Written Report – CA284

1. Introduction

This is my report on my CA284 project. I used 5 algorithms throughout this project, they are selection sort, insertion sort, pancake sort, quick sort, merge sort. Selection sort and insertion sort are beginner type algorithms and were used mostly in the testing grounds. Quick sort and merge sort are good for integrating another function to do most of the work. Pancake sort brings in multiple functions to do the work and then executed in the pancake sort function.

1. Datasets

To be honest, I didn’t get very creative in this section, I had a lot of trouble trying to implement datasets that were harder than the basic one so I decided not to the most time on the part with the least amount of percentage going for it, so I skipped over the creative areas and would have went back if I had gotten the time to, in the end I didn’t get the time.

1. Algorithms

When the number of elements grew in the dataset, the algorithms performance grew at a steady rate. The timing never changed drastically throughout the testing phase. As the numbers grew it really took a long time to execute which shows how much longer it takes with a much bigger sample case, for example: 2 million numbers. Because I didn’t use any creative number generators, I don’t know how much different the timings were. Although I would imagine a nearly sorted array would be execute much faster than a randomly generated array, the same with a partially sorted array. A randomly generated array generally took a long time to return sorted due to numbers being generated in an order unknown to the user. You can get a small chance that the array may be nearly or partially sorted when generated randomly, but it is rare. Pancake sort is an interesting algorithm, it was the first time I had come across it and I wanted to implement it, so I had to learn about it first. It is great for getting used to calling functions through different function. There were big differences between 10 integers and up to 2 million integers. You could see the difference once you hit 100,000 integers because of the length of time it took to execute. From 10 to 10,000 the run time wasn’t too long, but then you found yourself sitting there wating for to execute. It really does show how quick a computer can sort a sequence of randomly generated numbers in different sorting algorithms, which made it an interesting project.

1. Negatives

I could have done a lot better, especially by introducing more datasets, it feels a bit plain, but I wanted to nail down the rest of the code. I also would have liken to change the timing method of the code but I couldn’t get it to work the way I wanted it to work in the end, so I had to settle for what I had down.

1. Conclusion

All in all, I really enjoyed this project. The challenge is great and very rewarding when you get past a part you were stuck on. I would have likened to create more array types if I had more time. Also, I would have likened to nail down the timing aspect a bit more, it is a bit untidy and not working entirely perfect. I will be working on this project in my spare time to fully complete it by adding in these features I have pointed out and making some adjustments style wise.