Algorithms Project 1 Report

Robert Andion U79333661

1. The algorithms that worked best on data equal to or smaller than 1,000,000 elements were quick sort and insertion sort under specific circumstances. Python has a severe limitation on the call stack and this prevents quick sort from working on all constant data due to it being the worst case scenario of O(n^2). However it was siginficantly faster than selection sort under all sorted or random data conditions, and faster than insertion sort under random data conditions. (It is to be noted that quick sort does work on smaller arrays of constant data and is faster than selection sort.) The only time quick sort would lose, is to insertion sort on constant or sorted data, where insertion sort is unmatched in complexity, θ(n), and where quick sort couldn't run constant data of these sizes. This is the condition that insertion sort is one of the best algorithms for this data size, although it is very slow compared to quick sort on random data.

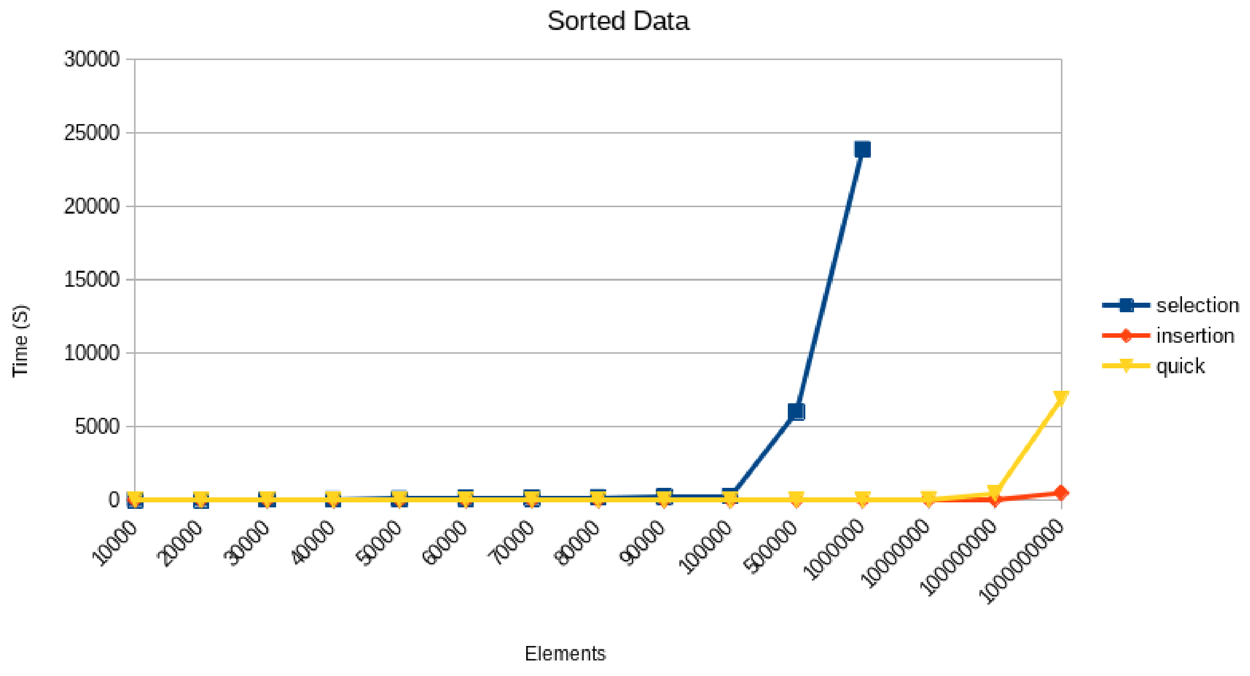
2. The best algorithm on 1,000,000 to 1,000,000,000 would be insertion sort under the condition that the data is either sorted or ordered. For this reason I declare insertion sort to be the best algorithm in this range as it can still run in 378 seconds for 1,000,000,000 constant elements whereas quick sort took 6859 seconds and selection sort showed no signs of progress after 2 days of running. The best case of θ(n) is shown to be significantly faster than O(nlgn) and by far better than selection sorts θ(n^2) run time

