

ME 499/599: Computer Programming for Mechanical Systems

Winter 2016

Purpose

The goal of this lab is to give you some more experience writing functions in different ways, and to think more about how the running time of algorithms scales with the size of their inputs.

Preparation

Look over any notes you took from the lecture on sorting lists of numbers.

Assignment

1. This lab is going to be all about sorting lists of numbers. First, implement the [bubblesort](#) algorithm. Your function should take a list of numbers as it's single argument and it should return a sorted version of that list. The original list should remain the same. Do the sort 10 times and return the average time taken.
2. Let's make sure that your bubblesort code is correct. Write a function that checks that the function passed to it as an argument correctly sorts a list. You can assume that the built-in sort function works correctly, and compare the results of using the function passed in to those obtained by running sort. Or, you can write your own function from scratch.
3. Graph the performance of bubblesort for lists of random numbers of length 100, 200, ... 2000. Remember to give your graph a title and label the axes. Also plot the running time of the built-in function sorted as a comparison.
4. Graph the performance of the built-in sorted function on the same set of data. What do you notice about it, compared to bubblesort?
5. Quicksort is one of the most famous sorting algorithms. Implement it, time it, and graph it.

6. **Graduate Students and Undergraduate Extra Credit:** Implement another sorting function, either `insertion_sort` OR `mergesort`. Implement it, time it, and graph it.

Grading

There are 11 points available for 499 students, with 3 possible extra credit points (2 for the sort, 1 for the graph). There are 14 points available for 599 students.

You should submit your file through the [TEACH](#) as usual.

1. Code does not run: Zero points for the whole lab. We're very serious about this. Your code must run without errors in order to get any points for this assignment.
2. Code not submitted to TEACH: Zero points for the whole lab.
3. 2 points for a working Bubblesort implementation. [2 points total]
4. 2 points for a working correctness function implementation. [2 points total]
5. 3 points for a working quicksort implementation. [3 points total]
6. **Graduate Students and Undergraduate Extra Credit:** 2 points for another working sort implementation. [2 points total]
7. 1 point for a correctly-constructed graph. 1 point each for the timing curve for each of the functions (Bubblesort, quicksort, `sorted`, another optional sort). [3 (+1 extra credit) points total for 499, 4 points total for 599]

Thoughts

1. Yes, all of the sorting algorithms in this lab are easily available on the web. Yes, we expect you to look them up. Yes, we expect that some of you will use some of the things that you find there. Don't use anything you don't understand, and don't use anything that's not yours without telling us about it. We *will* check, and will penalize such things harshly. If you're in doubt, **ask**.
2. Quicksort is defined recursively. Given a list, `l`, its call to `quicksort` should return

```
quicksort(all elements less than l[0]) +  
[all elements equal to l[0]] +  
quicksort(all elements greater than l[0])
```

This is a bit tricky to get your head around. If you're having trouble with it, ask in class, and we'll talk about it more. You might find the `filter` function useful for implementing this sort.

3. Prove your sorts work by running them through your testing code.
4. Yes, we're asking you to implement algorithms that you don't know. Yes, this is fair. A lot of your programming career will involve implementing other people's algorithms. To do this, you're going to have to figure out how they work, find example code to work from, and adapt it to your own purposes. Implementations of all of these algorithms are widely available on the web, and the collaboration policy for the class allows you to use these, as long as you end up understanding them and you **disclose your sources**. This is what actual programmers do every day.

If you want to talk about the algorithms, ask the instructor and we'll go into them a bit in class. Ask the TAs. Use Google. An important part of learning a programming language is to learn how to glean help from the Web, and this is what we're trying to give you a feel for in this lab.

The Rules

Everything you do for this lab should be your own work. Don't copy from other people in the class. Remember the attribution policy when looking at the web for information.

Page written by [Bill Smart](#).