Implementations of bubble sort and selection sort and reverses of previous sorting algorithm implementations
Homework #7
By
John Robertson
CS 303 Algorithms and Data Structures
October 9, 2019

0. **Warning**

Just like the last assignment with quicksort in it, I had to set the recursion limit to its maximum. If your computer is 32-bit and not 64-bit, please change the *setrecursionlimit(2**31 - 1)* at the top of *reverseLib.py*.

1. **Problem Specification**

This problem requires us to implement two more sorting algorithms and to write reverses of all of our previous sorting functions.

2. **Program Design**

This program has four files: a library file with reverses of my past and current sorting function, a file with the new sorting functions, and two drivers, one for quicksort and another for the rest.

The following steps were required to develop this program:

- Implement selection and bubble sorts
- Debug the above until they worked
- Make a reverse version of all the available sorting algorithms
- Debug the above until they worked
- Write two drivers
- Debug the drivers until they worked

Similar to last week, I made it an option to run the code for testing the ascending selection and bubble sorts on large inputs.

3. **Testing Plan**

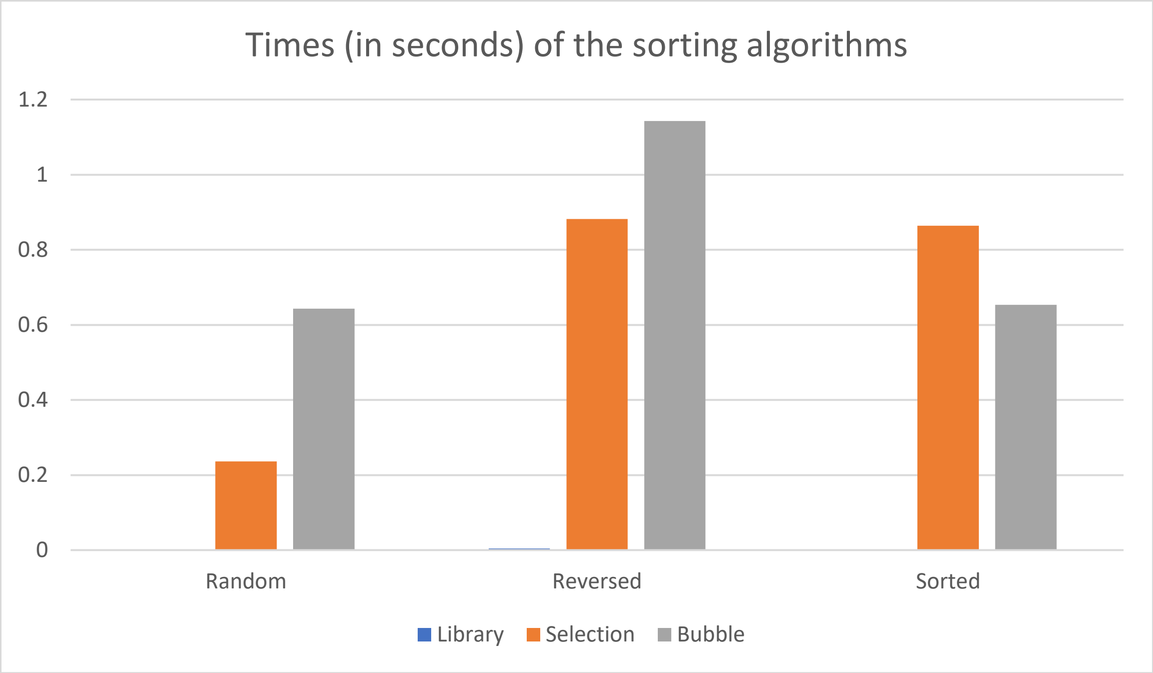
The first and most important thing was to make sure the implementations of bubble and selection sort were correct. I tested the results on a hard-coded array, an array of randomly selected elements, a larger such array, an empty array, and an already sorted array (a range). The functions compared correctly to the library sort each time.

4. **Test Cases**

bubbleselect.py tests the bubble and selection sorts on the sorted, reversed sorted, and random input files as well as the above-mentioned hand-coded tests. *quicksortdriver.py* and *driver.py* test all of the reversed sorted algorithms for correctness by comparing them to the standard library's *.sorted(reverse=True)* function, and then test them on all of the input files 100 through 50000 as well as the random, reversed sorted, and sorted input files.

