Testing the speeds of quicksort algorithm variants
Homework #5

By
John Robertson

CS 303 Algorithms and Data Structures
September 24, 2019

0. **Warning**

I reached the recursion limit for sorting 8192 elements using quicksort here, so I put at the top of the quick_sort.py file "sys.setrecursionlimit(2**31 – 1)", the maximum recursion depth for my machine. If your machine somehow has a smaller C long than mine, please manually change the 31 to a 15.

1. **Problem Specification**

This project implements two version of quicksort, vanilla and with median-of-three, and compares it to prior sorting algorithm implementations.

2. **Program Design**

This program has a file with the quicksort function, a library file for the other sorting functions, and a driver file to test it. The quicksort functions work by choosing the last element as the pivot, sorting subarrays on either side of the index, and merging the sorted sub-arrays. Because they are on either side, value-wise, of the pivot, there is no need to sort again when merging, because they are already sorted relative to the pivot. Median-of-three quicksort does the same thing, but it chooses its pivot by choosing the middle value of the first, last, and middle indices.

The following steps were required to develop this program:

- Implement quicksort
- Implement quicksort with median-of-three
- Create test cases for the two
- Create a library file for my functions for easier code reuse now and later.
- Create a driver similar to previous drivers.

3. **Testing Plan**

I tested the bare implementation first on an arbitrary array to see if it worked. After being satisfied that it worked, I implemented the median-of-three quicksort and tested it in the same way.

4. **Test Cases**

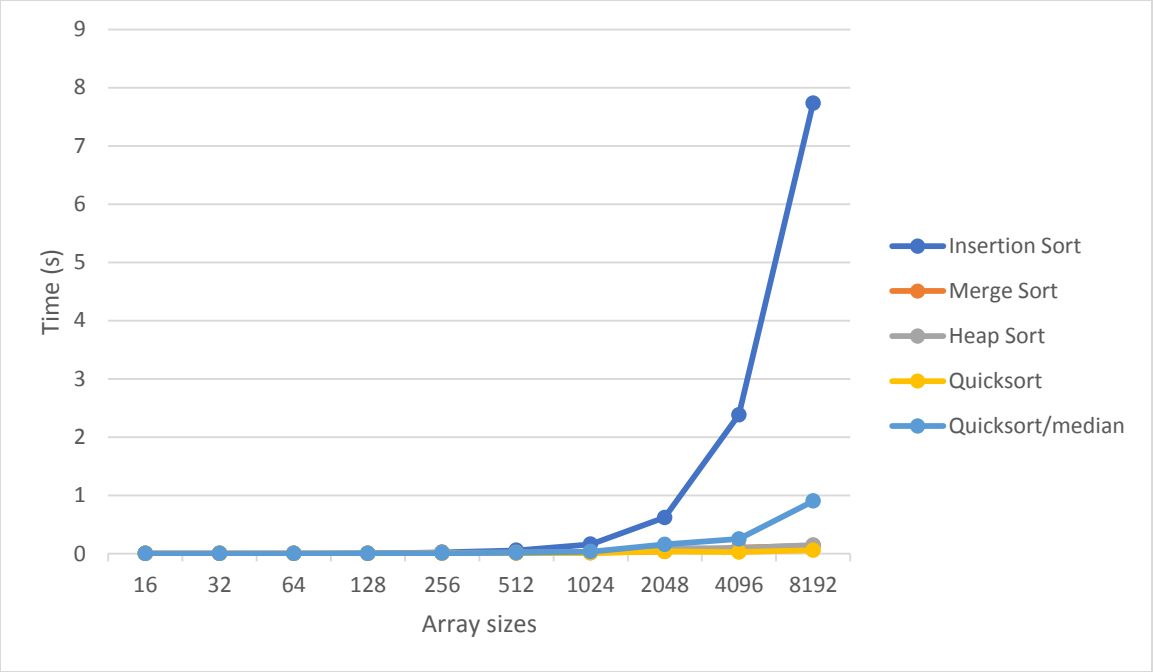
I tested both quicksort and quicksort with median-of-three by comparing them with the standard library sort on the following: a small amount of random elements, a large amount, zero elements, and an already sorted array.

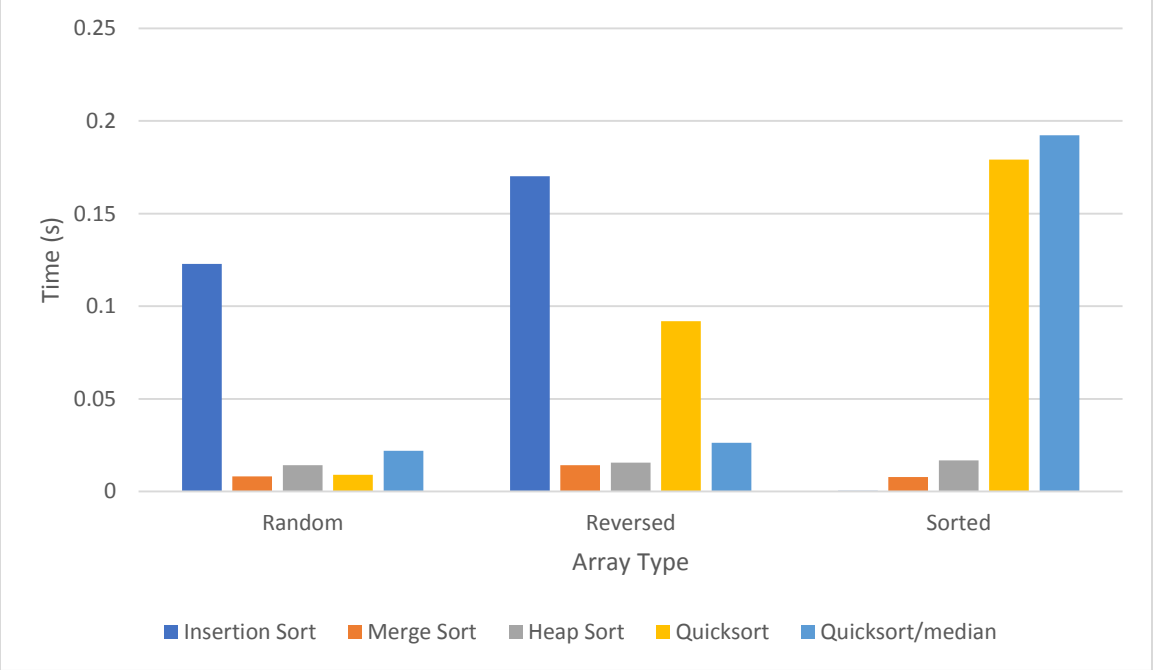
The rest of the cases are from the input files.

5. **Analysis and Conclusions**

Inputs:	*16*	*32*	*64*	*128*	*256*	*512*
quicksort	0.000115	0.00014259	0.00299	0.00178949	0.00243	0.00481
mediansort	0.00013	0.00043779	0.000764	0.0030167	0.0045	0.011646

1024	*2048*	*4096*	*8192*	*Random*	*Reversed*	*Sorted*
0.0054759	0.01141119	0.0419399	0.0805099	0.0131575	0.9610869	0.2687401
0.03083349	0.07895489	0.249707	0.95868639	0.0264537	0.0234769	0.35021639





Immediate impressions: quicksort with median-of-three doesn't outperform vanilla in any of the cases except for one: when the array is sorted in reverse order. Both quicksorts severely underperform

the prior sorting algorithms for an already sorted array. Vanilla competes with merge and heap for speed; vanilla quicksort is marginally faster than them.

6. **References**

No references were used other than Dr. Unan's slides and the problem pdf.