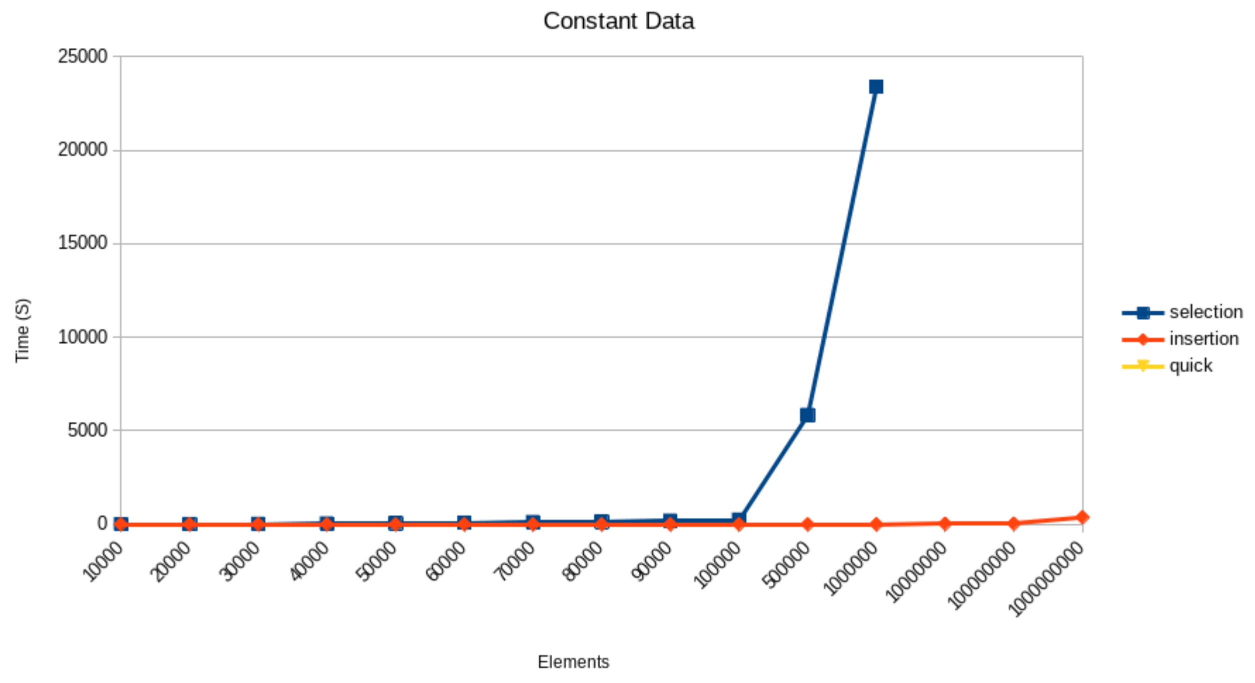
3. I was able to run insertion sort on constant and sorted data for 1,000,000,000 elements, and quick sort on sorted elements. This size array had trouble allocating enough memory for the actual array and I was only able to get it to successfully run with over 40GB of ram in use. With selection sort even when the array was built successfully, the run time would be longer than reasonable for this project, as it would run for more than a few days at only 10,000,000 elements and I had to stop testing further data for that reason.

