
Winter 2021

Programming Assignment 3-CIS3501

12/7/21

Nico Bokhari

UMID: 38284715

College of Engineering & Computer Science

University of Michigan - Dearborn, Dearborn, MI 48128

I. INTRODUCTION

In this assignment, I utilized an array-based implementation of Merge Sort, Deterministic Quick Sort, and Randomized Quick Sort. All the algorithms are in a recursive implementation.

II. PROGRAM ANALYSIS

A. Time & Space Complexities of Implemented Sorting Algorithms

Methods	Worst Case Time	Worst Case Space
MergeSort()	$O(n \log n)$	$O(n)$
merge()	$O(n)$	$O(n)$
detQuickSort()	$O(n^2)$	$O(n)$
randQuickSort()	$O(n^2)$	$O(n)$
swap()	$O(1)$	$O(1)$

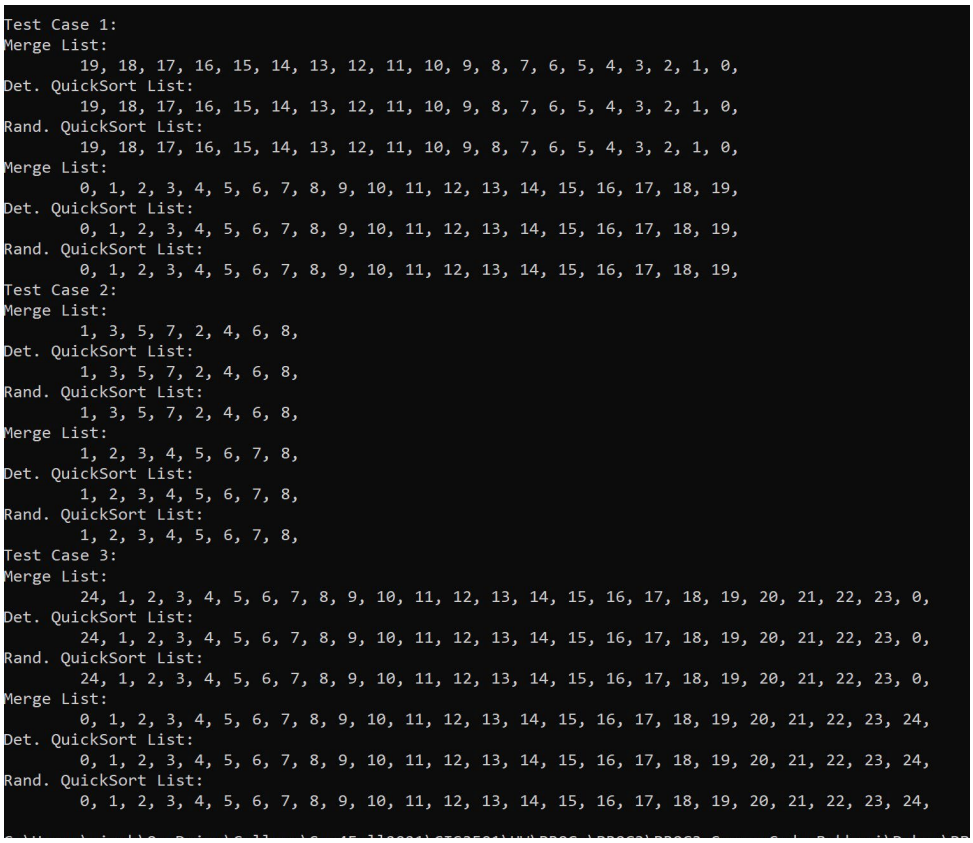
B. Pre & Post Conditions of Algorithm Methods

Method	Pre-condition	Post-condition
sortingLists::mergeSort	The list is non-empty with at least 1 element.	The list is sorted in ascending order.
sortingLists::merge	The list is non-empty with at least 1 element, and its partitions are sorted in ascending order.	The list's partitions are combined such that they are placed in ascending order
sortingLists::detQuickSort	The list is non-empty.	The list is sorted such that all elements less than the pivot(set to the last element) are in the beginning of the list, and numbers greater are in the end of the list.
sortingLists::randQuickSort	The list is non-empty	The list is sorted such that all numbers less than some random pivot are in the beginning of the list, and numbers greater are in the end of the list.

C. Test Cases

Input	***Expected Output***	***Actual Output***	***Success/Fail?***
{19, 18, 17…..2,1,0} //reverse order	{0,1,2,…..18,19} for all algorithms	{0,1,2,…..18,19}	Success
{1,3,5,7,2,4,6,8} //maximum number of comparisons for MergeSort	{1,2,3,4,5,6,7,8} for all algorithms	{1,2,3,4,5,6,7,8}	Success
{24, 1, 2, 3, ….. 22, 23, 0} //first and last elements swapped	{0,1,2,3,…..23,24} for all algorithms	{0,1,2,3,…..23,24}	Success

