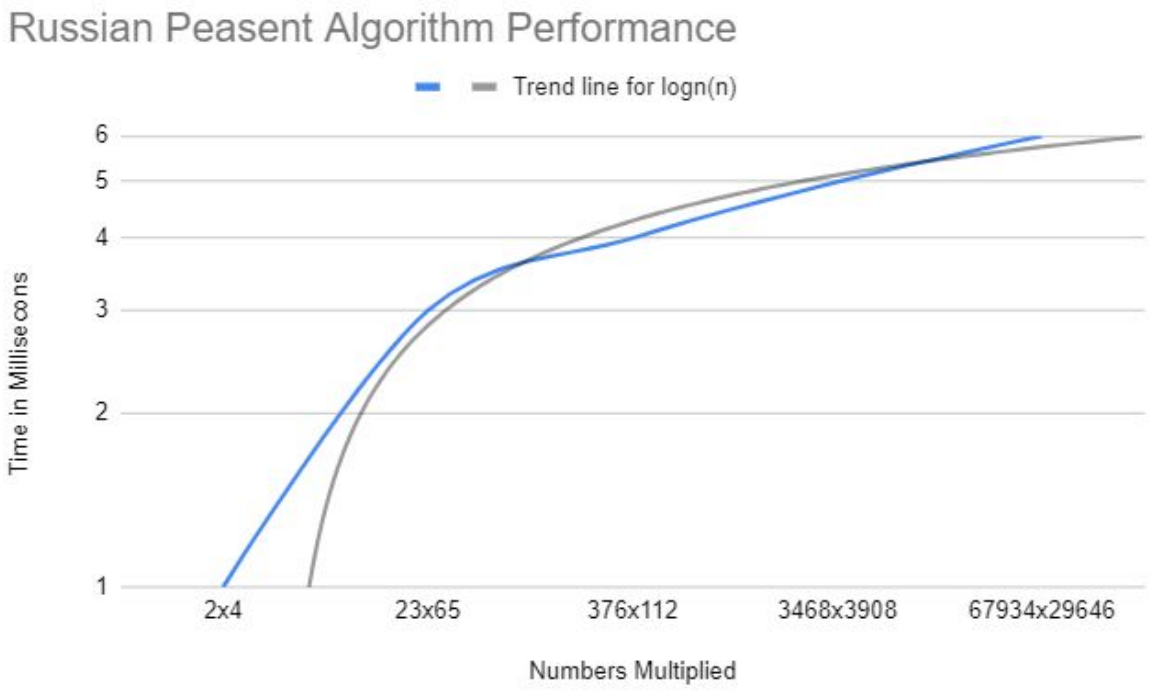
Algorithm Analysis - Adam Russell (18328861)

Link to Google Sheets:

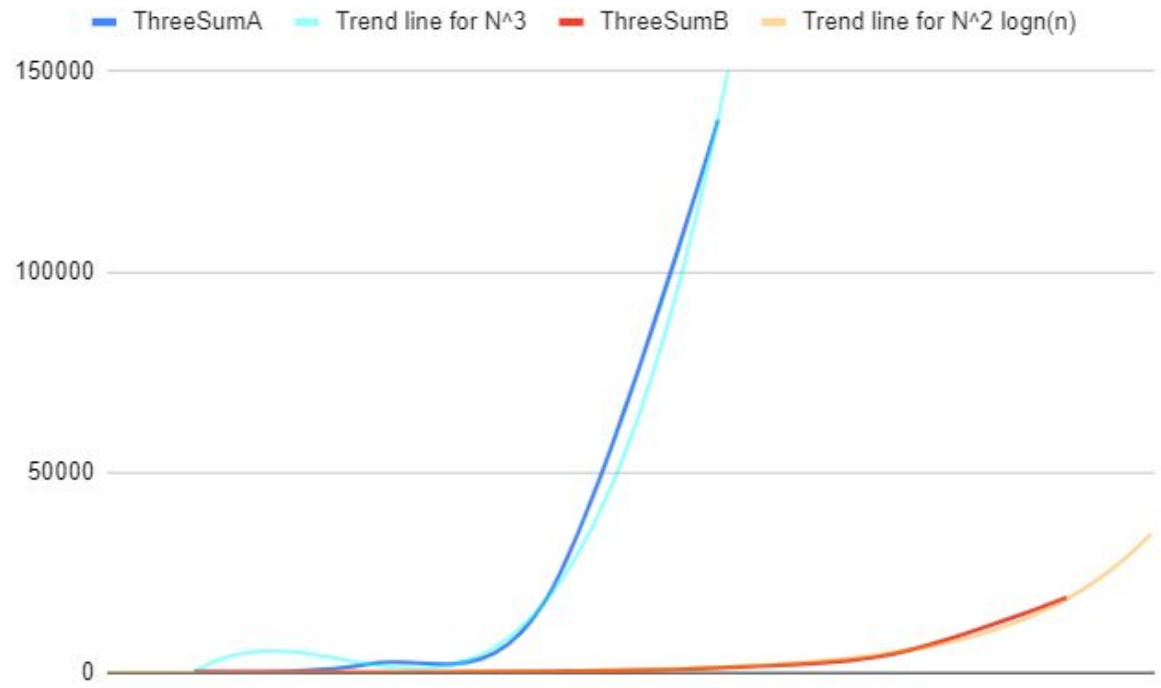
https://docs.google.com/spreadsheets/d/1okFcOSyPnkcgeKQZLfD30blLTSamzbztGlgX-IN5hO8/edit?usp=sharing

