**Introduction**

This is my report for this sorting algorithm project. For my project, I chose to use bubblesort, quicksort and mergesort.. My reasoning for choosing an algorithm like bubblesort despite its simplicity in comparison to the other algorithms is that I wanted to compare the difference in timing between a slower algorithm and some faster algorithms, as well as because I just like the name. I chose quicksort and mergesort as algorithms to compare bubblesort to, as they are both objectively “faster” algorithms than bubblesort, but I wanted to see how much faster they were.

**Datasets**

I used a few different types of datasets, including random, sorted, reversed and partial. I chose these datasets because they were suggested in the project report and I wanted to test how each algorithm would work with these datasets. I wouldn’t necessarily call my choice in datasets all that imaginative, as I only really chose a few standard ones, although I did have my own reasons for using the ones that I have. Partially sorted, as I wanted to see how each algorithm would deal with an array that was already partially sorted. Reversed, to see how each algorithm reacted to having to sort in ascending order, when the data is already sorted in descending order, and sorted, as I wanted to see how each algorithm dealt with what one would assume would typically be the best case for most algorithms.

**Algorithms performance**

Mergesort had the overall best performance out of all the algorithms, which was to be expected considering that its worst case was linearithmic, which is considerably better than quicksort and bubblesort, which both have quadratic runtimes for their worst cases.

As expected for bubblesort, it performed the worst overall in terms of runtime compared to the other two algorithms, although I can appreciate that it at least sorted 1 million integers, despite taking up to 45 minutes to do so. Interestingly, bubblesort’s best case was when the array was already sorted, as the algorithm only did comparisons, which decreased the time it took to run significantly. Bubblesort’s worst case ended up being random, as for each array size, it can be observed that random takes the longest to run in comparison to the other datasets, taking up to 36 minutes to run on 1 million integers (refer to sorting-times.txt).

Quicksort is the strangest algorithm that I tested so far, as it seems to do well on datasets like random and partially sorted, but does horrendous on reversed and sorted arrays. The dataset that quicksort performs the best on is a partially sorted array, although it is only better than a random array by a slim margin. The absolute worst case for quicksort is a sorted array, with quicksort taking up to 35 seconds on 100,000 integers (refer to the 100,000 ints section of sorting-times.txt in the report directory to see my point). With either a sorted or reversed array, quicksort chooses one of the first elements to be its pivot, which leads to the worst possible case for quicksort. Unfortunately, I was unable to record the times for quicksort on reversed and sorted arrays of 1 million integers, as both cases throw up a segmentation fault after a few minutes, and end up not finishing, although this tells me that quicksort requires a lot more memory to sort arrays of this size than something like bubblesort.

Mergesort had the lowest average time for each array size, with minimal differences between each dataset, although as the size of the array increases, it can be observed that the “worst” case for mergesort is a random dataset, although the difference between a random array and other datasets is minimal and only really becomes obvious at around 100,000 to 1 million integers. Overall, mergesort has the most consistent sorting time for any algorithm I have tested so far, with a much higher ceiling than the other two algorithms. Even at 1 million integers, mergesort takes less than a second to sort, even at its worst case, definitively proving that between itself and quicksort, that it is the overall fastest sorting algorithm within this group.

**Negatives**

One of my main regrets with this project was not spending enough time trying to figure out why quicksort throws a segmentation fault when attempting to sort either a reversed or sorted array of 1 million integers. While I feel it’s due to memory allocation, I can’t say for sure what causes this issue. Another issue that I wish I had fixed sooner was how I allocated the memory for the main array and the copied array, as I initially made the mistake of copying the main array into four separate arrays, which really didn’t bode well when it came to creating and sorting arrays above a certain threshold.

**Conclusion and future work**

Overall, it can be concluded that mergesort ended up being the best algorithm that I have used for this project, as it ran quickly and efficiently while also having the best worst case out the three algorithms, with quicksort being the second best, and finally, to no one’s surprise, bubblesort is the worst.

One thing I would like to do if I had more time is to implement more of a variety of datasets to work with, such as an array that has one element out of order, or all elements the same etc.

The second thing I would do if I had more time would be to implement another “bad” sorting algorithm, like bogosort or something similar, and just wait for the algorithm to sort for maybe 6 hours.