Test Cases Raw Output



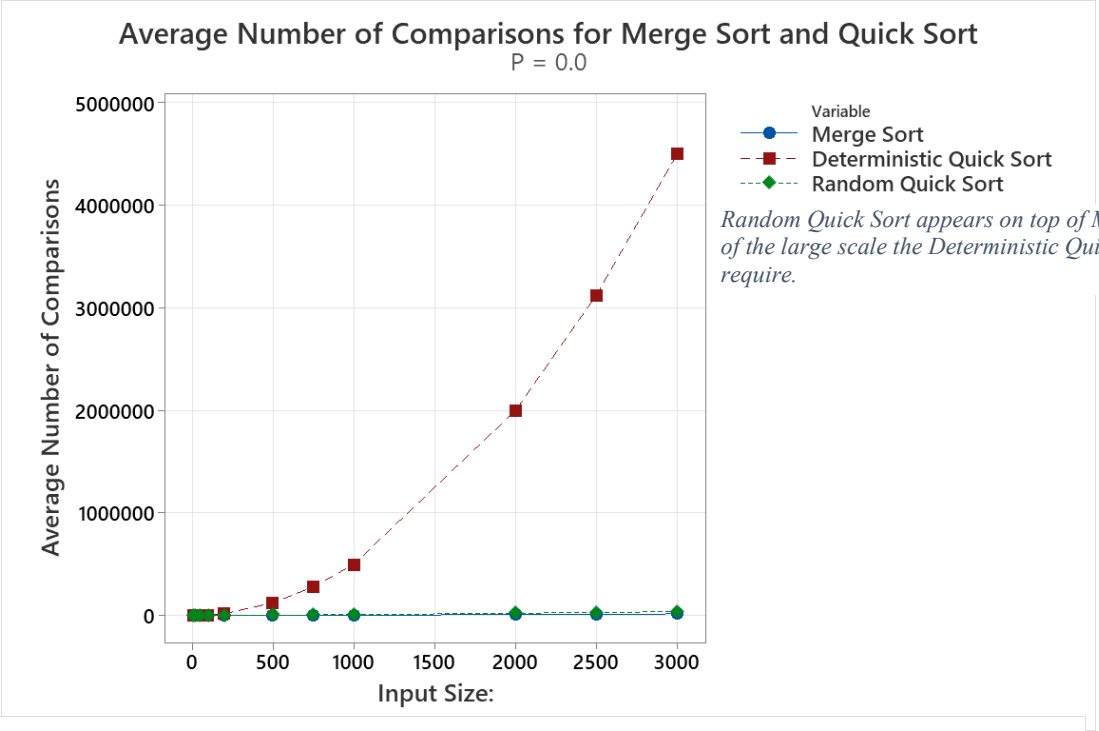
```
Test Case 1:
Merge List:
        19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0,
Det. QuickSort List:
        19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0,
Rand. QuickSort List:
        19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0,
Merge List:
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
Det. QuickSort List:
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
Rand. QuickSort List:
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
Test Case 2:
Merge List:
        1, 3, 5, 7, 2, 4, 6, 8,
Det. QuickSort List:
        1, 3, 5, 7, 2, 4, 6, 8,
Rand. QuickSort List:
        1, 3, 5, 7, 2, 4, 6, 8,
Merge List:
        1, 2, 3, 4, 5, 6, 7, 8,
Det. QuickSort List:
        1, 2, 3, 4, 5, 6, 7, 8,
Rand. QuickSort List:
        1, 2, 3, 4, 5, 6, 7, 8,
Test Case 3:
Merge List:
        24, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 0,
Det. QuickSort List:
        24, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 0,
Rand. QuickSort List:
        24, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 0,
Merge List:
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
Det. QuickSort List:
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
Rand. QuickSort List:
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
```

D. Experimental Comparison of Sorting Algorithms

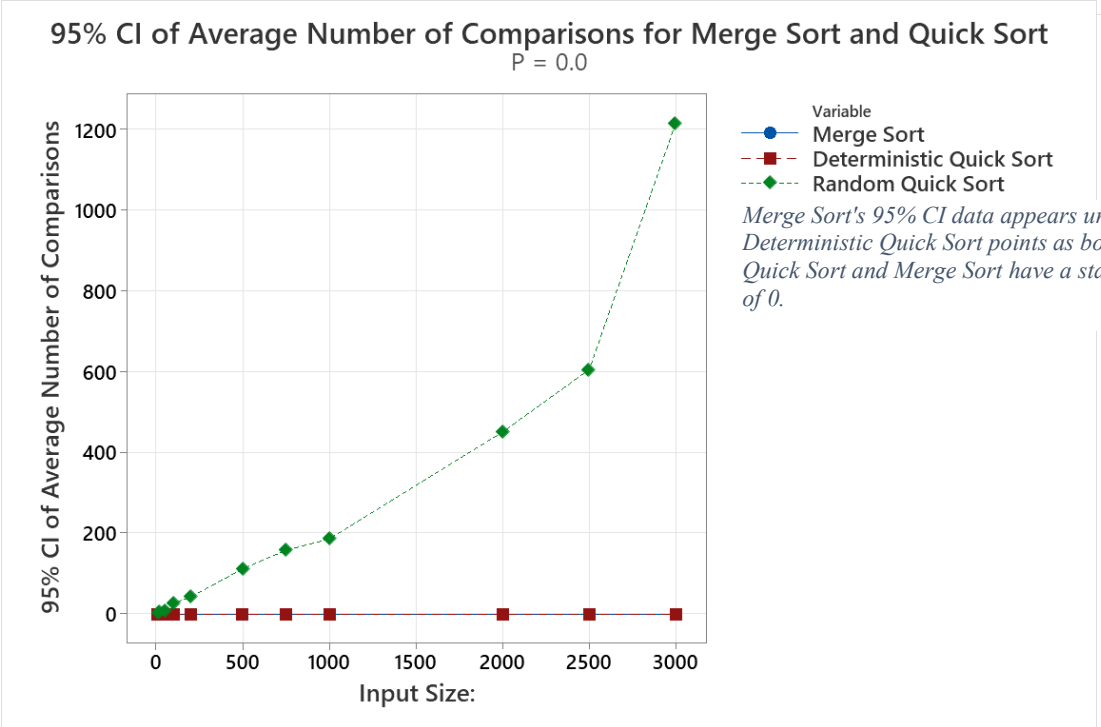
Each p-value designates how “random” or “unsorted” each list is before each sorting algorithm is applied. Three identical lists were created with the same randomization swaps and subjected to a different algorithm. For instance, if the list = {4,2,1,0,3}, one copy would only be subjected to Merge Sort methods, another copy only for Deterministic Quick Sort, and so on. With the size of the list(n) varying, each n and p pair ran for 25 instances; both the CPU time(milliseconds) and number of comparisons performed in each algorithm are calculated, averaged, and plotted over list size; additionally, the 95% Confidence Interval of the mean for number of comparisons and CPU time over 25 instances was calculated for each n, and each sorting algorithm. ***Lines on the following scatterplots are not trendlines as the graphing software cannot show regressions for logarithmic relationships.***