Russian Peasant Multiplication Algorithm:



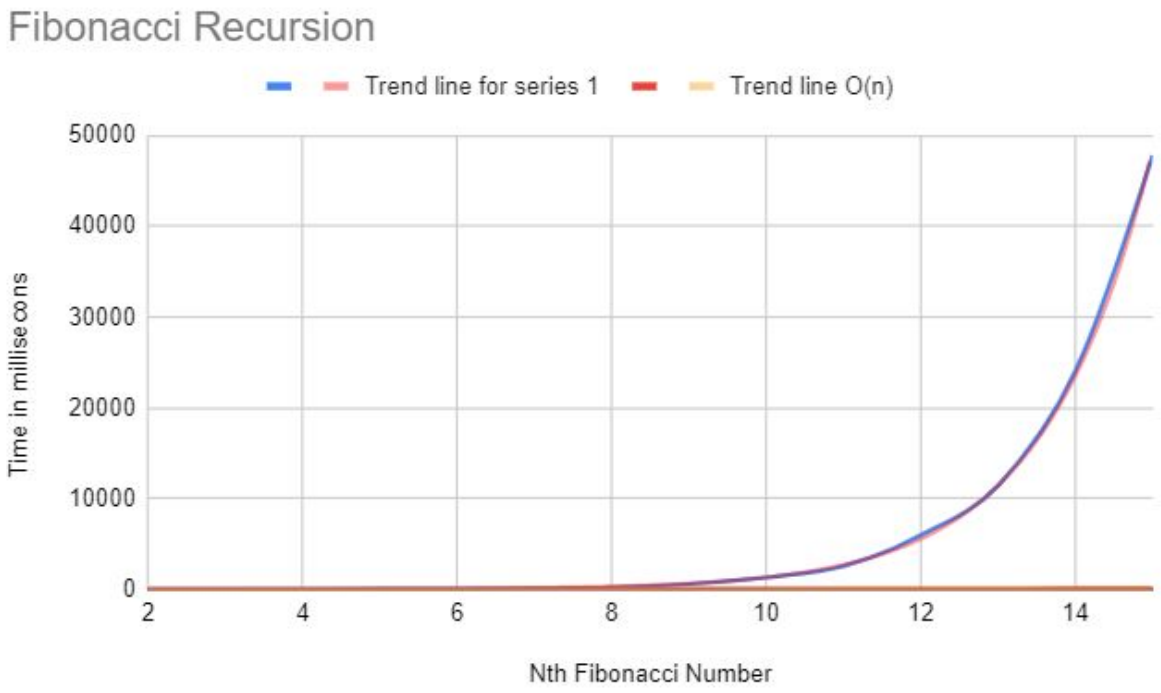
The time complexity of the R.P algorithm is roughly O(logn). The blue line above represents the time's I got through testing alongside a trendline for the log(n). We can see that my times roughly follow the trendline for log(n).

