CM1210 coursework 2 report

In short report I shall be talking about the recently completed 2nd coursework in which we were tasked with creating 3 different programs. In this report I will be talking mainly about the 2nd program but will also talk a little bit about the 3rd program.

For the 2nd part we were tasked with coding a merge sort and an insertion sort program. We then had to count the amount of comparisons and swaps that were made in each algorithm, we also had to time it as well. We then had to do all of this for 100, 200, 300 and 400 words in the text document.

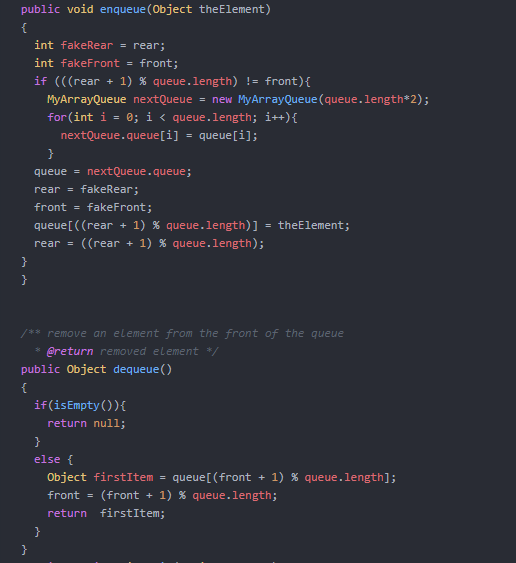
I have displayed the results of the timings, comparisons and swaps 3 different graphs. I grouped the insertion sort, comparisons and moves, in one graph and the merge sort in the other. I then compared the two different algorithms times in one graph. As we can see from the graphs the merge sort does a lot more comparisons and moves yet follows a similar trend to the insertion sort where the moves and comparisons are similar in number. For the time we see that the insertion sort is a lot faster up to 300 words when the merge sort starts to get quicker. After this we can see that insertion sort rises rapidly only for 400 words when it come down again at 468 words. At the same time the merge sort becomes faster. This is probably an anomaly and we would need to have a larger sample size to see if this happens again at other words.

To find these results I took the average of running the code 5 times.

To run the code simply open a command line and navigate to the “Sorting.java” file. Then type in “javac Sorting.java” followed by “java Sorting” and it should run.

To start developing the coursework I went back and did some reading on merge and insertion sort algorithms and queues in java. I found this helpful as it gave me a broad view of what I needed to do. I then moved onto writing a high-level pseudo code for the different tasks. This was the most useful step as I then knew exactly what I needed to do and now just needed to write it out in java. This helped a lot.

Here are two snippets from the sorting algorithms. As you can see the merge sort is a lot longer which effects how long it takes to run.



This is a screen shot of both parts from part c. The enqueue starts of by naming two variables which act as a temporary front and rear value. It then adds the value to a temporary queue in the right position in the queue. It then sets the queue, rear and front values to the temporary values.

For dequeue it is a lot simpler. It starts by checking that the queue isn’t already empty. It then starts to delete each item starting from the front first. Returning the item that it deleted at the end.