Creating a novel sorting algorithm

Homework #6

By

John Robertson

CS 303 Algorithms and Data Structures

October 4, 2019

0. **Requirements**

You need to have the "powers of 2" input files from the last assignment, as well as 'Input_Sorted.txt', in the same directory as driver.py.

1. **Problem Specification**

The intent of this project is to get the student to implement their own algorithm without the immediate aid of pseudocode. To this end, the program must have functions that 1) sort by primitive-recursively finding minima and maxima of an array and 2) stably sort a transaction log so that one level of order is changed but another preserved.

2. **Program Design**

This program has four original files: a library file with my past sorting functions, a file with the new sorting functions, a driver, and a handmade text file for testing.

The following steps were required to develop this program:

- Implement the novel sorting algorithm from the problem description
- Debug the former-mentioned until it works
- Implement a function that sorts the transactions
- Test the above two functions on larger inputs
- Write the driver

For the speed (and the sake) of the one grading, I made it an option to run the code for testing the novel sorting function only once; the same was done for testing on many inputs.

3. **Testing Plan**

I tested the novel sorting function on an array of 256 random elements to see if it got the same result as the library's sort() function. I did the same for an arbitrary but bounded number of times; it tested right every time. I also tested for a reverse sorted array and an empty array. For transaction sorting function, I tested it using the default. After that, I created my own similar file **with the times not starting off sorted in addition to the city names** in order to push myself to improve my transaction sort implementation (which was just a call to my previously written heap_sort()).

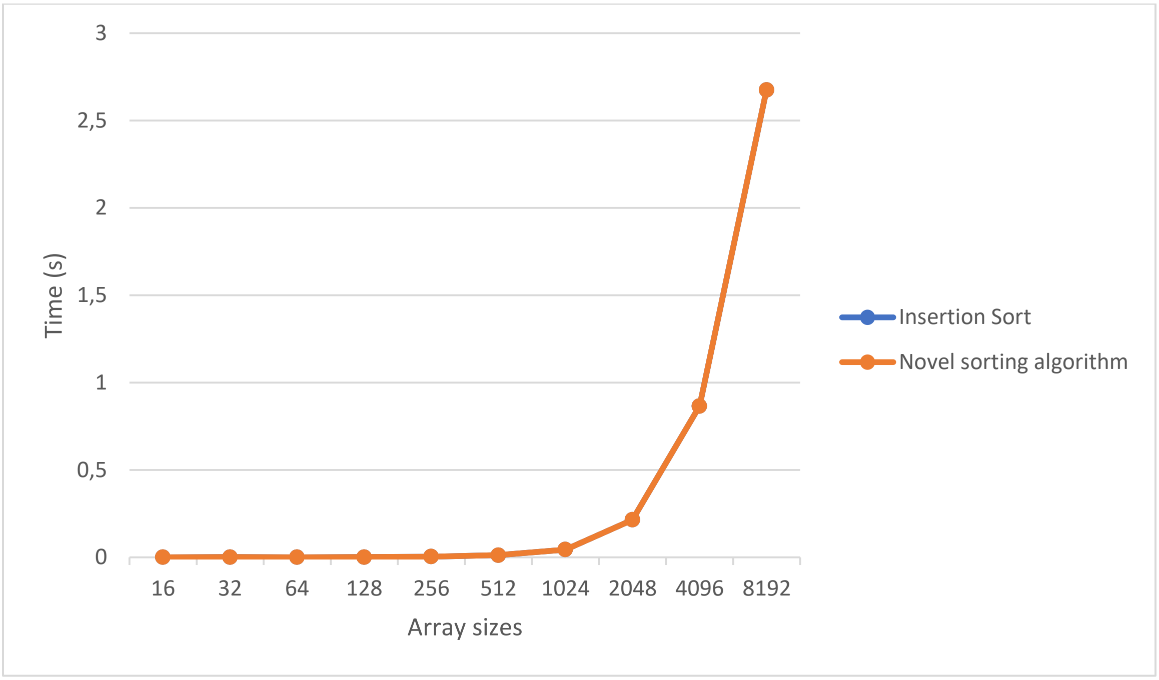
4. **Test Cases**

The test cases of the novel sorting algorithm are an empty array, a reverse sorted array, and an array populated with random elements. The single give input file was used for the transaction, but I also made a file with the times not already sorted to solve the bonus problem.

5. **Analysis and Conclusions**

	16	*32*	*64*	*128*	*256*
Novel Sorting	0.000075349	0.0015069	0.00037259	0.0011314	0.0037447
Insertion Sort	0.00004269	8.349	0.002709	0.00112269	0.004463

512	*1024*	*2048*	*4096*	*8192*	*Already Sorted*
0.01266359	0.0439544	0.21495	0.8660291	2.67459	0.046154
0.0203951	0.08910749	0.81080749	1.391223	5.62708329	0.00047649



Note: insertion sort is graphed, but its line is so similar to the other sorting algorithm's graph that it is obscured.

It is apparent that just a small change in program specifications can make big differences in what one must implement. Seven lines were added to my transaction sort, including a second call to heap_sort(), just by changing the assumption that the times at the ends of the lines are already sorted. The implementation of the novel transaction sort showed me how to abstract one level away from pseudocode by solving an XY question, that is, the command to "do X (sort) using Y (minima and maxima)." As can be shown above, this novel algorithm performs with similar efficiency to my insertion sort implementation.

6. **References**

Dr. Unan's slides and lectures, especially his comment on using negative and positive infinity to find the minima and maxima of lists

My previous sorting algorithms for this course