a. P = 0.0

i. Average Number of Comparisons

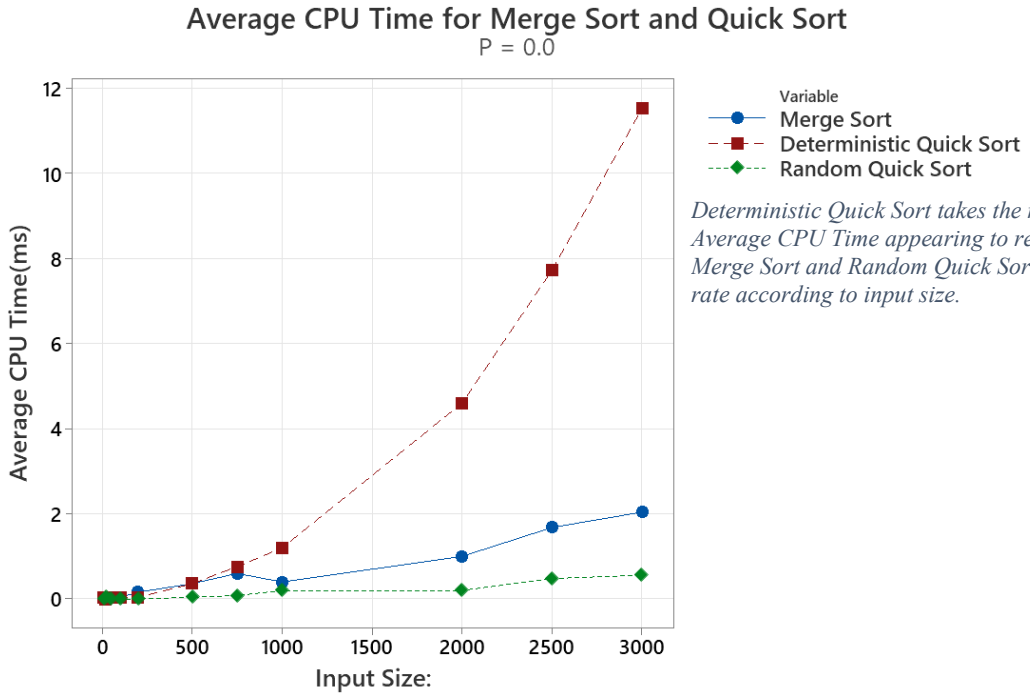


Random Quick Sort appears on top of Merge Sort because of the large scale the Deterministic Quick Sort values require.

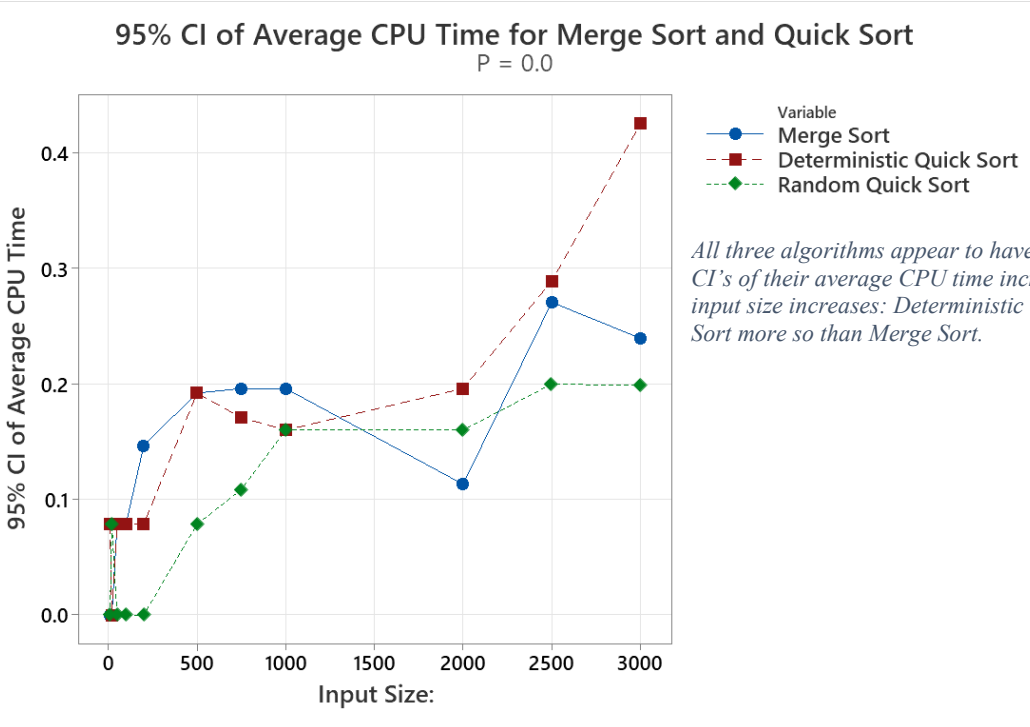


Merge Sort's 95% CI data appears under the Deterministic Quick Sort points as both Deterministic Quick Sort and Merge Sort have a standard deviation of 0.

ii. Average CPU Time (ms)



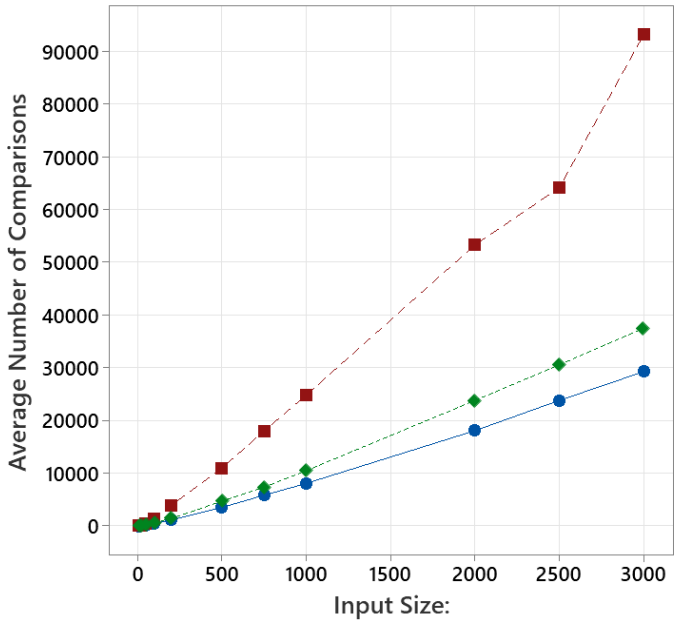
Deterministic Quick Sort takes the most time with Average CPU Time appearing to regress quadratic. Merge Sort and Random Quick Sort regress at a slower rate according to input size.



