# Analysis of basic sorting algorithms

**By Carter Brady**

**CSC 2400-004 | Assignment II**

Due 10/27/2023

Instructor: Professor Radian

# Table 1:

| Algorithm | Array type | size of n | time to complete in seconds | time factor increase | size factor increase |
|---|---|---|---|---|---|
| Shell sort | random | 90 | 0.00 | | |
| Shell sort | random | 900 | 0.02 | | 10.00 |
| Shell sort | random | 5000 | 0.01 | | 5.56 |
| Shell sort | random | 40000.00 | 0.05 | 5 | 8.00 |
| Shell sort | random | 100000.00 | 0.27 | 5.4 | 2.50 |
| Insertion sort | random | 90.00 | 0.00 | | |
| Insertion sort | random | 900.00 | 0.00 | | 10.00 |
| Insertion sort | random | 5000.00 | 0.10 | | 5.56 |
| Insertion sort | random | 40000.00 | 6.46 | 64.6 | 8.00 |
| Insertion sort | random | 100000.00 | 108.28 | 16.76160991 | 2.50 |
| Selection sort | random | 90.00 | 0.00 | | |
| Selection sort | random | 900.00 | 0.02 | | 10.00 |
| Selection sort | random | 5000.00 | 0.17 | 8.5 | 5.56 |
| Selection sort | random | 40000.00 | 34.42 | 202.4705882 | 8.00 |
| Selection sort | random | 100000.00 | 215.38 | 6.257408483 | 2.50 |
| Bubble sort | random | 90.00 | 0.00 | | |
| Bubble sort | random | 900.00 | 0.02 | | 10.00 |
| Bubble sort | random | 5000.00 | 0.22 | 11 | 5.56 |
| Bubble sort | random | 40000.00 | 73.08 | 332.1818182 | 8.00 |
| Bubble sort | random | 100000.00 | 497.12 | 6.80240832 | 2.50 |
| Shell sort | increasing | 90 | 0.00 | | |
| Shell sort | increasing | 900 | 0.00 | | 10.00 |
| Shell sort | increasing | 5000 | 0.00 | | 5.56 |
| Shell sort | increasing | 40000.00 | 0.01 | | 8.00 |
| Shell sort | increasing | 100000.00 | 0.04 | 4 | 2.50 |
| Insertion sort | increasing | 90.00 | 0.00 | | |
| Insertion sort | increasing | 900.00 | 0.00 | | 10.00 |
| Insertion sort | increasing | 5000.00 | 0.00 | | 5.56 |

| | | | | | |
|---|---|---|---|---|---|
| Insertion sort | increasing | 40000.00 | 0.00 | | 8.00 |
| Insertion sort | increasing | 100000.00 | 0.01 | | 2.50 |
| Selection sort | increasing | 90.00 | 0.00 | | |
| Selection sort | increasing | 900.00 | 0.00 | | 10.00 |
| Selection sort | increasing | 5000.00 | 0.17 | | 5.56 |
| Selection sort | increasing | 40000.00 | 34.31 | 201.8235294 | 8.00 |
| Selection sort | increasing | 100000.00 | 217.08 | 6.327018362 | 2.50 |
| Bubble sort | increasing | 90.00 | 0.00 | | |
| Bubble sort | increasing | 900.00 | 0.00 | | 10.00 |
| Bubble sort | increasing | 5000.00 | 0.16 | | 5.56 |
| Bubble sort | increasing | 40000.00 | 31.54 | 197.125 | 8.00 |
| Bubble sort | increasing | 100000.00 | 197.76 | 6.270133164 | 2.50 |
| Shell sort | decreasing | 90 | 0.00 | | |
| Shell sort | decreasing | 900 | 0.00 | | 10.00 |
| Shell sort | decreasing | 5000 | 0.00 | | 5.56 |
| Shell sort | decreasing | 40000.00 | 0.02 | | 8.00 |
| Shell sort | decreasing | 100000.00 | 0.08 | 4 | 2.50 |
| Insertion sort | decreasing | 90.00 | 0.00 | | |
| Insertion sort | decreasing | 900.00 | 0.00 | | 10.00 |
| Insertion sort | decreasing | 5000.00 | 0.17 | | 5.56 |
| Insertion sort | decreasing | 40000.00 | 36.25 | 213.2352941 | 8.00 |
| Insertion sort | decreasing | 100000.00 | 226.88 | 6.258758621 | 2.50 |
| Selection sort | decreasing | 90.00 | 0.00 | | |
| Selection sort | decreasing | 900.00 | 0.00 | | 10.00 |
| Selection sort | decreasing | 5000.00 | 0.14 | | 5.56 |
| Selection sort | decreasing | 40000.00 | 31.08 | 222 | 8.00 |
| Selection sort | decreasing | 100000.00 | 194.76 | 6.266409266 | 2.50 |
| Bubble sort | decreasing | 90.00 | 0.00 | | |
| Bubble sort | decreasing | 900.00 | 0.01 | | 10.00 |

