)
   if l[p] != l[q] or l[p] != v:
      print('Uh oh!')
```

Key observations about this function:

- Of course, `n` becomes the length the of the list that the program creates and then operates upon.
- Specifically, `makeRandomList(n)` must create and return a list of `n` randomly selected integers in the range of 1 to 10,000. **You must write this function.**
- The magic line, `l.sort()`, invokes a *dotted method* on the list `l`. These are functions that must be called, in this strange form (which we will discuss later), on some list. The dotted method modifies the list itself, and returns no value.
- `v` is a randomly selected value for which your functions will search within the randomly generated `l`.
- The function `linearSearch(l, v)` searches the list `l` for the value `v`. It does so by testing each position in the list in turn, testing to see if it's value matches `v`. If found, the function must return the index at which it was discovered; if `v` is not found, the function should return `None`. **You must write this function.**
- The function `binarySearch(l, v)` also searches the list `l` for the value `v`. It does so by repeatedly testing the element in the middle of the available range and then discarding half of that range depending on the comparison of the middle value and `v`. Again, this function returns the index at which the value is found; otherwise, it returns `None`. **You must write this function.**

Piece by piece, write and test these functions to be sure that they are working properly.

2 Measuring efficiency

Once you've written these functions and made them work properly, there is one more capability that you must add to your search function. Specifically, each of `linearSearch()` and `binarySearch` should keep its own counter of the number of *comparison operations* performed. That is, each time one of these functions compares `v` to some entry in its `l`, the function should increment a counter of the total number of such comparison operations. Before returning, this function must print this number of comparisons. **Do not confuse the *printing* of the number of comparison operations with the *returning* of the position at which the value is found.**

Create a plot: Once you've added the above capability to your search functions, run the program a number of times, providing varying values of `n`, and recording the number of operations each search function reports. Notice that if you run the program more than once with the name `n` value, you may get varying results! **Trying plotting the points that you record; what do you see?** More information will later be provided about what you need to submit for this part of the lab, but record and plot a decent set of points as soon as you are able. (Using something like *Excel*, or if you're up for it, *R*, is a good idea.)

3 Submitting your work

Go to the CS submission system to submit your **code**, namely the `search.py` module, for this lab. (Note that the plots will be submitted separately, next week.)

This assignment is due on Friday, Oct-24, 11:59 pm