Computer Science 112 - Fundamentals of Programming II
Lab Project 3
Due on Github 11:59pm Monday 3 October

In this lab, you will write some functions and then profile them for their run time performance. Your work should be in six files, named **counter.py**, **profile.py**, **tools.py**, **fibs.py**, and **sorts.py**, and **analysis.txt**.

1. Complete the function named `getRandomList` in **tools.py**. This function expects the size of the list as an argument and returns a list of unique numbers between 1 and this size, where the numbers are in random order. ***Hints***: use the `list` function to convert a range of numbers to a list, and then run the `random.shuffle` function on this list to randomize it. Do ***not*** write a loop in your function!

2. In the file **fibs.py**, complete the two versions of the Fibonacci function discussed in class. One is the straight recursive version and the other is a recursive version with a memorizing cache. Feel free to copy/paste/modify code from the lecture slides, but be aware that the minus sign in the lecture slides can turn into a long dash in your code, so you'll have to repair it by hand. You may ignore the counter argument for now. Test each function to verify correctness.

3. Now use the `counter` argument in each Fibonacci function. This counter should track the number of calls of the function. If the counter exists, it must be incremented on each call of the function. The best place to do this is on the first line of code within the function. *Hint*: Make sure to pass the counter object when you call the function recursively. Test/verify.

4. In the file **sorts.py**, complete the functions `selectionSort` and `quickSort` discussed in lecture and test to verify that they work correctly. You may ignore the two counter arguments in each function for now. To make the output easier to read, I found it useful to put

5. Now use the two counter arguments in your sort functions. One counter should count the number of **<** comparisons of data values, while the other counter should count the number of swaps. If a counter exists, you need to increment it. Test to verify correctness.

6. Now, run the two profiling functions in **profile.py** to profile each of your sort and Fibonacci functions. To profile a Fibonacci function, just pass the name of your function and an optional upper bound to the function `profileFib` when you call it. To profile a sort function, just pass its name and an optional number of iterations to the function `profileSort` when you call it. As usual when profiling code, the trick is to find a number of iterations that will give you prominent time differences but not take forever to run.

7. Write a short summary of your profiling experiments in a file **lab3report.pdf** file with your name on the top. ***Do not turn in a Microsoft Word or Word-like file!*** Be sure to point out any differences in the performance of your functions and account for them in terms of the complexity theory you have learned. Be precise, using big-O notation to characterize the run-time behavior. Vague statements like "It performs way better" or "It's slow as molasses!" will result in no credit!