| | | | | | |
|---|---|---|---|---|---|
| Bubble sort | decreasing | 5000.00 | 0.27 | 27 | 5.56 |
| Bubble sort | decreasing | 40000.00 | 54.41 | 201.5185185 | 8.00 |
| Bubble sort | decreasing | 100000.00 | 340.96 | 6.26649513 | 2.50 |

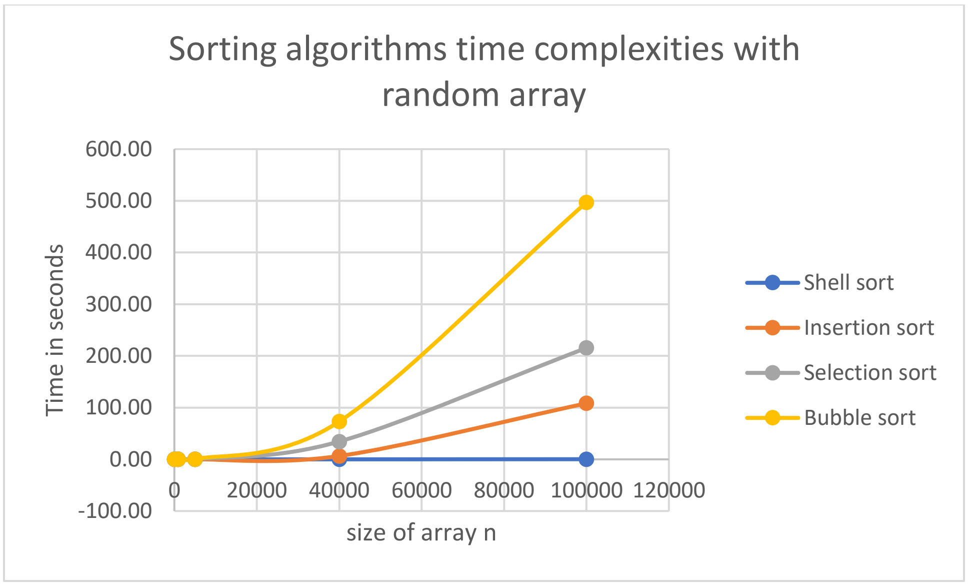
## The empirical time complexity for each is:

