# Report

## Introduction

The algorithms I chose to sort out my elements are selection sort, insertion sort, radix sort and shell sort. I picked selection sort as it is a relatively easy program to write and to initially test your number generator on. Insertion sort works like selection sort but places the element in its correct position. I picked radix sort as it is a complex algorithm and I wanted to test my skills. I also picked to do shell sort as when doing research for different sorts, shell sort intrigued me the most.

## Datasets

The first dataset that I used to test on my algorithms was an unsorted dataset that was being generated by the program “num\_generator.c” I was struggling to find a way for my number generator to output different datasets. I managed to come up with a solution where my program was able to output a half-unsorted dataset, as the original program outputs a fully unsorted dataset. The slight edit which I did in the program can be seen on line 14 of the “num\_generator.c” program in the src directory.

## Algorithm Performance

### Unsorted Performance:

I tested the algorithms from one program called “main.c” I tested 100, 1000, 100000 and 200000 elements. As expected, the runtime for each algorithm increased as more numbers were tested. I noticed that for the lower data the runtime would not be displayed in seconds, so I used the <sys/time.h> header which allowed me to use the member “tv\_usec” this can be seen on line 138, line 144, line 150 and line 156 in the “main.c” file in src. This member let me record the milliseconds it takes for the program to run. As expected, the runtime of radix sort and shell sort was much quicker than insertion and selection sort. Shell sort performed the best for the two lower lists however radix sort was better for larger numbers. Shell sort is much quicker than insertion and selection as for shell sort you do not need move newly sorted items up one position each time like you do in insertion. Shell sort can relocate items by swapping places. Radix sort is not comparison based and you can choose how many buckets to use and the memory you wish to allocate, due to this radix sort can be faster than O(nlogn). Insertion was slightly slower than selection which I also expected as they are similar but selection sorts slightly faster.

### Half-Unsorted Performance

As I would have expected, the performance was very similar when I used the half-unsorted dataset compared to the unsorted dataset. Shell sort was the quickest for the lower amounts again. How ever radix and shell were interchanging for which program ran faster, for 100,000 shell sort was quicker, however for 200,000 radix sort pipped the speed. What I did notice is that overall, the programs ran faster with a half-unsorted dataset.

## Negatives

I am not happy about not being able to successfully generate more datasets to do more testing. I initially struggled to get grips implementing my number generator into a driver file that would run all the algorithms but after some research and testing I managed to do that. I want to improve my notation of my code and be able to write more concise and possibly faster code.

## Conclusion and Future Work

From doing this project I have managed to observe, test and learn that programs are affected when they are run with different datasets. I also gained a lot of experience in managing my code in a project environment, as this is something I have not done before.

If I would have had more time, I would have researched how to output my number generator into a .txt file and use that as an input for each algorithm. Another thing I would have done with more time would have been to come up with more complex datasets.