Analysis

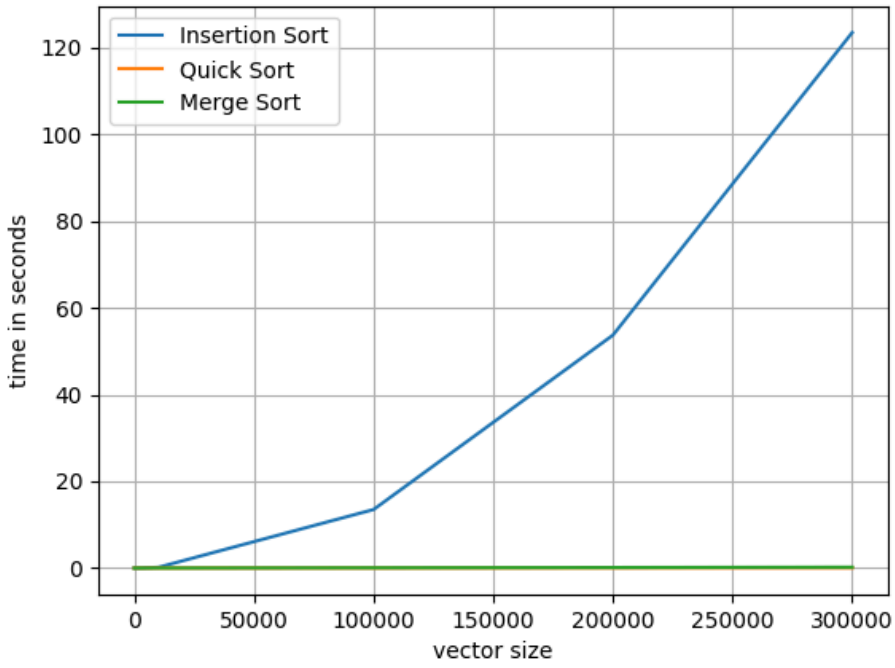
Execution time of the 3 sorting algorithms in an unsorted array shows that insertion sort appears to have an exponential increasing time as size of the array grows, compared to merge sort having a linear growth and quick sort having appearing to have a logarithmic growth. Having a higher level view of the data, insertion sort has the worst time compared to the other 2 algorithms, taking insertion sort 120 seconds to execute an array with 30,000 unsorted elements while merge sort took .2 seconds and quick sort about .3 seconds.

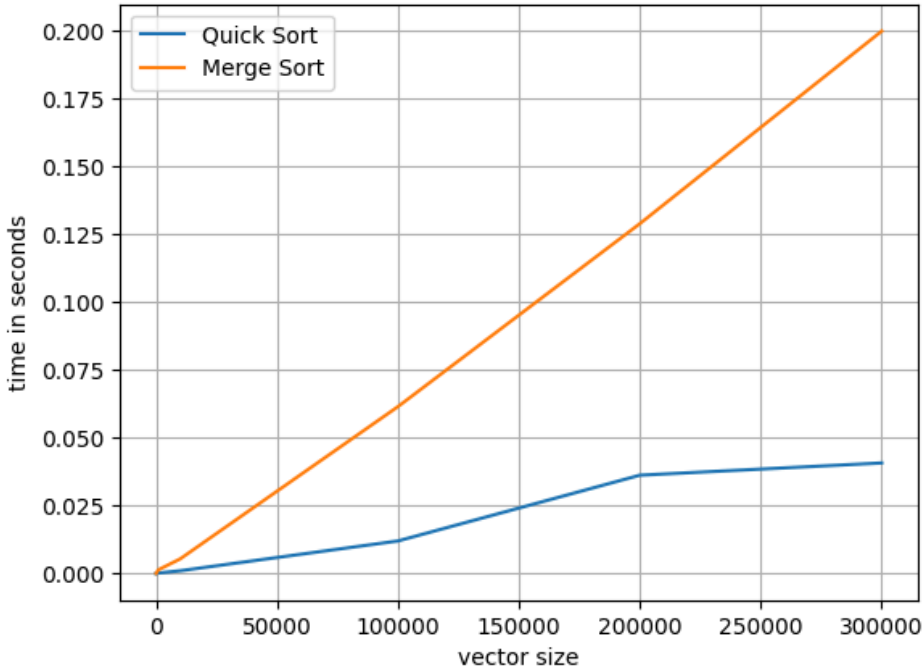
Execution on sorted algorithms shows significant differences. First quick sort shows a high positive linear growth while insertion and merge short have a very low positive growth, both being significantly lower than quick sort. In terms of time, quick sort had the highest time of 20 seconds when executing on 100,000 elements.

Big O notation is apparent in the graphs shown. For insertion sort, the best case is an array of all sorted elements $O(n)$, which is shown by the almost 0 linear line in the graphs of sorted array, and the worst case is an array that has elements sorted in decreasing order $O(n^2)$, which is shown by the exponential line in the graph of unsorted arrays. For merge sort, all cases would result in $O(n\ log\ n)$, which can be shown in both graphs of sorted and unsorted. The graphs show the initial logarithmic growth of this algorithm.For quick sort, the best case is $O(n\ log\ n)$ and the worst case is $O(n^2)$. The logarithmic line is apparent in graphs of sorted arrays, indicating that this coil is an average case.

In terms of space complexity, insertion sort has the most efficient complexity of $O(1)$, since all operations are in place. Quick sort has a space complexity of $O(n)$ in the worst case and $O(n\ log\ n)$ in the best and average case. This is due to the divide of the arrays, making space complexity logarithmic, but in the worst case where the array is sorted in decreasing order, every element will be divided into parts containing 1 element, requiring there to be a memory space of n for the array. Although in the worst case there requires a memory location the same size of the array, there will be no extra space needed for anything else, making this another memory efficient algorithm. For merge sort, the space complexity is $O(n)$. This is because each recursion of the sorting algorithm creates a temporary array, which by summing up the size of all the temporary arrays results in n size.

Plots of unsorted arrays





Plots of sorted arrays