**Shell sort: n log n**

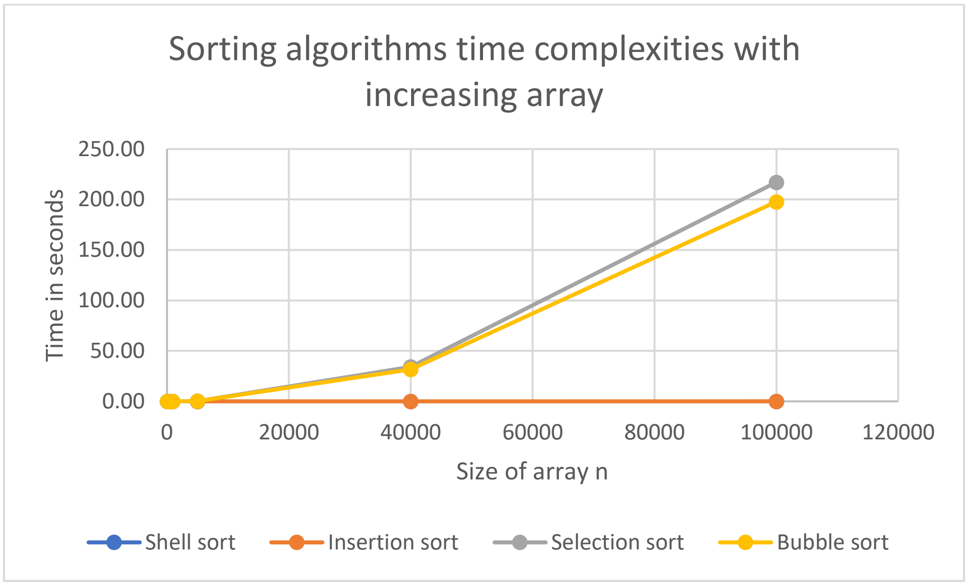
**Insertion sort: $n^2$**

**Selection sort: $n^2$**

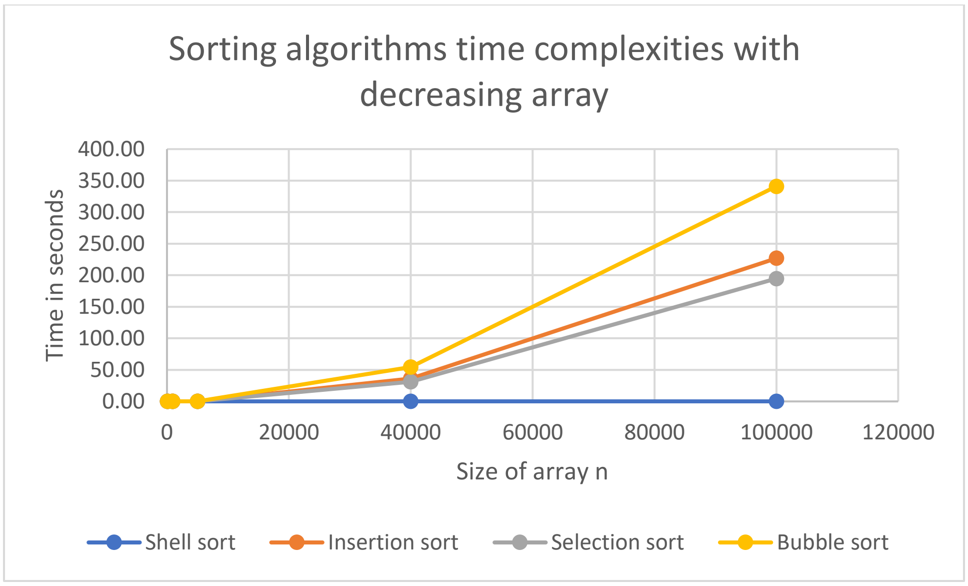
**Bubble sort: $n^2$**



**Graph 1**



**Graph 2**


Sorting algorithms time complexities with decreasing array
Time in seconds
400.00
350.00
300.00
250.00
200.00
150.00
100.00
50.00
0.00
0
20000
40000
60000
80000
100000
120000
Size of array n
Shell sort
Insertion sort
Selection sort
Bubble sort

**Graph 3**

The four sorting algorithms being analyzed for this assignment are shell sort, insertion sort, selection sort, and bubble sort. While the mathematical formula for the complexity of each algorithm is already known it is important to experiment and prove the mathematical formula with empirical evidence. The way this evidence will be gathered is through the generation of 15 different arrays of varying types. 5 of the arrays will be completely random, 5 of them will be randomly increasing, and 5 of them will be decreasing. These arrays will be copied to temporary arrays so that the same 15 arrays may be sent to each sorting algorithm in each trial. 3 trials will be conducted at each of 5 orders of magnitude for size of the array. These results are tabulated and provided in table 1. Each algorithm will be compared to their mathematical complexity by observing the change in time complexity and size.