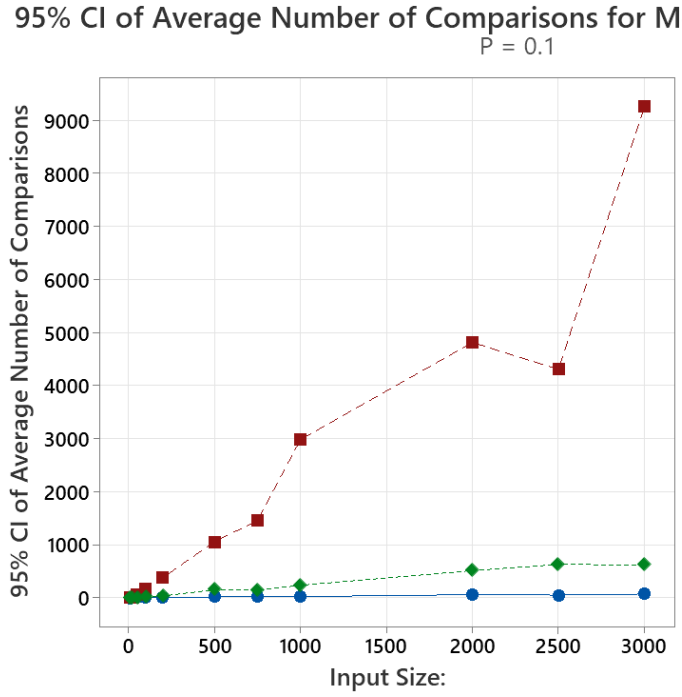
All three algorithms appear to have 95 % CI's of their average CPU time increase as input size increases: Deterministic Quick Sort more so than Merge Sort.

b. P = 0.1

i. Average Number of Comparisons

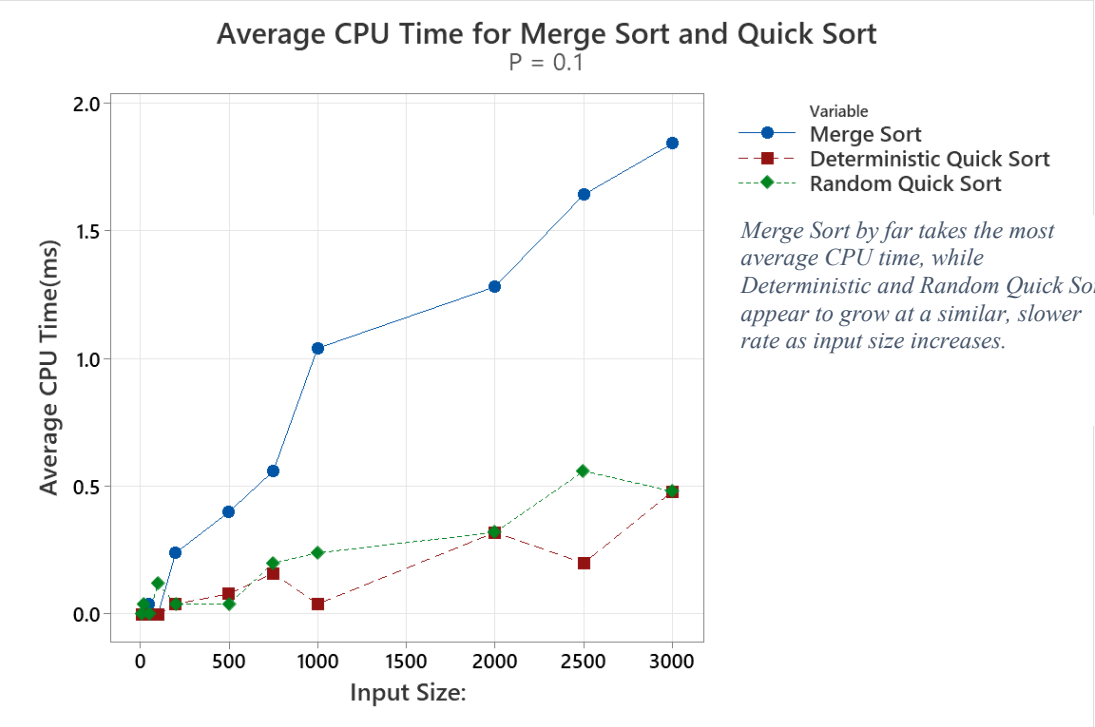


Deterministic Quick Sort appears to dominate in the number of comparisons as input size increases, followed by Random Quick Sort, then Merge Sort.

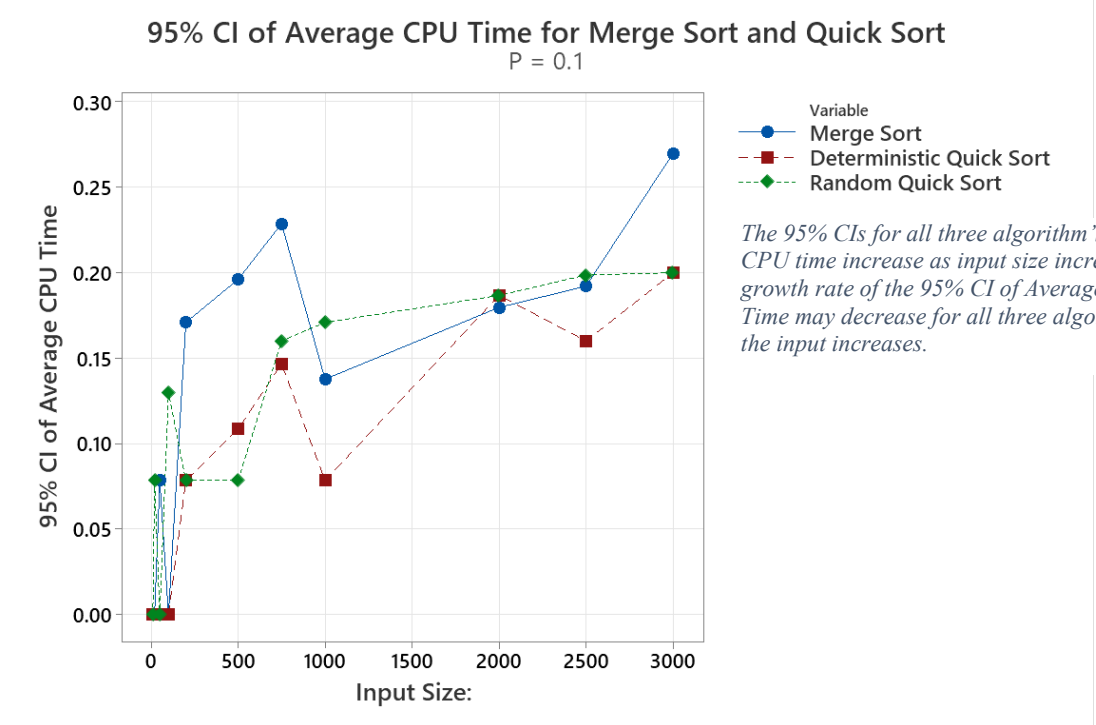


Merge Sort has the lowest 95% CI for its average number of comparisons as input size increases. Random Quick Sort increases faster than Merge Sort, but it may have a similar growth to Merge Sort. Deterministic Quick Sort has the fastest growth rate.

