William Santosa
wsantosa@ucsc.edu
15 February 2021

CSE13S Winter 2021
Assignment 5 : Putting Your Affairs in Order
WriteUp Document

Description

This WriteUp document discusses a variety of things regarding the implementation of my programs and each sort. Namely:

1) Identifies respective time complexity of each sort
2) Discussion about the constant of each sort
3) What I learned about each sorting algorithm
4) How I experimented with the different sorts
5) Graphs comparing/contrasting different sort performance
6) Analysis of graphs

Time Complexity

Bubble Sort: O(n^2)
Shell Sort (Pratt's Sequence): O(nlog^2(n))
Quick Sort: O(nlog(n))
Heap Sort: O(nlog(n))

Constant of Each Sort

Bubble

Bubble sort appears to have a constant of 0.75, as 100^2 * 0.75 is vaguely similar to the amount of moves that bubble sort has to make in order to sort the array. This pattern works for n = 200, 400, and 800 as well, indicated by these images.

Shell

Shell sort appears to have a constant of around 1.8 or 1.9, as the time complexity(nlog^2(n)) is a little more than half the amount of moves you have to make. This pattern, like for bubble, works for n = 100, 200, 400, and 800, indicated by these images.

Quick

Quick sort's constant appears to be between 1.8 and 2.5, as it follows nlog(n)'s increase but has slightly different moves at the end. This pattern works for n = 100, 200, 400, and 800, indicated by these images. This pattern works for n = 100, 200, 400, and 800, indicated by these images.

Heap

Heap sort's constant appears to be a little bit larger than 4 times Quick Sort's constant, as the sort has the same time complexity as Quick Sort yet runs 4 times as slow as it, indicated by the amount of moves it has to make.

What I Learned

I learned that each sort has a different time complexity and that these time complexities drastically affect the runtime of each sort when using arrays with larger amounts of elements. However, they also have a seemingly negligible effect on arrays with small amounts of elements. As such, I have learned to never use bubble sort to sort arrays with large amounts of data and that the other sorts have similar runtimes to one another, which make them all good to use.

Experimentation with Sorts

Adding on to what I have previously stated, I experimented with the sorts by using arrays with differing amounts of elements and I found that bubble sort exponentially becomes worse (indicated by its time complexity) as more and more elements are added onto the array. I also used numbers that doubled each time, from 100 to 200 to 400 to 800 to test out the number of moves and comparisons of each sort.

Analysis of Graphs

Bubble (graph)

Bubble sort's graph looks similar to an exponential graph as the run time increases slowly but near 20,000 elements it increases faster and faster. This drastic increase can be attributed to its time complexity and functionality, as increasing the amount of elements within an array means that it has to do that much more work to iterate through all of the array elements. As in, increasing an array from 200 elements to 201 elements would result in it having to do the work of 200 elements plus 201, which makes the run time increase dramatically in arrays with large elements.

Shell (graph)

Shell sort's graph looks almost linear aside from the initial start of the graph where it fluctuated a little bit. As such, it follows the time complexity of O(nlog^2(n)) as the graph of nlog^2(n) looks similar to a linear graph when zoomed out. However, as stated above, it is important to note that

it took about twice as long to sort 200,000 elements when compared to quick sort and heap sort, which makes shell sort an inefficient sorting algorithm to use.

Quick (graph)

Quick sort's graph began nearly flat until it reached 100,000 elements, where it shot up to .02 seconds. The slope decreased slightly as it reached 300,000 elements and .03 seconds, then increased in slope to reach .08 seconds at around the 500,000 elements mark. This makes Quick Sort much, much faster than the other sorts where Heap Sort is twice as slow than Quick Sort at sorting 500,000 elements.

Heap (graph)

Similar to shell sort, Heap sort's graph looks almost linear due to its time complexity of nlog(n). However, despite having the same time complexity as Quick Sort, it appears to be twice as slow and therefore less efficient to implement than Quick Sort.

Images (Graphs & Screenshots) *All Graphs Are Made From Miles' Script

Figure 1.