ThreeSumA & ThreeSumB:



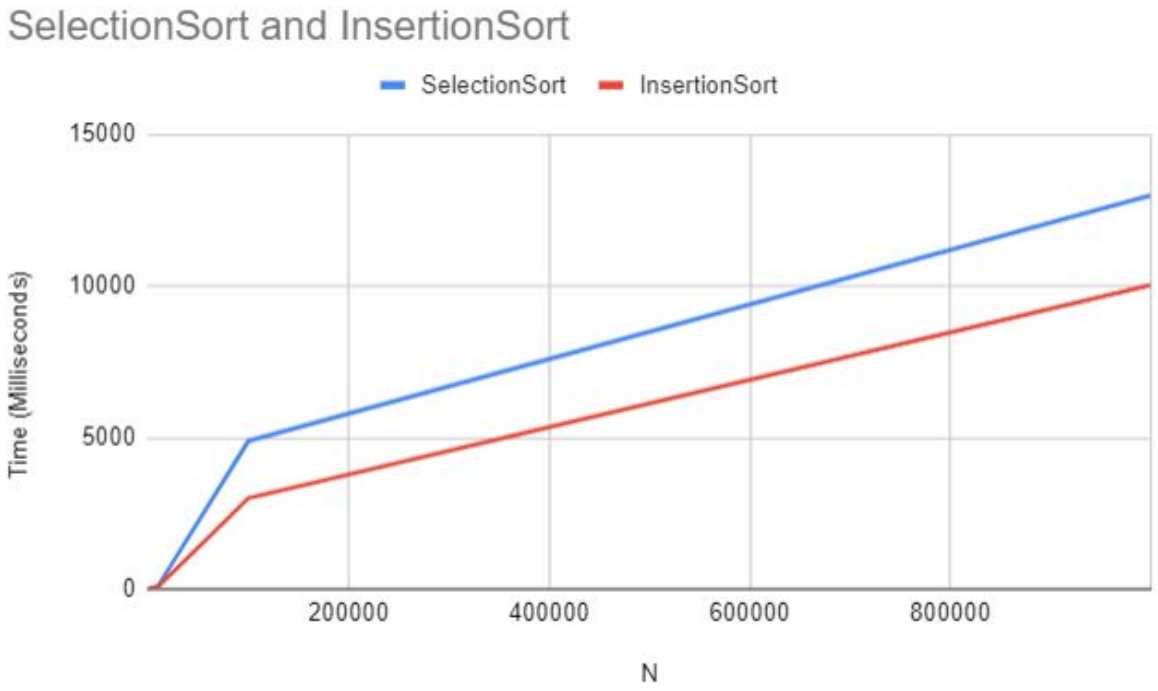
Above is ThreeSumA (O(n^3)) and ThreeSumbB (O(n^2logn)) plotted with the data I got through testing, alongside trendlines for expected outcomes, which match extremely well with the experimental data I received.

Recursion:



