AMELIA GRIGORIEV

19348241

**CA284 SORTING ALGORITHM REPORT**

**Introduction**

For this project, I chose the following sorting algorithms: Selection Sort, Radix Sort and Heap Sort. I chose these algorithms because I understand them well and wanted to expand my knowledge and understanding of these algorithms. Selection sort was a great algorithm to start with as it is pretty simple and easy to understand, so it was a great choice to get me started on the project. Next, I chose radix sort because it is a little more difficult than selection sort and I wanted to challenge myself. I also wanted to have some variety in the difficulty levels of my sorting algorithms. For my last algorithm I chose heap sort as it was one that I wanted to learn a bit more about.

**Datasets**

I chose the following datasets to test my algorithms: sorted in reverse, partially sorted, sorted in numerical order and random meaning no particular order. The generated integers in each of my datasets are all less that or equal to 1,000,000. I chose these datasets because I felt that they offer a good range of different data for the sorting algorithms to be tested with. These datasets include different stages of the sorting process such as not sorted at all (the random dataset), partially sorted and then already fully sorted. I thought it would be interesting to see how long the sorting algorithms take on these stages of sorting. I was also interested in seeing how long it would take the algorithms to sort data that is already sorted but in reverse. The partially sorted algorithm is only the first half of the dataset sorted numerically.

**Algorithm Performance**

The results of how long each sorting algorithm took on each dataset can be viewed in the results.txt file in test. When the selection sort algorithm had to sort the 1,000,000, 2,000,000, and 5,000,000 integer datasets it took too long to sort so I did not come up with a time for those. The highest number of integers I tested with selection sort was 1,000,000 integers. These results can also be seen in the results.txt file. The other two sorting algorithms, radix and heap sort, did not have any issues sorting the larger datasets. Both algorithms sorted the 5,000,000 integer datasets in a couple of seconds or under.

Before starting this project, I had my predictions and assumptions about which datasets the algorithms would sort the quickest. Now, after testing the sorting algorithms on the different data myself, I can clearly see that a lot of my predictions were wrong. I thought that the already numerically sorted data would take very little to no time as it is already sorted but after running the algorithms, I can see that this is not the case. This revelation definitely shocked me as I was not expecting it. I also assumed that the partially sorted data would take very little time to sort as it was already halfway there, this however, was also not the case. I also thought that the reversed order data would take the longest to sort by the sorting algorithms. This was somewhat true. For some of the sorting algorithms, the reversed data took the longest to sort, and for some it didn’t.

For each of the sorting algorithms, using the same number of integers with different datasets didn’t make too big of a difference. The result was always around the same. When testing selection sort, the already numerically sorted data seemed to be the fastest. For radix sort, the random data, partially sorted data and numerically sorted data seemed to fluctuate with being the fasted whereas the reverse sorted data seemed to be the slowest. For heap sort, testing the algorithm on random data appeared to be the slowest, and testing on reversed data seemed to take the least amount of time. Selection sort was visibly the slowest algorithm to be tested and although it took a very small amount of time to sort the datasets with a small number of integers, as can be seen by the results in the results.txt, it clearly took the longest amount of time larger amounts of integers such as 1,000,000.

**Negatives**

A negative is definitely how long the selection sort took in my opinion. I also would have liked to test the data on a dataset that had the second half of the data numerically sorted as I already have a dataset which has the first half of the data numerically sorted. Another dataset which I think would have been a good addition is a dataset where every second number is numerically sorted. I also had some issues with memory allocation which frequently led to segmentation faults. I eventually fixed these errors by using malloc memory allocation. This was a useful lesson for me to learn as I feel like this will solve many issues for me in the future before they even happen. While working on the project, I also ran into some issues with reading and writing to files as it was my first time doing this in C. This issue was quickly resolved, and I feel like it was an extremely valuable thing to learn. One other thing that I would have liked to implement into my code but had a bit of difficulty implementing is an option for the user to choose which file the numbers go into, as my code only works with the numbers going into the nums.txt file in test. I am glad that I am now aware of these issues that may arise in future projects and that I now know how to fix them.

**Conclusion**

In conclusion, I have learned a lot about various sorting algorithms, and I was surprised at some of my findings. Before starting this project, I had my predictions about which datasets the algorithms would sort the quickest. The results were unlike my predictions which shocked me, and it is something that I find quite interesting. If I were to continue with this project, I would most definitely like to add a bunch of other sorting algorithms as well as more datasets to test on. There are so many other sorting algorithms to choose from and I think it would be very interesting to implement them into a project like this one. I would have also liked to add more datasets for the algorithms to be tested on.