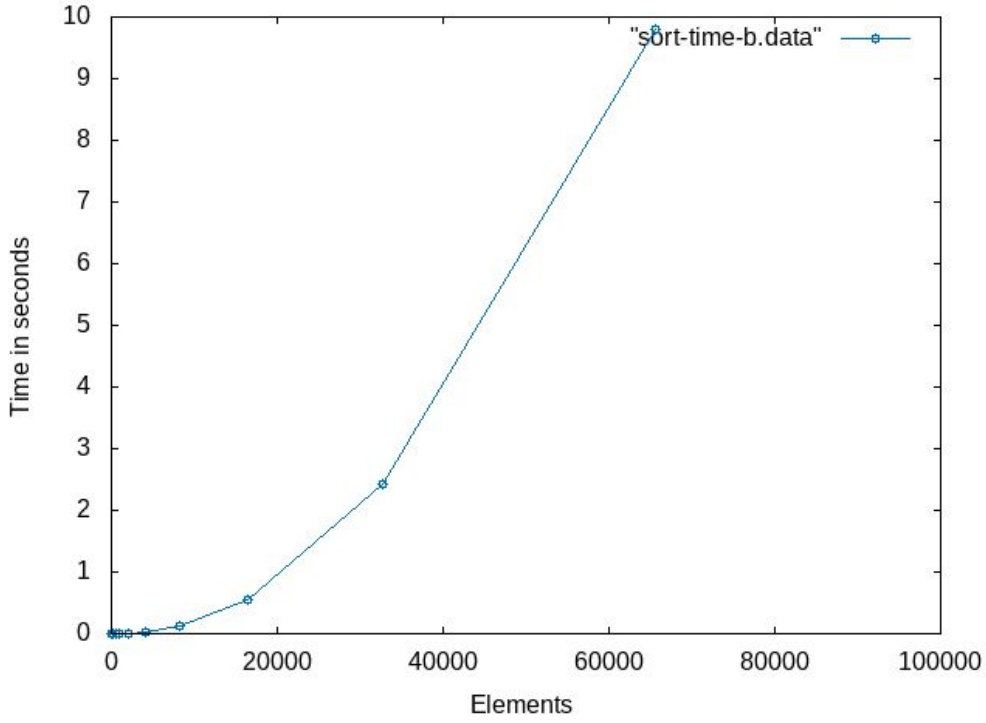

Figure 2.