ii. Average CPU Time (ms)



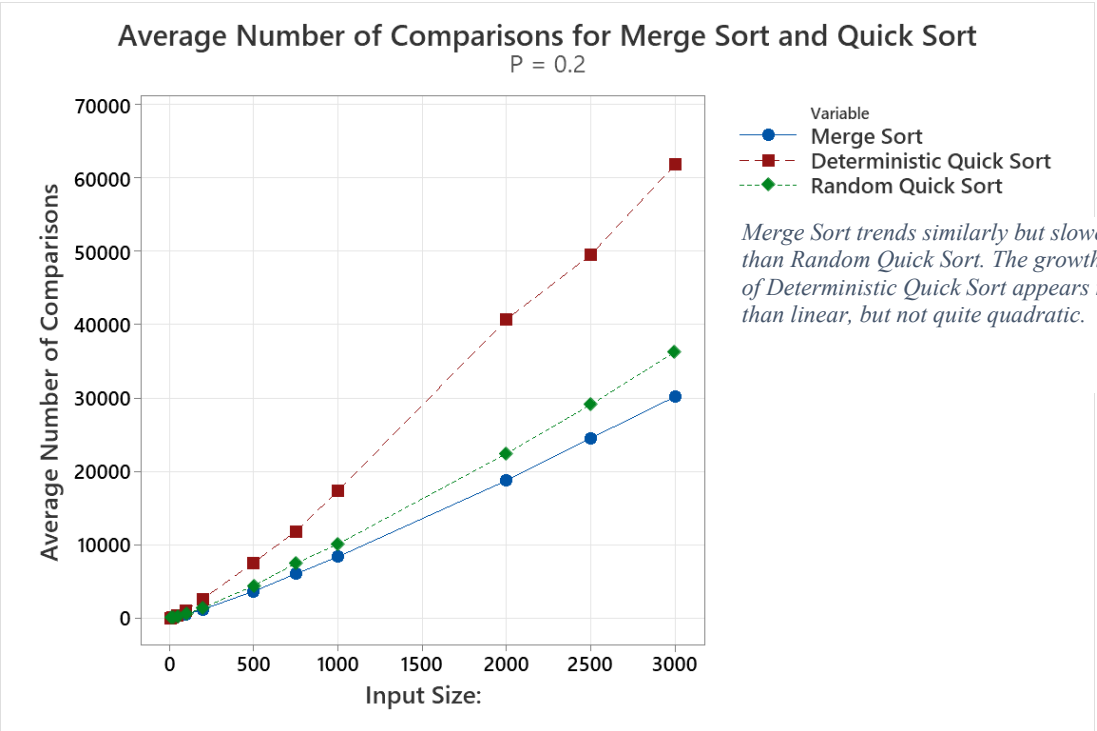
Merge Sort by far takes the most average CPU time, while Deterministic and Random Quick Sort appear to grow at a similar, slower rate as input size increases.



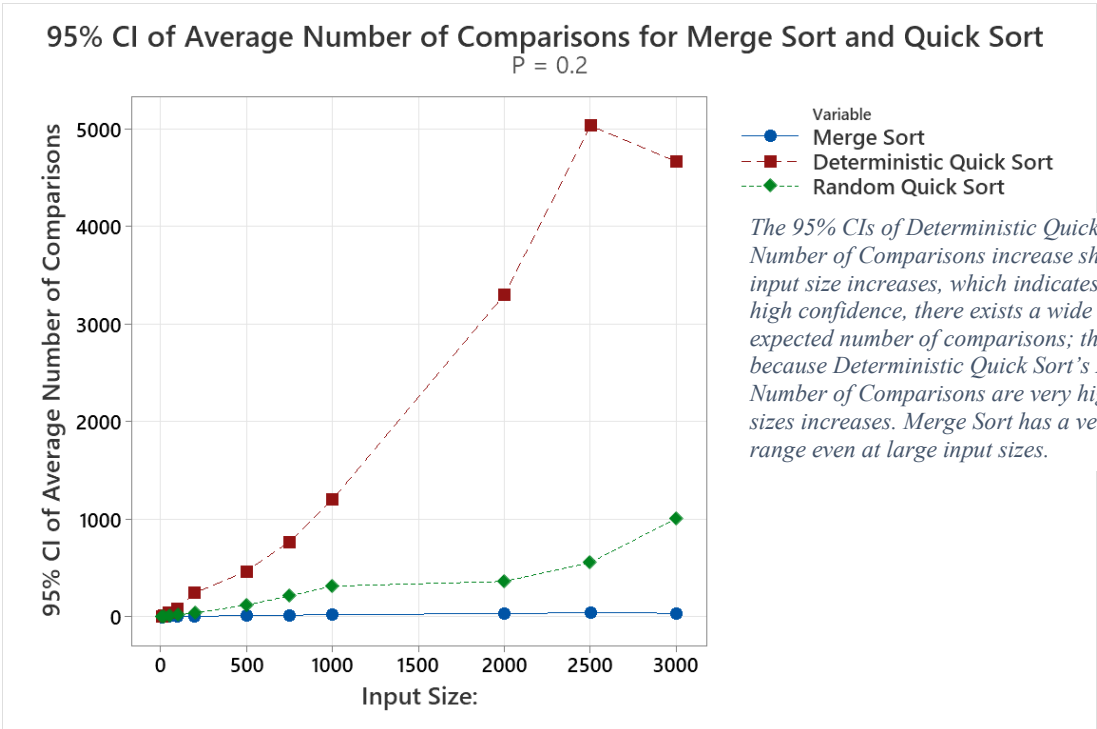
The 95% CIs for all three algorithm's average CPU time increase as input size increases. The growth rate of the 95% CI of Average CPU Time may decrease for all three algorithms as the input increases.

