**Lab 1: Benchmarking Insertion Sort**

**CS 3851 Algorithms**

**Learning Outcomes**

* Implement sorting algorithms such as heapsort and quicksort
* Apply asymptotic time complexity analysis to choose among competing algorithms

**Overview**

You are going to implement the insertion sort algorithm. You will then benchmark the run time of insertion sort under the best, worst, and average case scenarios. You will plot the run times and interpret the plot in relation to the asymptotic run time.

Insertion sort is described in Chapter 2 of *Introduction to Algorithm*s, 3rd ed. by Cormen, Leiserson, Rivest, and Stein.

**Instructions**

**Installing the Anaconda Python Distribution**

We will be using Python this quarter for all of our labs. The easiest way to install a full Python distribution system is with Anaconda:

<https://www.anaconda.com/distribution/>

Please install the 64-bit version of Python 3.7. This can take some time (especially if everyone is doing it once), so please try to install this before coming to lab.

**Implement Insertion Sort**

You have been provided with two .py files (sorting\_stub.py, test\_sorting.py). The file sorting\_stub.py contains the signature for a sort function. Your job is to finish that implementation. The file test\_sorting.py contains unit tests designed to help you check the correctness of your implementation.

1. Rename sorting\_stub.py to sorting.py.

2. Implement the following methods:

* void insertion\_sort(lst) – The function takes a Python list and sorts it in place.

You can test your code by running:

$ python test\_sorting.py

**Benchmark Insertion Sort**

Next, you are going to benchmark insertion sort. Although we can derive the run time formally and express the run time in big-oh notation, benchmarking the algorithm can help us develop intuition. We are going to benchmark insertion sort on lists of numbers. We will evaluate three cases: list is randomly permuted, list is already sorted, and list is already sorted in reverse order.

Use the following skeleton for your benchmarks:

times = []

for i in range(n\_trials):

# DO LIST SETUP HERE

start = time.clock()

insertion\_sort(lst)

end = time.clock()

elapsed = end - start

times.append(elapsed)

Note that time.clock() returns in the time in seconds. You can use the mean() function from Numpy to calculate the average run times.

3. Create a Jupyter notebook named lastname\_lab01. The notebook should have a title, your name, and an introduction.

4. In the notebook, import the sorting, matplotlib, Numpy, random, and time modules.

5. Benchmark insertion sort with lists of 100; 1,000; 10,000; and 100,000 numbers. Create the lists from random numbers using random.random(). Use random.shuffle() to generate a random permutation of the list and the list.sort() method to sort the list in order and reverse order (using the reverse keyword). Run each benchmark 10 times (trials).

6. Use Matplotlib to plot the list size versus the run time using a line plot. There should be one line for each scenario (randomly permuted, sorted, and reverse sorted). Label each line and create a legend. You may need to take the log10 of the list sizes to get a nice graph.

**Analyze Insertion Sort Run Time**

Big-oh notation (e.g., O(n)) gives an upper-bound on the run time of an algorithm relative to the size of its input. In this case, we will be sorting lists of numbers, and n refers to the number of items in the list. The run time is specified in terms of the number of instructions executed.

Algorithms are evaluated in three cases: the best, worst, and average cases. Each of the cases is associated with a property of the input data.

7. Answer the following questions in your notebook:

1. Were there any differences between the run times of the three cases? What inputs gave the best-, worst-, and average-case runs time?

2. Why do you think the different inputs caused different run times? (Hint: focus on the inner loop of insertion sort.)

3. Look up the best-, worst-, and average-case run time of insertion sort in big-oh notation and provide those. Are the formally determined run times consistent with your benchmarks?

I will be will looking for the following:

* That the tests pass
* An introduction (including your own statement of the problem) and a written summary of your results at the top of the notebook in Markdown. Make sure to put your name at the top of the notebook.
* That your line plots look reasonable.
* Reasonable attempts to answers the questions.

**Submission Instructions**

Save the notebook as a HTML file named lastname\_lab01.html. Put your HTML file and sorting.py files into a zip file. Upload the zip file to Blackboard.

**Rubric**

|  |  |
| --- | --- |
| Followed submission instructions | 10% |
| Insertion sort implementation passes tests | 30% |
| Correctly benchmarked the three cases | 30% |
| Plot of the run times | 10% |
| Answers to questions | 20% |