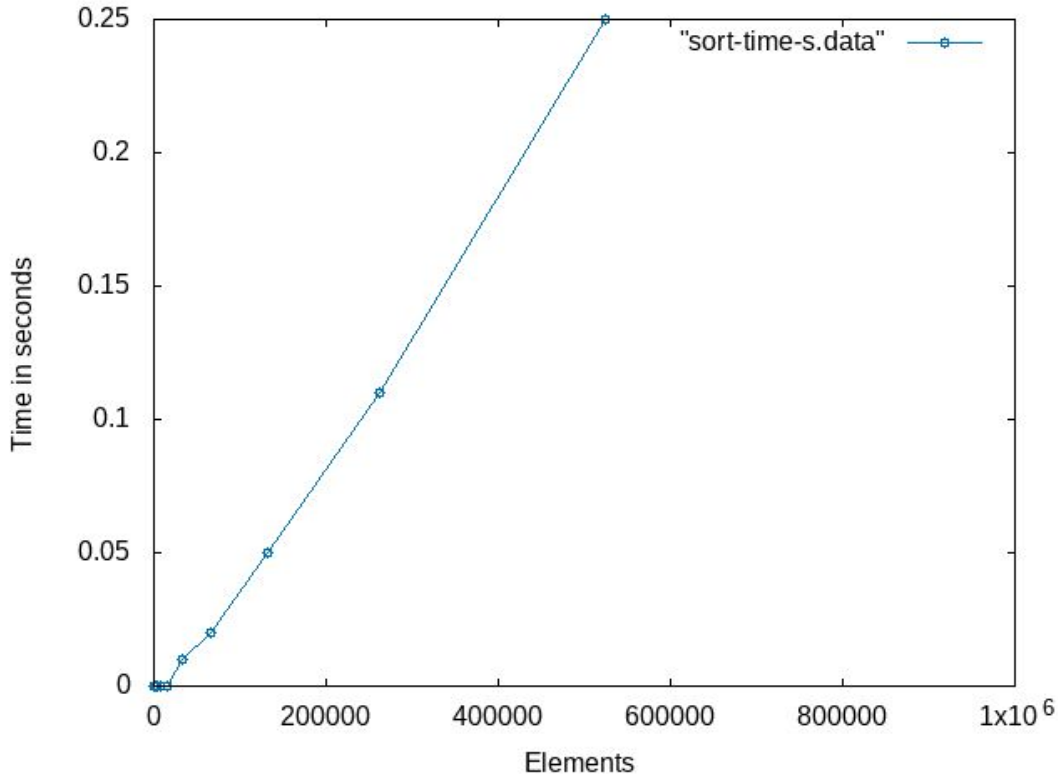

Figure 3.

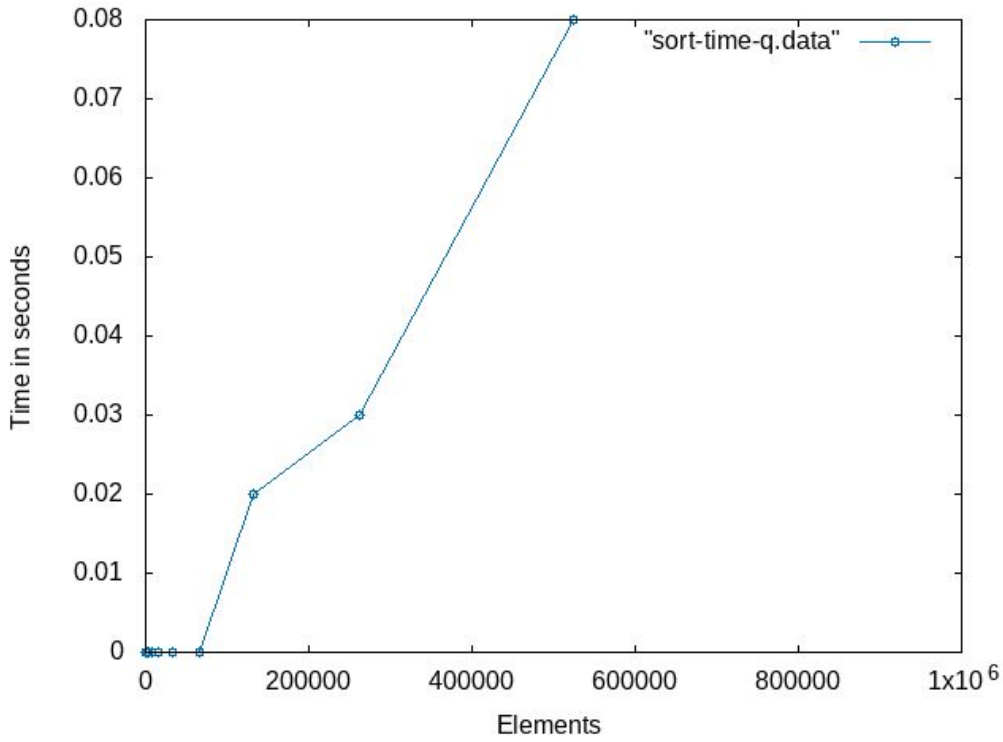

Figure 4.