c. P = 0.2

i. Average Number of Comparisons

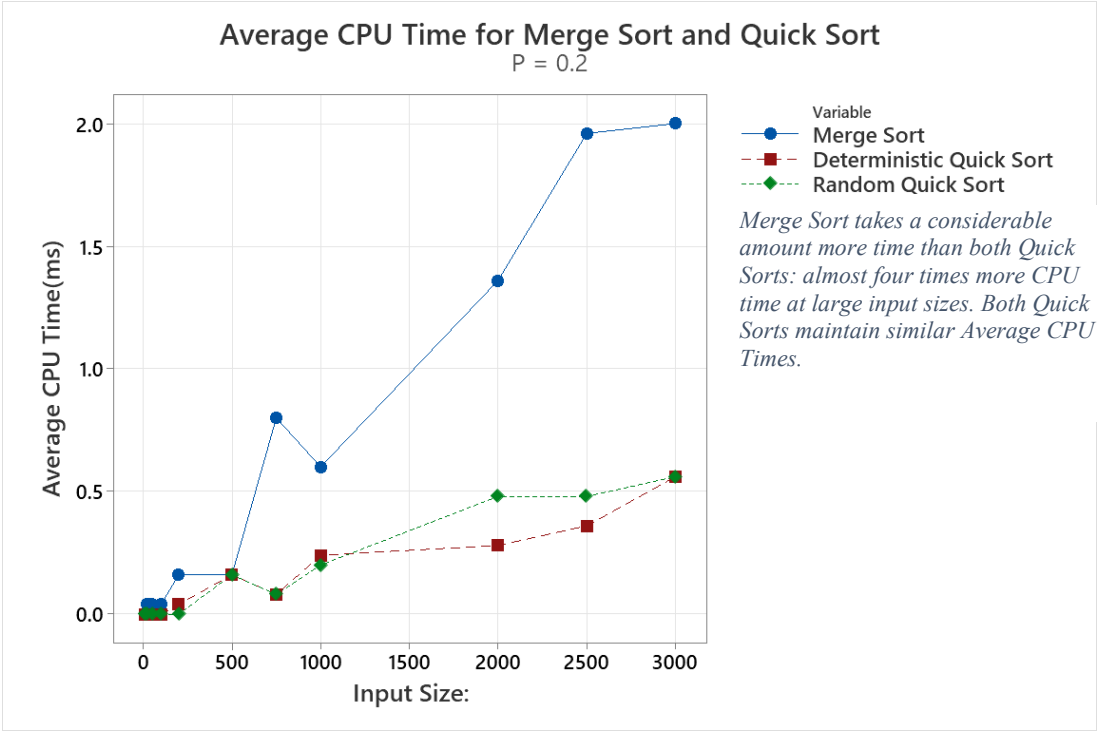


Merge Sort trends similarly but slower than Random Quick Sort. The growth rate of Deterministic Quick Sort appears more than linear, but not quite quadratic.

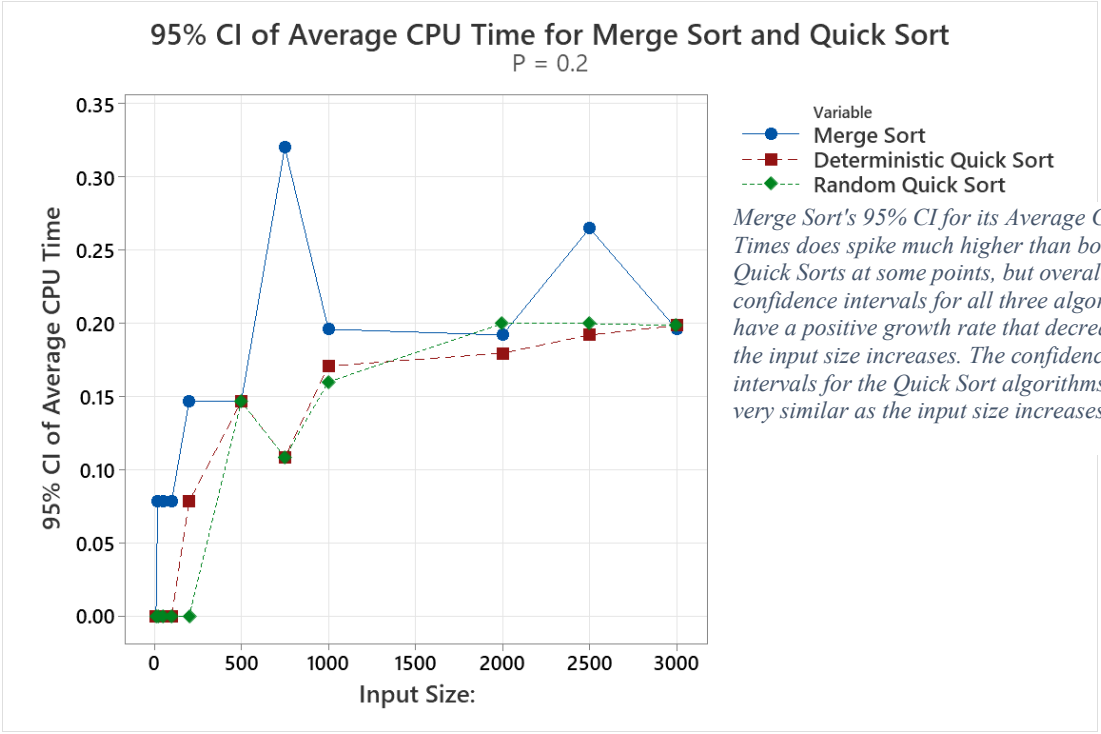


The 95% CIs of Deterministic Quick Sort's Average Number of Comparisons increase sharply as the input size increases, which indicates that despite a high confidence, there exists a wide range for the expected number of comparisons; this may occur because Deterministic Quick Sort's Average Number of Comparisons are very high as input sizes increases. Merge Sort has a very small CI range even at large input sizes.

ii. Average CPU Time (ms)