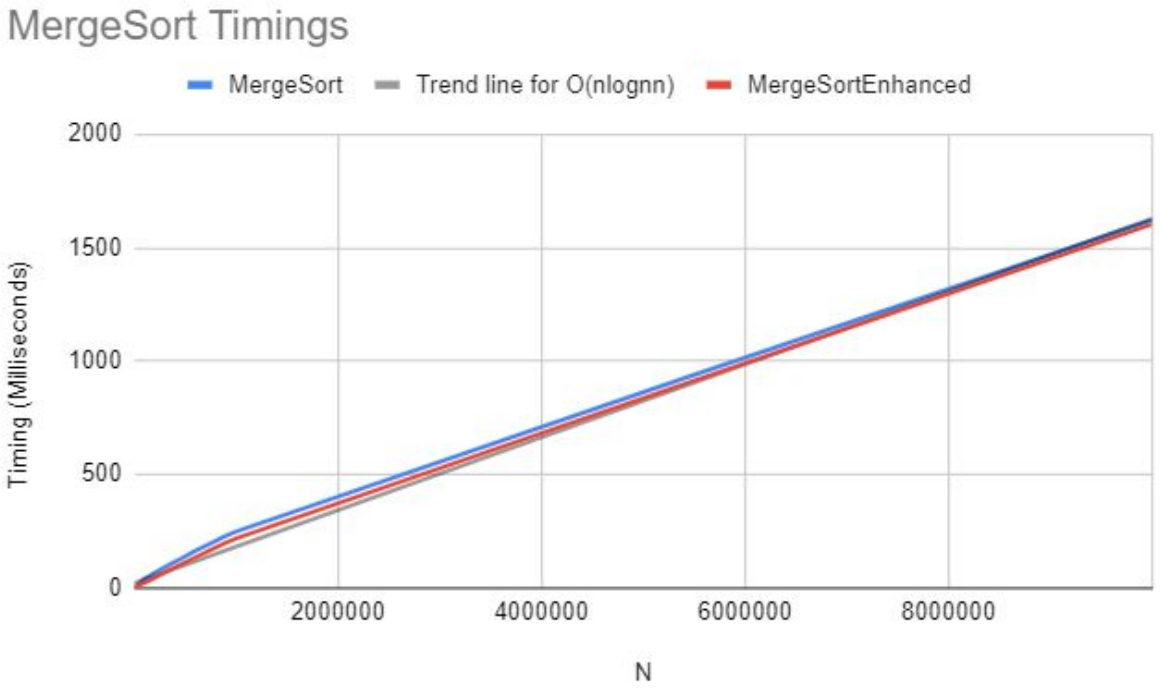
A line graph based on my testing of a recursive and iterative implementation of finding the nth fibonacci number, alongside a trendline for 2^n, showing a good correlation for the recursive implementation, and n (linear) for the iterative implementation. After a certain n (roughly n = 7) the iterative approach begins to outpace the recursive implementation immensely.

Sorting:
Elementary Sorting:



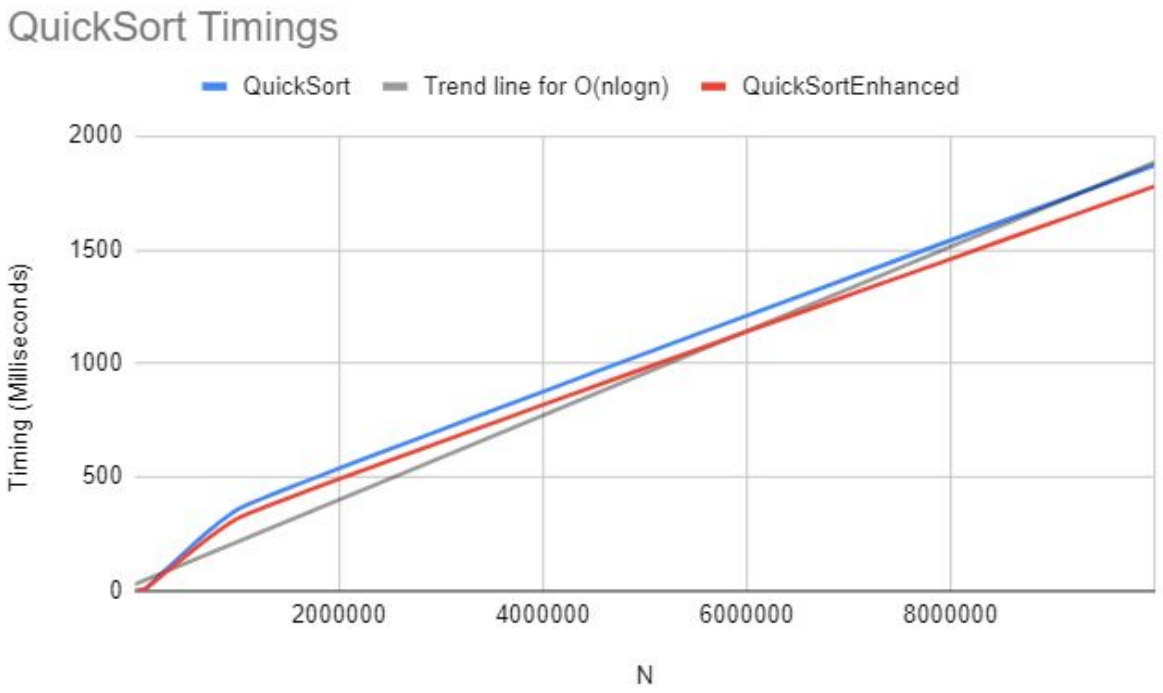
Above is a graph of my times for various arrays of length n using selection sort and insertion sort. The start of the graph is a bit wobbly as for small arrays times will be affected by system noise, but after that it is a straight line, which is what is to be expected. My arrays increased in size by a factor of n^2, and since SelectionSort and InsertionSort both have O(n^2), as straight line is to be expected when graphed using these input sizes. InsertionSort is faster than SelectionSort as seen by my testing. I believe this is because SelectionSorts best case is still O(n^2), while InsertionSorts best case is O(n), meaning InsertionSort can be done faster than O(n^2), which is seen by my graph.
N.B. I did not include my third ridiculous sorting algorithm in the graph as Stalin Sort works in linear time and would have therefore just been a flat line at the bottom of my graph.

