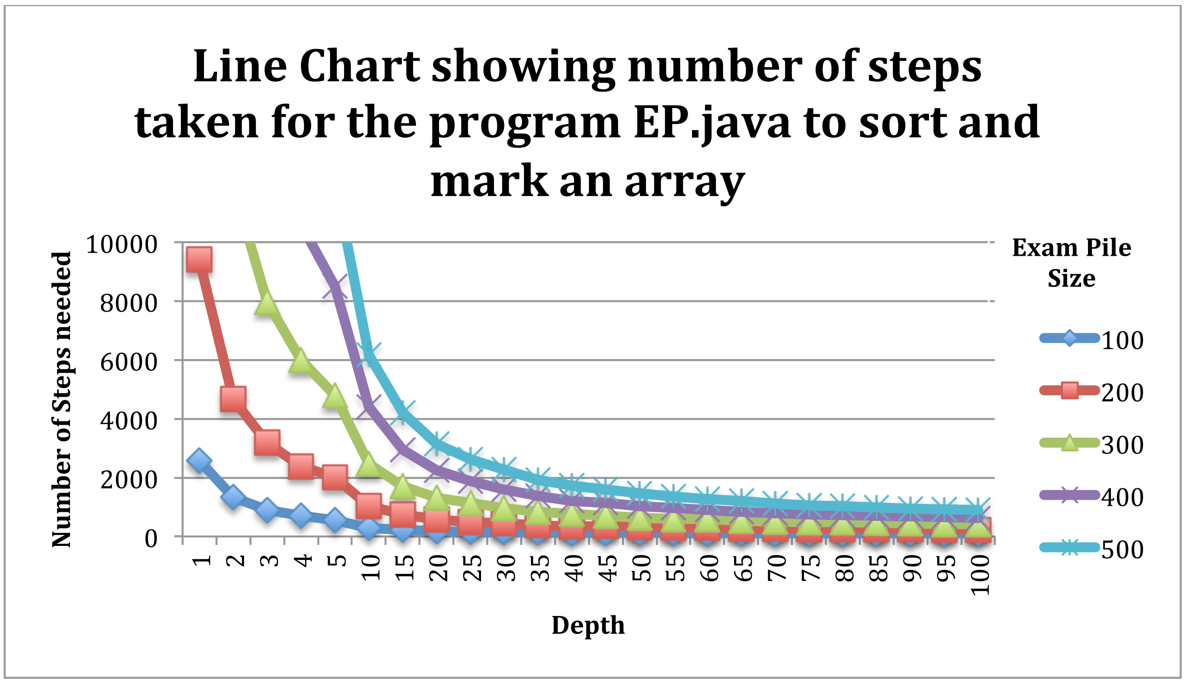

In order to understand algorithmic efficiency we need to consider two principles - time efficiency, and memory efficiency. We overlooked memory efficiency as we did not have a limit on the amount of memory we were able to use to complete this task, and as memory is very cheap these days it is less important. Our main focus was on time efficiency, in order to analyse this we looked at the number of steps our program took to sort through our exam pile. The method sortingSteps was used to process the exam pile, and after testing the algorithm's contained within the program on a wide range of different exam pile sizes, we discovered some interesting results. We decided to test exam piles with sizes of 10, 100, 200, 300, 400, 500, 1000, and 10,000. From this we discovered that it took just over 4 times longer to search through an exam pile of size 1000 then it did for a size of 500 with a depth of 1. When we tried to test the exam pile of 10,000 elements with a depth of 1, we could not get it to run in the timeframe given, this immediately gave us a clue as to what type of BigO behaviour we were observing.

As well as recognizing a pattern between the exam pile size and the amount of time it took to process, we also noticed a difference between the depth and the number of steps it took to process an exam pile of a specific size. We observed the general pattern that when the depth was increased, the number of steps taken to sort through the exam pile was decreased. For example the bigger the depth, the more likely the value we are searching for would be found and therefore the mark method would be called. This is more efficient as no elements in the exam pile have to be delayed to the bottom of the pile before the mark method is called. This alerted us to the fact that the worst-case scenario would always be a depth of 1. From these results we were then able to look back at the algorithms within the program and notice they were displaying typical behaviour of BigO notation of n^2 (exponential growth). We realise that this behaviour we are witnessing is the worst-case, and therefore to fix this we could use a binary search tree instead of an Array List to sort through the exam pile, this would instead give us BigO behaviour of log n.