The shell sort algorithm is a variation on insertion sort that breaks the array down into smaller sub arrays before performing insertion sort. This algorithm, while simple, is much more effective than the other sorting algorithms presented in this report. This algorithm is upper bounded by all other algorithms presented in this report. The average mathematical time complexity of shell sort is n log n which is very efficient. When comparing this to the results provided in table one you can see that the observed time complexity is roughly n log n. While it is hard to prove that it exactly fits that with only a few trials and a few different sizes with hardware that can vary run to run you can still tell that when comparing change in n and the time complexity that it fits the average mathematical time complexity. For example, when looking at the factor increase at n = 100000 you can see that the time complexity grew by 2.78 times and n grew by 2.5 times. This shows that the relationship is slightly above linear but not by much which is what n log n is. It is also worth noting when looking at graph 1 provided above that the shell sort has a flat looking line while the rest of the algorithms have some slope up. This is a great showing of why time difference in time complexity is so important because it shows how the different order of growth leads all of the rest of the algorithms to shoot up after at a certain point while shell sort still grows at a steady rate. Breaking the problem down into smaller pieces allows shell sort to operate with an efficiency that far surpasses what just insertion sort can do.

Insertion sort will compare an item in an array to the item to the left of it then it will swap the two items if the item to the left is greater than item being compared. This algorithm gets the job done however it is not nearly as efficient as shell sort. Its average mathematical complexity is $n^2$ which is not very efficient and as the size of n gets bigger its shortcomings become very apparent. This algorithm is lower bounded by shell sort and tight bounded by both selection and bubble sort. While it is tight bounded by those two algorithms when testing it is noticeable that insertion sort ran slightly faster than the other two while still increasing on the same scale. The empirical complexity that was tested and recorded in table one matches very well with the average complexity of $n^2$. This can be seen by looking at n= 40000 for insertion sort and comparing the time factor increase to the size factor increase. The size factor increased by 8 and the time factor increased by 63 which is nearly 64 which would be exactly $n^2$. The rest of the data from the table also closely match this finding. When working with small sizes of n insertion sort is alright to use but when dealing with a lot of data it becomes inefficient very quickly.

Selection sort goes through an array and finds the minimum value then swaps that with the current index. This process is repeated with a shrinking size of what the algorithm must compare. While selection sort does work and is efficient for small sizes of n it very quickly becomes a slow process when n gets high. This algorithm is tight bounded by insertion and bubble sort and lower bounded by shell sort. The empirical complexity tested and recorded in table one matches very well with the average complexity of $n^2$. It can be observed by looking at n= 40000 for selection sort and comparing time factor increase to size factor increase. Here the size factor increased by 8 and the time factor increased by 64 which would be exactly $n^2$. The rest of the observations for this algorithm further shows this finding.

Bubble sort iterates through an array and compares two values next to each other and has to go through less data each pass. This algorithm is very simple to make and takes very few lines of code however it is inefficient for large groups of data similarly to selection and insertion sort. This algorithm is tight bounded by insertion and selection sort and is lower bounded by shell sort. Of all the algorithms tested this seems to have the highest running times on average however its order is the same as insertion and selection. For bubble sort the empirical complexity that was tested and recorded in the table matches very well with the mathematical average time complexity which was $n^2$. When looking at different points in the table for this algorithm it is worth mentioning that the time factor is usually a bit more than $n^2$. However, the order of growth is still $n^2$ as can be seen when looking at multiple factors in the table. This is because while each time the factor is a bit more than $n^2$ that is the closest order that it grows with each time being slightly above $n^2$ rather than being a whole other order of growth.

When observing the algorithms with a increasing array and a size of 40000 it can be noticed that shell sort  takes almost no time and insertion sort takes so little time I couldn't capture it while selection and bubble both took a very long time. This is because the best case efficiency for shell sort is approximately O(n*log(n)) and insertion sorts best case is  O(n). Both of these best cases are very efficient compared to selection. Selection sort has a best case of O(n^2) which is very inefficient with larger sizes such as 40000. With bubble sort however the best case efficiency should be O(n) but with the way I setup the algorithm it is somehow still O(n^2) like selection sort. I plan to look into how I can optimize my algorithm.