MergeSort:



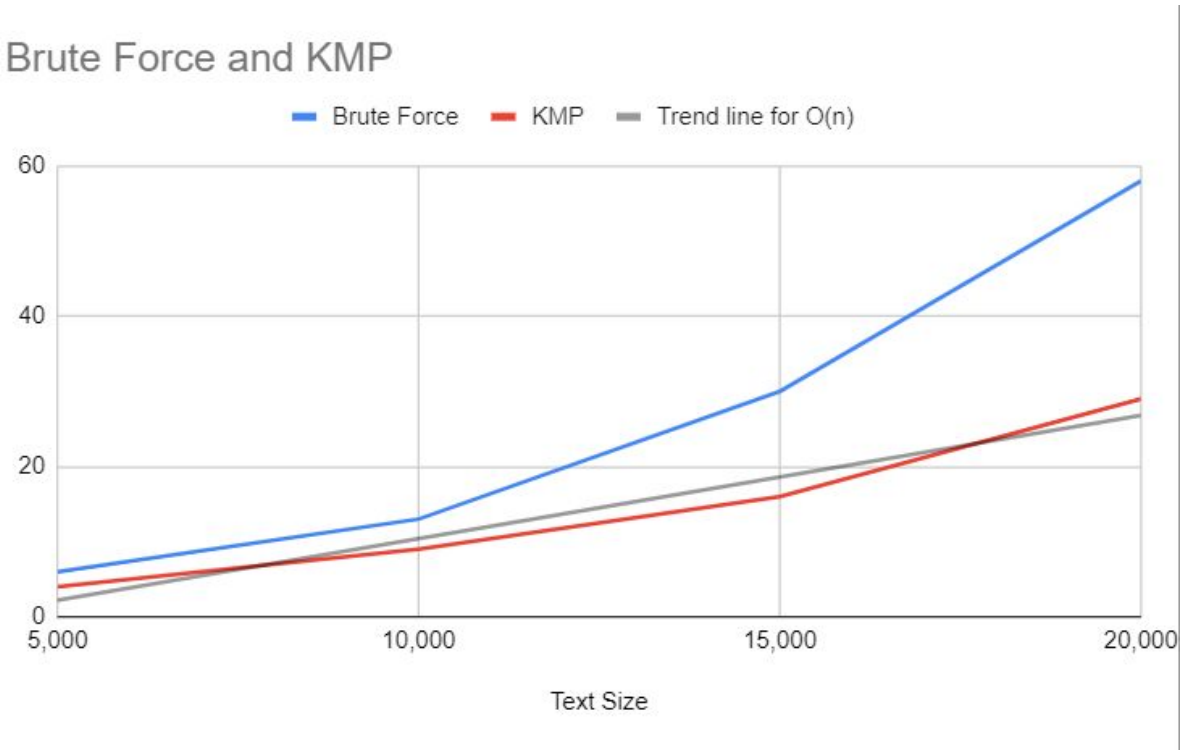
These are my timings for MergeSort and MergeSortEnhanced alongside a trendline for O(nlogn). MergeSortEnhanced is slightly faster than MergeSort (roughy 10%), but as the size of the array to sort gets larger, MergeSort and MergeSortEnhanced get closer timings as the enhancements are effective at the small array stage.

QuickSort:



These are my timings for QuickSort and QuickSortEnhanced alongside a trendline for roughly O(nlogn). QuickSortEnhanced is slightly faster than QuickSort, and they roughly follow the average expected time of O(nlogn).

String Pattern Matching:



These are my timings for string pattern matching using a brute force approach vs. the KMP algorithm, which is also graphed against a trendline for O(n) linear time.