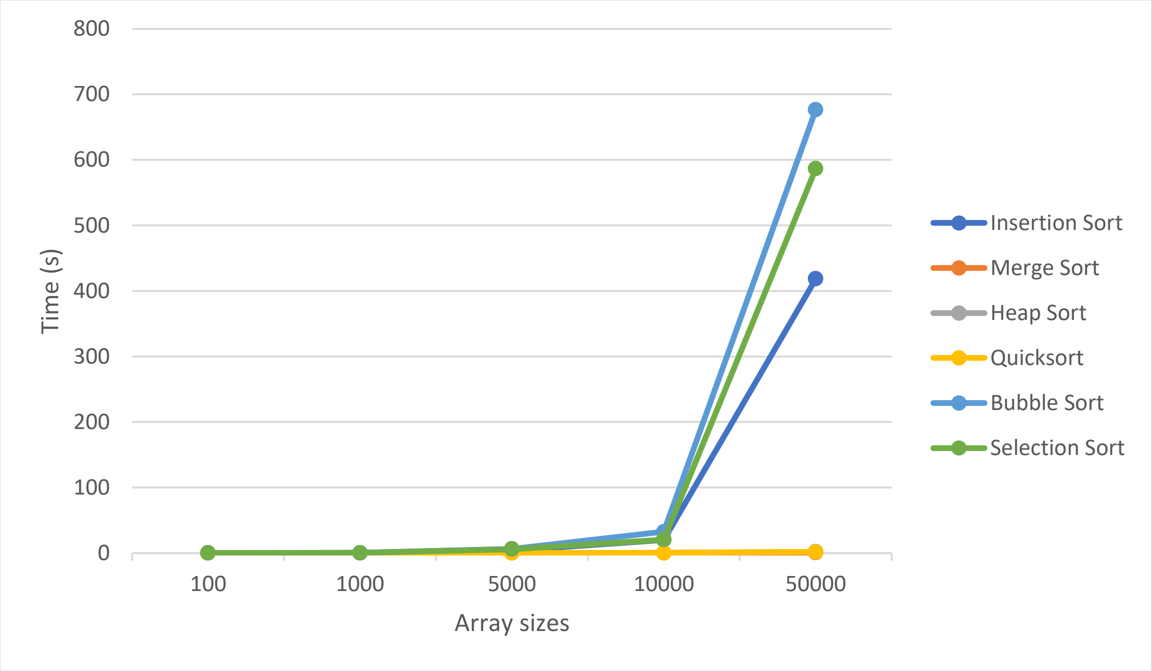
5. Analysis and Conclusions

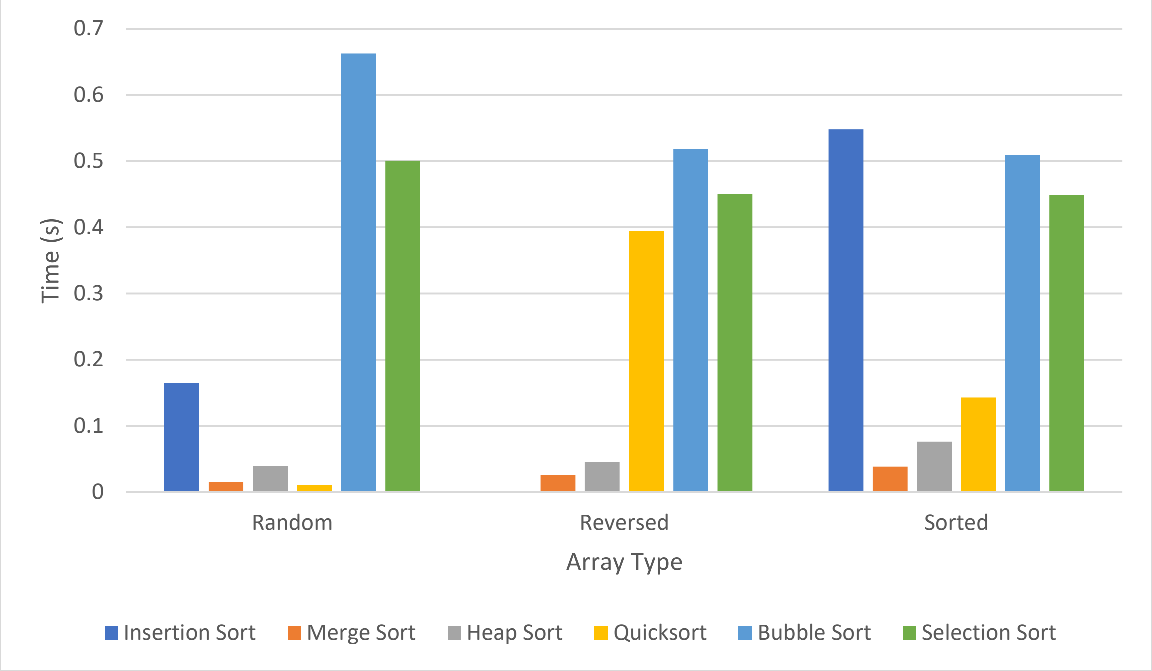
	Random	**Reversed Sorted**	**Sorted**
Standard Library	**0.00035289**	**0.00038889**	**0.00003379**
Selection sort	**0.23593789**	**0.88193589**	**0.8644507**
Bubble sort	**0.6427179**	**1.1426369**	**0.65736639**



The standard library sort took so few seconds that it is barely visible on the chart.

Inputs:	*100*	*1000*	*5000*	*10000*	*50000*	*Random*	*Reversed*	*Sorted*
insertion	0.00135889	0.23454589	3.82963519	20.0437384	418.4780366	0.1649743	0.0010948	0.5479487
merge	0.00108139	0.02344659	0.0377659	0.15177879	0.62497259	0.0149234	0.02551339	0.03824
heap	0.00149469	0.0383259	0.09224009	0.3729363	2.10281139	0.0394404	0.0450316	0.0761964
quick	0.0008268	0.0136821	0.065915	0.3524266	1.8286396	0.0108812	0.3940207	0.1427822
bubble	0.00179539	0.37946429	6.3864255	32.6196	676.69403239	0.6626786	0.51790449	0.50930779
selection	0.0019229	0.3904854	6.06901179	20.5217	586.7567696	0.5008608	0.450316	0.44851979





Because selection and bubble sort both involve two nested loops, they are both O(n^2), making them in the same rank as insertion sort in terms of worst-case speed. In fact, the both of them do worse than insertion sort. Concerning the efficiency of the reversed sorting algorithms, they do not appear to have been significantly impacted by the change, which shouldn't be surprising, because at most three or four instructions were changed, and none of the instructions in question were anything that would impact efficiency.

6. **References**

My previous sorting algorithms for this course
https://visualgo.net/en/sorting?slide=1