Introduction

The algorithms I chose are Selection sort, Insertion sort and Radix Sort. The reason I chose Selection and Insertion sort is because these are sorting algorithms I use them often so being able to improve their performance would be beneficial for my coding skills. Selection sort takes the first item in a list and compares it to each number in the list. It swaps it with any number smaller than it. Every number is compared to each number in the unsorted part of the list. Whereas with Insertion sort, the number is only compared with its neighbours and then swapped. It continues swapping with its neighbour until it reaches a number bigger than it. The final algorithm is Bubble sort and I chose it as it is very similar to Insertion sort but instead of swapping an item until it reaches a number bigger than it, bubble sort swaps the item with its neighbour if needed. It then moves on and compares the next numbers.

Datasets

I chose 4 different datasets. The first is completely unsorted data and the reason behind this is that unsorted data is very common and requires the most swaps which costs a lot of time. The second is sorted data which I chose as this would show how fast the algorithm can be when it doesn't need to stop. The third set of data is partially sorted. The data has sorted data which then repeats itself. For example, 123123. I chose this as it some algorithms are meant to sort partially sorted like Bubble sort so I thought it would be interesting to see how it performs. The final data set is completely sorted data but it contains one number out of place. For example, 123645. I chose this as each algorithm sorts differently and would require different amount of swaps. This would show which has more swaps as the times would be larger as swaps take more time.

Selection.c

10	*Real*	*User*	*Sys*
Unsorted	0.003s	0.002s	0.001s
Sorted	0.003s	0.001s	0.003s
Partially sorted	0.003s	0.002s	0.001s
One wrong	0.003s	0.003s	0.000s

100	*Real*	*User*	*Sys*
Unsorted	0.005s	0.005s	0.000s
Sorted	0.005s	0.000s	0.004s
Partially sorted	0.016s	0.009s	0.008s
One wrong	0.017s	0.009s	0.009s

10,000	*Real*	*User*	*Sys*
Unsorted	0.184s	0.166s	0.012s

Sorted	0.175s	0.155s	0.017s
Partially sorted	11.769s	11.549s	0.168s
One wrong	0.182s	0.166s	0.012s

Insertion.c

10	*Real*	*User*	*Sys*
Unsorted	0.003s	0.001s	0.003s
Sorted	0.003s	0.003s	0.000s
Partially sorted	0.003s	0.003s	0.000s
One wrong	0.003s	0.003s	0.000s

100	*Real*	*User*	*Sys*
Unsorted	0.004s	0.004s	0.000s
Sorted	0.003s	0.003s	0.000s
Partially sorted	0.013s	0.009s	0.004s
One wrong	0.013s	0.001s	0.013s

10,000	*Real*	*User*	*Sys*
Unsorted	0.016s	0.126s	0.020s
Sorted	0.092s	0.017s	0.075s
Partially sorted	4.800s	4.590s	0.160s
One wrong	0.030s	0.013s	0.017s

Bubble.c

10	*Real*	*User*	*Sys*
Unsorted	0.003s	0.003s	0.000s
Sorted	0.003s	0.004s	0.000s
Partially sorted	0.003s	0.001s	0.003s
One wrong	0.003s	0.001s	0.003s

100	*Real*	*User*	*Sys*

Unsorted	0.026s	0.013s	0.013s
Sorted	0.020s	0.005s	0.015s
Partially sorted	0.021s	0.011s	0.010s
One wrong	0.018s	0.010s	0.009s

10,000	*Real*	*User*	*Sys*
Unsorted	0.398s	0.373s	0.017s
Sorted	0.186s	0.169s	0.012s
Partially sorted	16.392s	16.160s	0.168s
One wrong	0.181s	0.158s	0.016s

Algorithm Performance

The performance of all three algorithms is very similar when it came to the smallest data set of 10 integers. They all have the exact same run time for real. But as the data set gets bigger, Insertion sort is overall the fastest and Bubble sort is the slowest. Bubble sort being so slow compared to the other algorithms is probably because Bubble sort only compares a number with its neighbour which means it takes a lot longer as the others can compare a number to multiple numbers in the list until it finds a number bigger than it.

If we look at each data set which has 10,000 integers, we can see that the partially sorted data has taken a greater time for each algorithm than the other data sets took. Insertion sort was a lot faster than the other sorting algorithms by 0.07-0.12 seconds. The reason behind why Insertion was so much faster than Selection sort is likely because of the fact that Selection sort compares will go through the entire list every time. Whereas Insertion sort stops once it encounters a number smaller than itself so the algorithm doesn't go through every single number every time. Also, Selection sort has a complexity of O(N to the power of 2) which makes it slower on larger data sets. Insertion sort is faster than Bubble sort probably because of the fact that Bubble sort only compares one number to its neighbour and then moves on to the next number. It is of note however that Bubble sort is usually used for partially sorted data so it is interesting that it would be the slowest considering this show be its preferred data set. It doesn't keep looking with the same number until it has found a number smaller than it. Also in the data set with 10,000 integers, Insertion sort did well with the data set that contained one integer out of place. This likely down to the same reasons Insertion sort did better with the partially sorted data set.

The different algorithms all had their own strengths and weaknesses but overall Bubble sort and Selection sort were best with sorted data. Whereas the Insertion sort was were best with unsorted data. What is very interesting is the fact that for all three, the partially sorted data set took them longer than the others as the data set increased. The reason Bubble sort and Selection sort prefer sorted data is because they don't have to stop and swap any numbers which means they just read through the data and be done quickly.

Negatives

The main negative with the code I have is that the part of the code which reads in the data has a memory limit which means you have to know the data set's size. As you can see in the line, "#define MAX_ARRAY 1000000". This limited the size of the data sets I used.

Conclusion & Future Work

In conclusion, overall Insertion sort was the fastest of the three sorting algorithms I chose and the Bubble sort was the slowest. As expected, all three were slower when they had bigger data sets but the increase in time required was much bigger for Bubble sort and Selection sort. If I had more time, I would find a way to remove the limitation on how much memory the code has for reading in data. I would also look into is malloc is the most efficient way of storing this data.