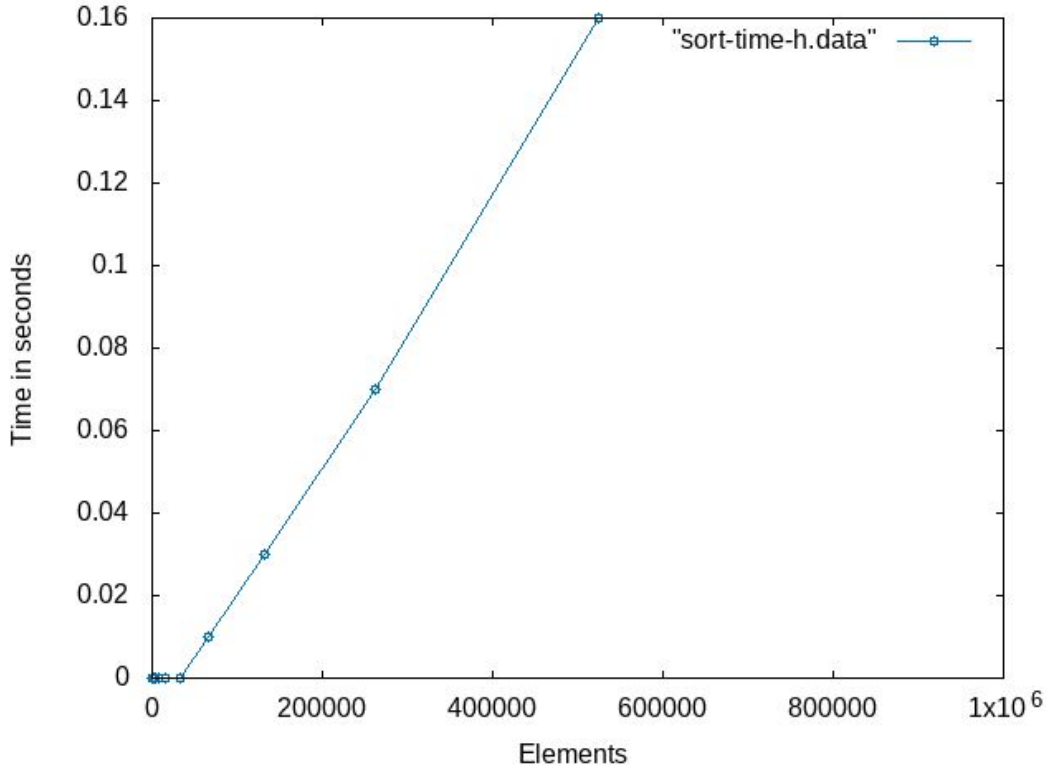

Figure 5. (Number of elements is 100)

```
will@will-VirtualBox:~/Desktop/wsantosa/asgn5$ ./sorting -a -n 100 -p 0
Bubble Sort
0 elements, 7548 moves, 5050 compares

Shell Sort
0 elements, 762 moves, 1489 compares

Quick Sort
0 elements, 462 moves, 976 compares

Heap Sort
0 elements, 1752 moves, 1151 compares

will@will-VirtualBox:~/Desktop/wsantosa/asgn5$ ./sorting -a -n 200 -p 0
Bubble Sort
0 elements, 29802 moves, 20100 compares

Shell Sort
0 elements, 1842 moves, 3755 compares

Quick Sort
0 elements, 1068 moves, 2129 compares

Heap Sort
0 elements, 4080 moves, 2694 compares
```

Figure 6. (Number of elements is 100)

```
will@will-VirtualBox:~/Desktop/wsantosa/asgn5$ ./sorting -a -n 400 -p 0
Bubble Sort
0 elements, 120837 moves, 80200 compares

Shell Sort
0 elements, 4143 moves, 9444 compares

Quick Sort
0 elements, 2397 moves, 5055 compares

Heap Sort
0 elements, 9291 moves, 6149 compares

will@will-VirtualBox:~/Desktop/wsantosa/asgn5$ ./sorting -a -n 800 -p 0
Bubble Sort
0 elements, 474618 moves, 320400 compares

Shell Sort
0 elements, 9486 moves, 23293 compares

Quick Sort
0 elements, 5394 moves, 10654 compares

Heap Sort
0 elements, 21150 moves, 13966 compares
```

References

1. https://piazza.com/class/khyix5qk2sw2nm?cid=913 (Note @913 / Thank you Miles!)