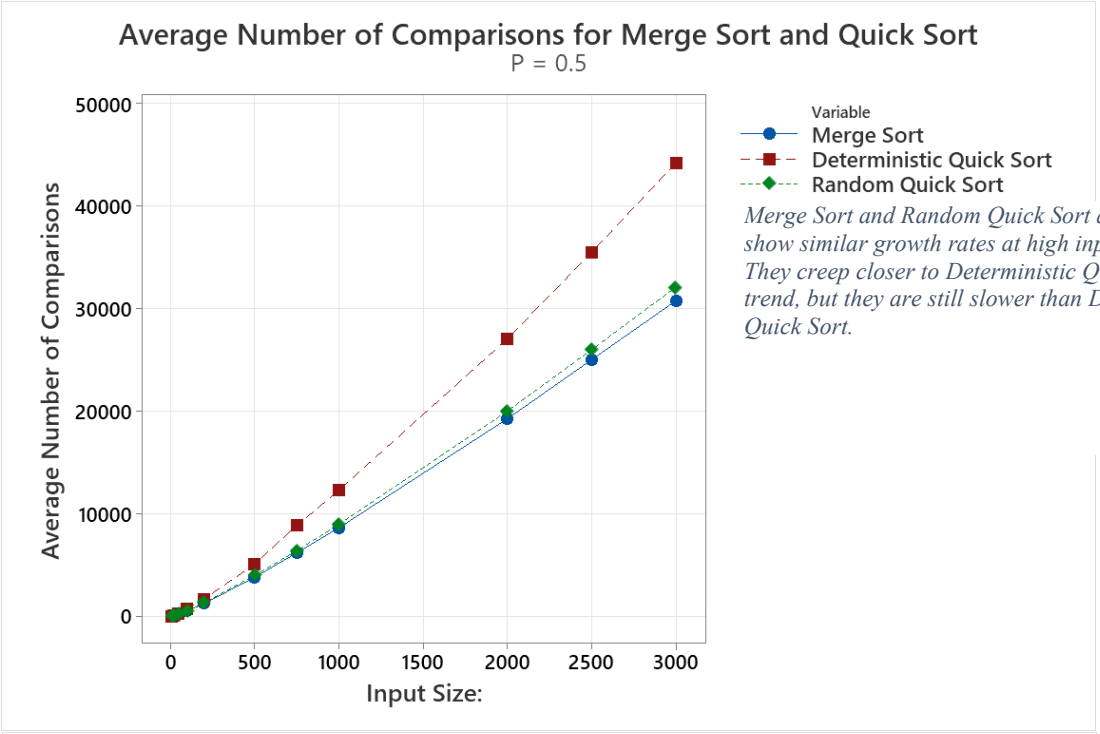
Merge Sort takes a considerable amount more time than both Quick Sorts: almost four times more CPU time at large input sizes. Both Quick Sorts maintain similar Average CPU Times.



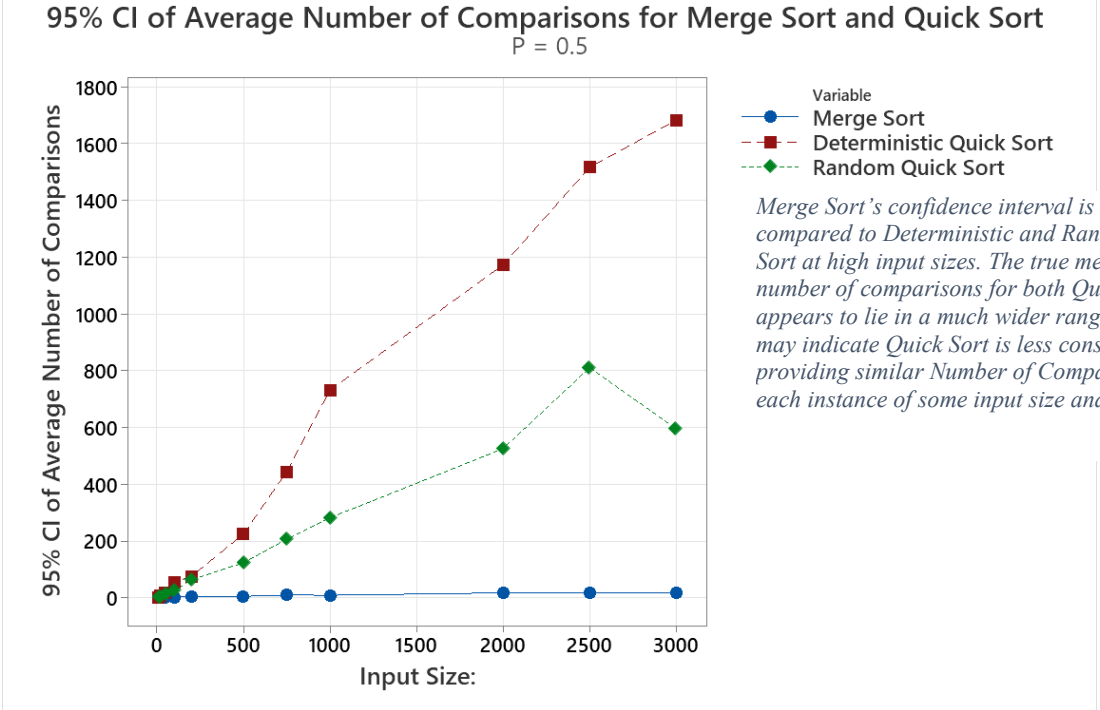
Merge Sort's 95% CI for its Average CPU Times does spike much higher than both Quick Sorts at some points, but overall, the confidence intervals for all three algorithms have a positive growth rate that decreases as the input size increases. The confidence intervals for the Quick Sort algorithms are very similar as the input size increases.

