Testing the speed of the merge sort and timsort algorithms

Homework #3

By

John Robertson

CS 303 Algorithms and Data Structures

September 12, 2019

1. **Problem Specification**

This assignment is supposed to measure the speed of merge sort (which is fast) and combine the algorithm with insertion sort (the combination of which is even faster).

2. **Program Design**

This program has both the implementation and the driver in the same file. I added an insertion sort wrapper for the timsort implementation in order to use the function as a mutator.

The following steps were required to develop this program:

1. Implement merge sort as the merge\_sort() function.
2. Do the same with tim\_sort().
3. Import the files in a loop in order to run merge\_sort() on their contents.
4. Print how long doing that takes.
5. Repeat c. and d. on tim\_sort()

3. **Testing Plan**

I at first tried the bigger test I tried last week (1000 random arrays of 400 elements each) and dialed back to a random array generated once. After I got merge\_sort to work for these, I used the same test cases of last week.

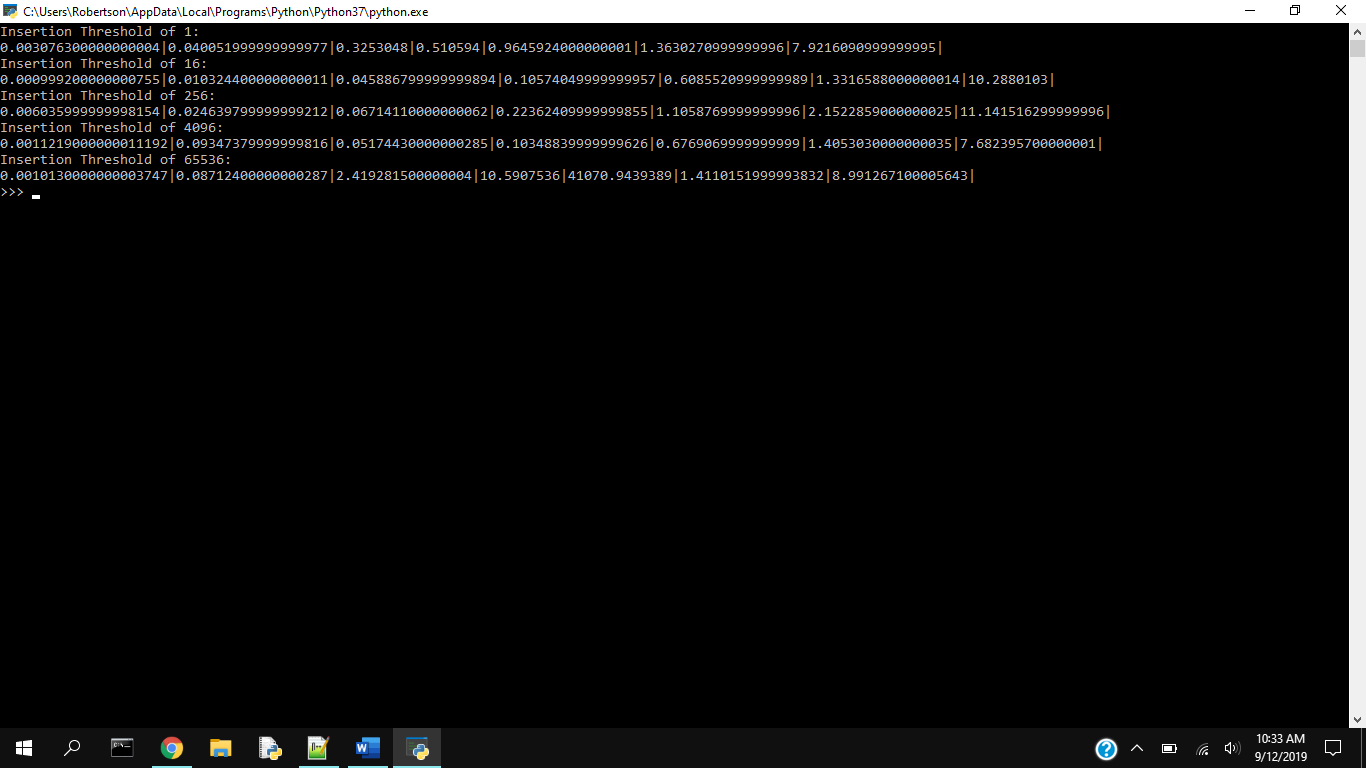
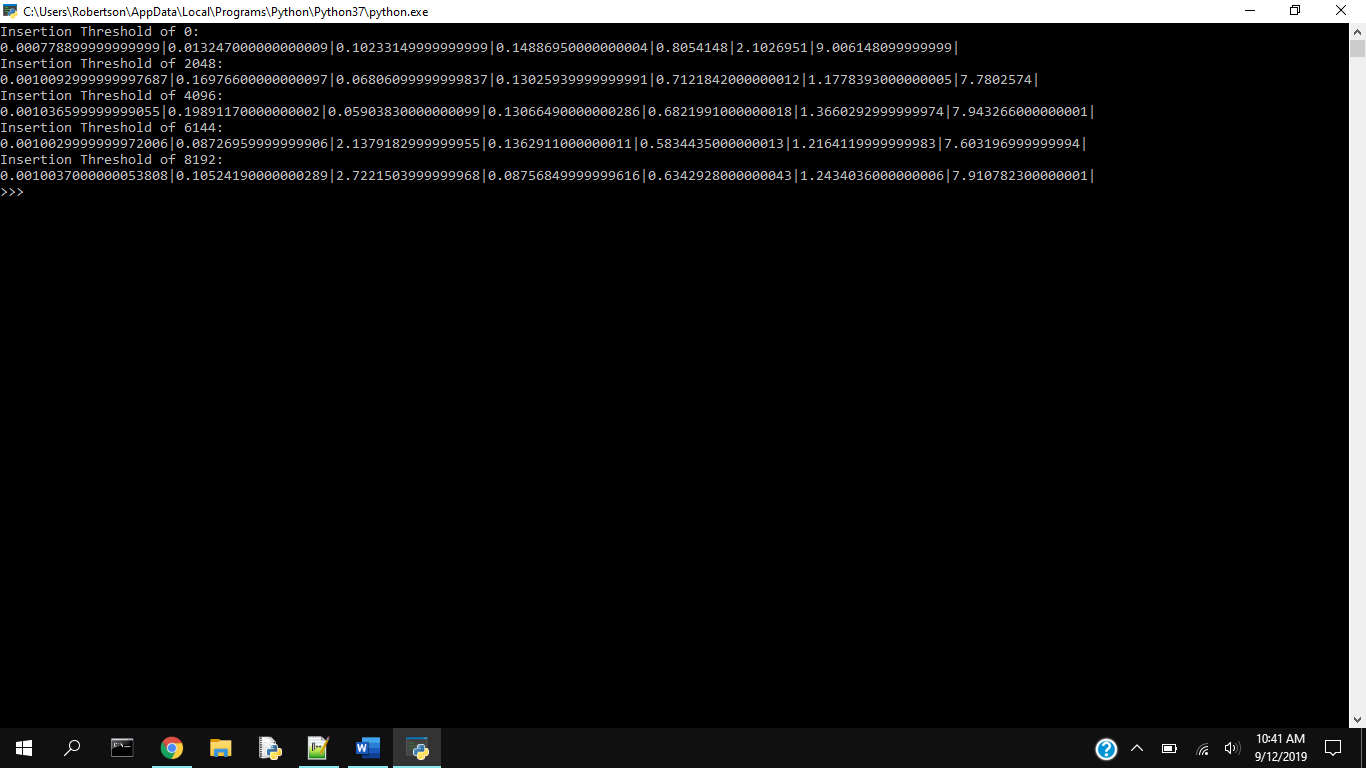
4. **Test Cases**