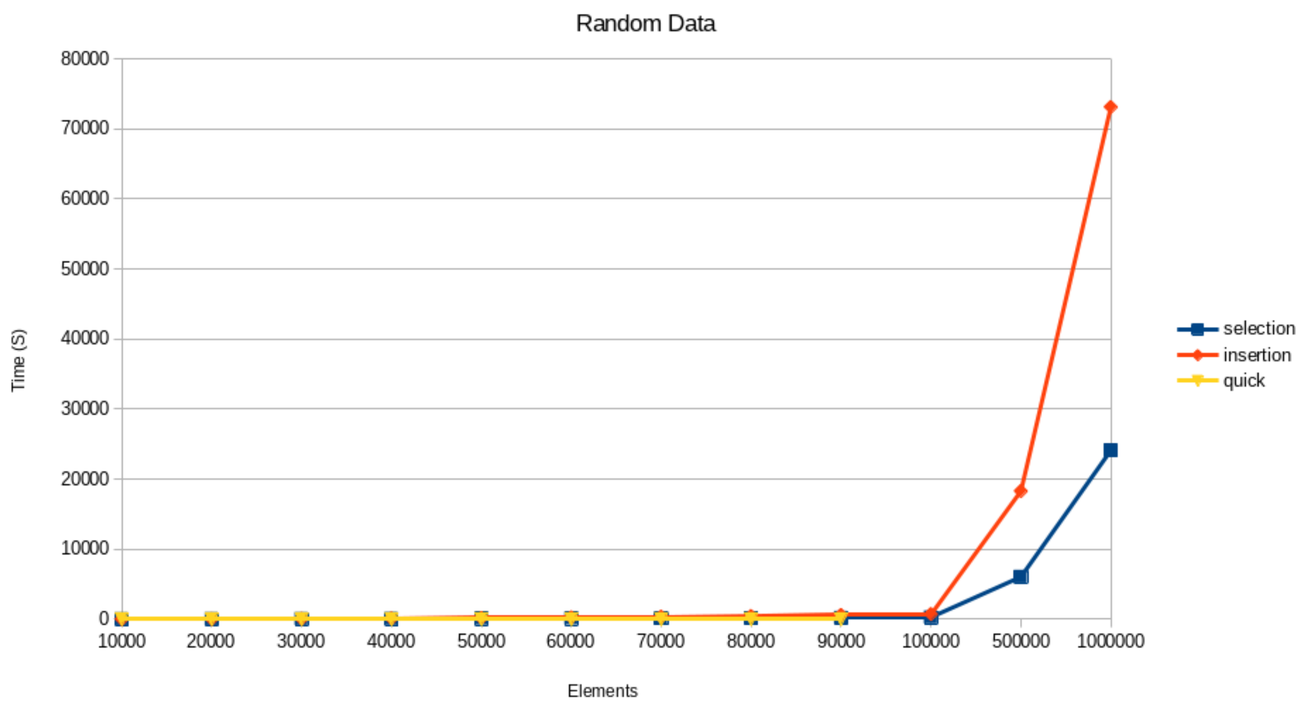
4. The results were not what I expected. I always imagined that selection sort and insertion sort were bad, but I never saw it at scale. In my programing I work with at most 1-2 thousand elements and selection or insertion sort get the job done seemingly instantaneously in my use case. However looking at larger numbers they become incredibly slow, and this really did surprise me as I always imagined modern hardware can do about anything instantly. I have learned why I do not want to write n^2 algorithms whenever possible. I was also greatly surprised at the massive scale that quick sort beats both of these algorithms, despite the data missing resulsts for constant as the stack did not allow it on data this size. Of course insertion sort wins with sorted or constant data, but is this really "sorting"? Most of the time we are dealing with random data and quick sort blows them out of the water. Even with only ten thousand elements quick sort is able to sort random elements in 0.04 seconds while selection sort and insertion sort both take more than 2, and 7 seconds respectively. I was more shocked at the speed of n lg n more than anything else. The lg n time makes a massive difference in computation time and I was able to connect this theoretical knowledge to actual programming applications. This also showed the justification of big oh in that small differences over large scale do not matter. Any of the constant operations were essentially meaningless and only the amount of iterations mattered. Even though insertion sort can be better than n^2 with random data, it was shown that it takes so long when approaching large numbers that it really makes no

difference whether or not it terminates a bit before actually reaching n^2, it is still going to take weeks, months or even years to run, which justifies the approximations made by big oh.

5.







Note for Random Data none were able to go above 1000000 elements in reasonable time, or without stack overflow, these also go back to the notion of Big Oh justification in question 4, the O(n^2) cases are so large it makes the other cases seem like zero computation time in relation.