**1. Introduction**

To decide what sorting algorithms I would use for this project, I wanted to choose algorithms that would not be commonly used amongst other CA284 candidates. I also wanted to implement new programmes into my project as algorithms such as insertion sort or selection sort which I have already done before, would not benefit the potential learning outcomes of this project as new algorithms would. I also knew that insertion sort and selection sort would be commonly used by other candidates. The first sorting algorithm I wanted to use was radix sort as I heard about its good time complexity and memory allocation. However after subsequent research of radix sort I came to the conclusion that the sorting algorithm was a bit difficult for me to comprehend it completely and I saw no point in forcing it into my project. That's when I came across bucket sort and counting sort. Both these sorting algorithms which are similar to radix sort on many levels were much more responsive to me and my ideology. After implementing these algorithms I came to the conclusion that they were indeed very fast sorting algorithms especially at handling large scale inputs. To add a bit more diversity to my project I wanted to include a slower algorithm. That’s when i came across quick sort, which is still quite a fast sorting algorithm but compared to bucket sort and counting sort it was a fair bit slower.

**2. Datasets**

Testing the various algorithms with different types of datasets was interesting. I first started testing my algorithms with an array of randomly sorted integers. All programmes worked fine and quite efficiently too. After testing with a randomly sorted array I came to the conclusion that bucket sort and counting sort had a relatively close time average where quicksort was slower than the two.

When the data set was an array which was already sorted it was clear to see that counting sort and bucket sort again was really close but this time counting sort had a slight time advantage both being faster than when the array was randomly sorted. Again quicksort was the slowest of the three being even slower than when the array was randomly sorted.

Finally I tested with an array of reversed sorted integers, with this final test I deduced that counting sort was the fastest faster than, bucket was the second fastest and quicksort being the slowest for the third time in a row. This time counting sort and bucket sort were the same speed as when the dataset was sorted and quicksort were slower than when the array was sorted.

**3. Algorithm Performance**

Sorting algorithm - Bucket sort

Bucket sort was the first sorting algorithm that i chose. When I first started testing with bucket sort I was surprised with how fast the algorithm was and how well it worked with large data sets. As I said earlier I began testing my algorithms with an array of randomly sorted integers. With relatively low data sets (0 - 270000) bucket sort has a time of 0.00 seconds. However I noticed that this stroke of time does not simply stay at this rate, when I tested with an array of 275000 integers the time taken was 0.015625 seconds.

When working with very large data sets the result averages are as follows:

1,000,000 = 0.031250

2,000,000 = 0.046875

5,000,000 = 0.140625

10,000,000 = 0.296875

When I tested with 1,000,000 and 2,000,000 integers I could see that the time taken between the two did not double given the size of the data set, however when I tested with 5,000,000 and 10,000,000 integers I could see that the time taken between the two were quite close to doubling.

When using sorted and reverse sorted arrays it was clear that bucket sort worked best with these types of datasets as the time taken for most datasets sizes worked faster especially at large scale datasets.

Sorting algorithm - Counting sort

Counting sort was the second algorithm I chose to implement into my project. After having experience with already testing bucket sort the examination of this sorting algorithm went much smoother. I again started to test with randomly sorted datasets. Like bucket sort, counting sort had an extremely low time increase with small data sets with about the same range of (0 - 270000) with a time of 0.00 seconds, and begins to increase at around 275000. When working with very large data sets the result averages are as follows:

1,000,000 = 0.015625

2,000,000 = 0.046875 (same as bucket sort)

5,000,000 = 0.156250

10,000,000 = 0.312500

With this information I deduced that at smaller intervals counting sort is slightly faster and at larger intervals bucket sort is slightly faster. From this information I still think that both programmes have a same time complexity.

When working with reverse sorted and sorted datasets I noticed that they both had about the same time average for most data set sizes and like bucket sort was quicker when the data set was randomly sorted.

Sorting algorithm - Quick sort

The last sorting algorithm I decided to implement into my project was quick sort and for reason. After noticing the speed of counting sort and bucket sort I wanted a slightly slower program to compare to the other algorithms.

I noticed that the speed of this algorithm increased at a much faster pace than the other two algorithms, I found that at 100,000 the algorithm speed was at 1.531250 seconds, 200,000 the algorithm speed was 6.062500 seconds and 300,000 the algorithm speed was 13.750000 seconds. And after that the time taken for increased dataset sizes was exponential. For reverse sorted and sorted arrays the time taken for the algorithm to work was unusually longer than when the array was randomly sorted. From this I deduced that quick sort was not a great sorting algorithm when sorting larger scale datasets

4. Negatives

I feel my quick sort program could work better and a lot better at that. The algorithm was very inconsistent where sometimes it would work for a certain data set size and another time, I would get an error such as a “segmentation fault”. I also feel like counting sort and bucket sort are very similar, at first when testing the algorithms I thought it was actually good that they were both similar but after initial thought I came to the realization the learning outcomes of this project could have been more constructive if the two algorithms differed more. Overall I was happy with my algorithm choices and the data sets I decided to use to test my program.

5. Conclusion and Future Work

So in conclusion I feel this project has been quite beneficial to me as a programmer and my development for future projects.If i had more time i would definitely try to find a way for quick sort to work on larger data sets but i could not find this solution with the time given. I would also like to add a more user friendly interface when starting up the program to make it as easy as possible for the program to be used. In the future I think i would like to continue this analysis beyond its current state as unlike other projects I quite enjoyed this one as my freedom was my own.