I used the same initial test cases as last week: zero elements, one, reverse sorted, and random elements. I also left a fixed array in the file for debugging purposes.

This week, I put a spin on the input files by going over them more than once with different values for the threshold value on when to use insertion sort rather than merge. I spaced them out to see when using insertion sort provides an advantage.

5. **Analysis and Conclusions**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Number of inputs | 100 | 1000 | 5000 | 10000 | 50000 | 100000 | 500000 |
| Insertion Sort | 0.0024714000000000125 | 0.2704601 | 0.7821910999999999 | 27.0296733 | 400.4907278 (7 minutes) | 1343.2281662 (23 minutes) | 36636.953898700005 (10 hours) |
| Pure Merge Sort | 0.0007162 | 0.0180035 | 0.0581905 | 0.1089698 | 0.9279293 | 1.5957571 | 8.6510837 |
| Python .sort() | 3.079999e-05 | 0.0003054 | 0.0019508 | 0.0044819 | 0.02433 | 0.05399959 | 0.3521366 |
| Merge Sort + Insertion Sort | See below images |  |  |  |  |  |  |



The fastest of these for sorting the chart of 500000 elements, with the threshold of 6144, is plotted below. Note that it took 2 seconds for 5000 elements but less than a second for twice that; threshold 2048, while faster than 6144 but still beating 4096, doesn't have this jump.

Obviously, using insertion sort for larger values provides the same disadvantage as simply using insertion sort; the file with 50000 elements took eleven minutes to sort with the 65536 threshold. A threshold of 1 is the same thing as only using merge sort. It appeared at first that a value around threshold 4096 had a performance gain; values near 4096 also are faster, even if only marginally so, than pure merge sort.

6. **References**

The following references were used:

<https://stackoverflow.com/questions/1614236/in-python-how-do-i-convert-all-of-the-items-in-a-list-to-floats>

Dr. Unan's Lab3.pdf for the pseudocode for the algorithm

pythontutor.com for the seed value for the array in the first test