Computer Science 112 - Fundamentals of Computer Science II

Lab Project 3

Due date: 5 PM, Monday, January 29

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**. Copies of the first five files are in **project3**.zip on Sakai.

1. Be sure that your name is at the top of each file that you turn in. Any file without your name in it will cause the instructor to give you a 0 on this week's project.

2. 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!

3. 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 memoizing cache. You may ignore the counter argument for now. Test each function to verify correctness.

4. 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. Test/verify.

5. 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.

6. 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.

7. 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.

8. Write a short summary of your profiling experiments in a text file. This file must have a **.txt** extension (Mac users can use TextEdit, PC users can use WordPad or NotePad, Linux users can use gEdit). ***Do not turn in a Microsoft Word or Linux 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!