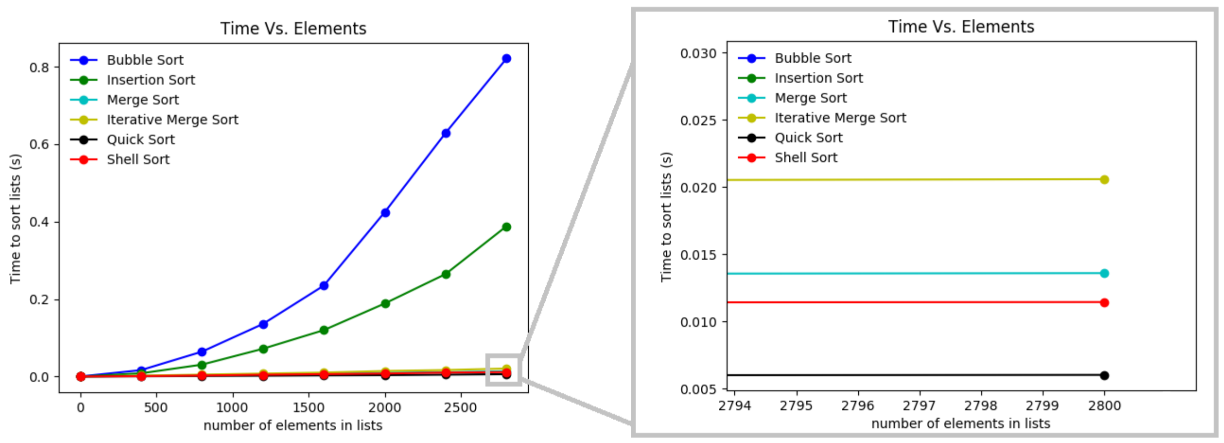

The Graph here compares the different sorting algorithms that were used on assignment 5.

Bubble Sort:

The bubble sort algorithm is one of the least efficient sorting algorithms as it will loop through the whole array almost as many times as the array size and each time it will loop again to compare all indices to one another, this gives the loop the big O notation of O(n^2) for average, and for the best it will be O(n) to run through the array once.

Insertion Sort:

Insertion sort is not as bad as bubble sort as it will run through the array and will check indices, but will replace them if needed, and that might save it some time. However, on average the sorting algorithm still runs the average of O(n^2) with the best case of O(n).

Merge Sort:

The merge sort and the iterative merge sort should be close to running the same run time, they both divide the array down to small pieces and start their way up and comparing the sizes, this cutting down will allow it to run half the time when comparing the elements, giving it the O(n * log n) algorithm for its best, and average test time. The iterative should be faster, due to the fact that it doesn't allocate to that extra space of recalling the merge sort.

Quick Sort:

The Quick Sort algorithm will compare 3 elements at a time at first, and start dividing the work between those 3 to make the work much faster. However, when the list size is small, it will use insertion sort as it will be faster. Its worse case is actually O(n^2) due to that, but on average and on its best, it will run the O(n * log n) time.

Shell Sort:

Shell sort has not been proved its time complexity due to its complex way of sorting. The sort usually starts off by comparing indices by half the size of space between them, and work its way down to each element and find the correct spot for them. This will give it an estimated O(n * log n) ^ 2 on average.