d. P = 0.5

i. Average Number of Comparisons

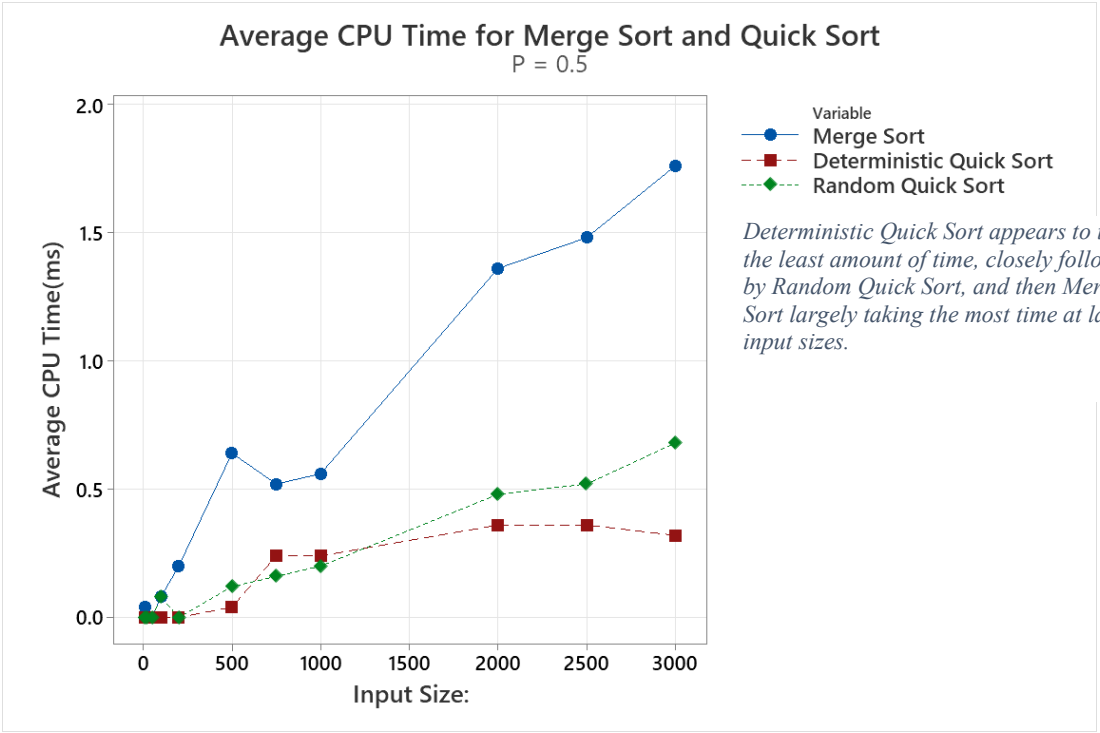


Merge Sort and Random Quick Sort appear to show similar growth rates at high input sizes. They creep closer to Deterministic Quick Sort's trend, but they are still slower than Deterministic Quick Sort.

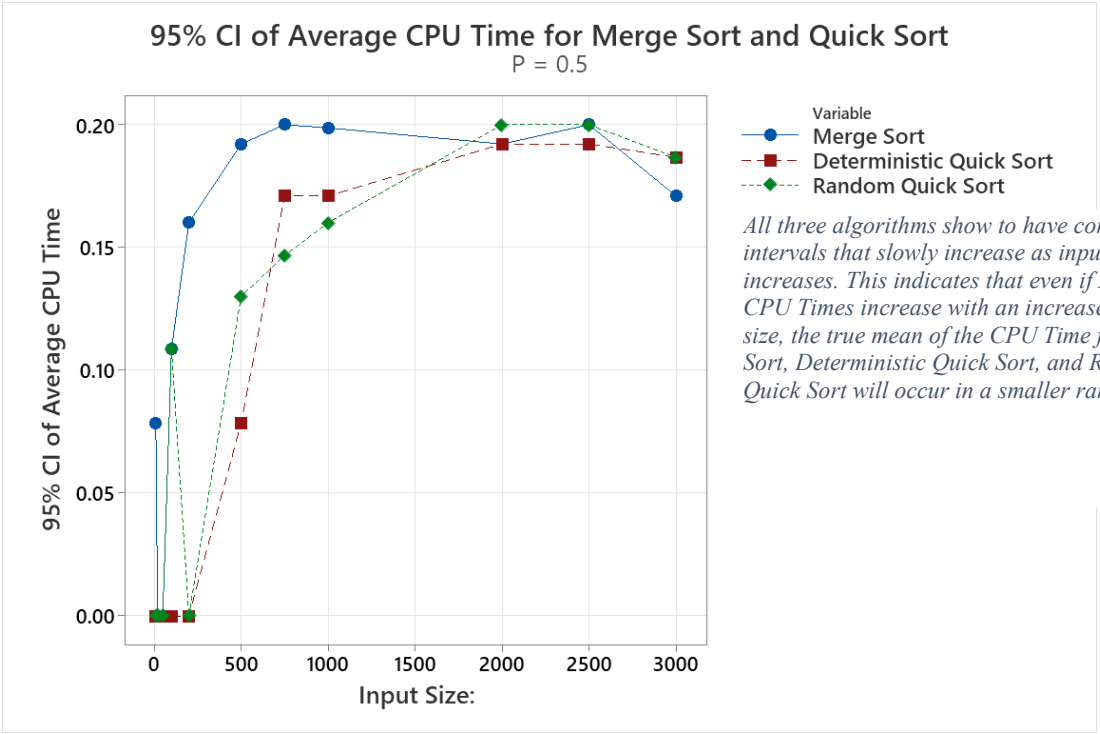


Merge Sort's confidence interval is very short compared to Deterministic and Random Quick Sort at high input sizes. The true mean of the number of comparisons for both Quick Sorts appears to lie in a much wider range; this may indicate Quick Sort is less consistent in providing similar Number of Comparisons in each instance of some input size and P = 0.5.

ii. Average CPU Time (ms)



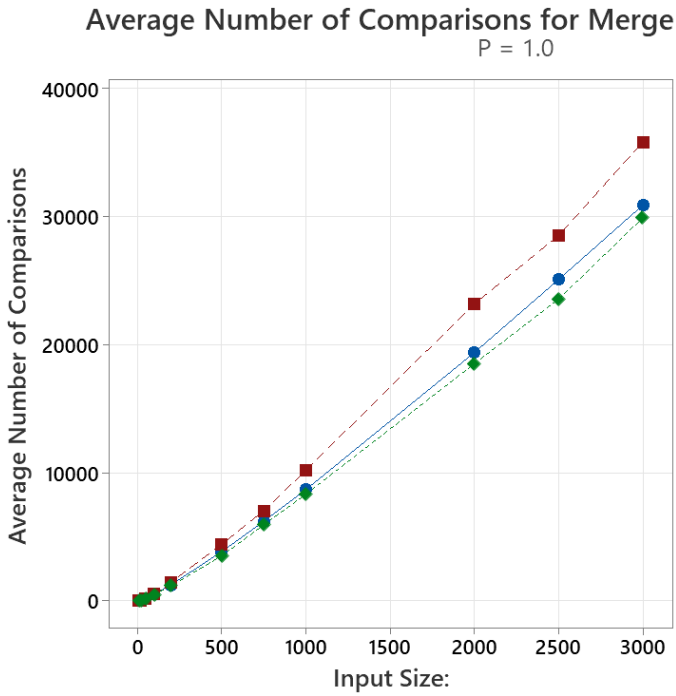
Deterministic Quick Sort appears to take the least amount of time, closely followed by Random Quick Sort, and then Merge Sort largely taking the most time at large input sizes.



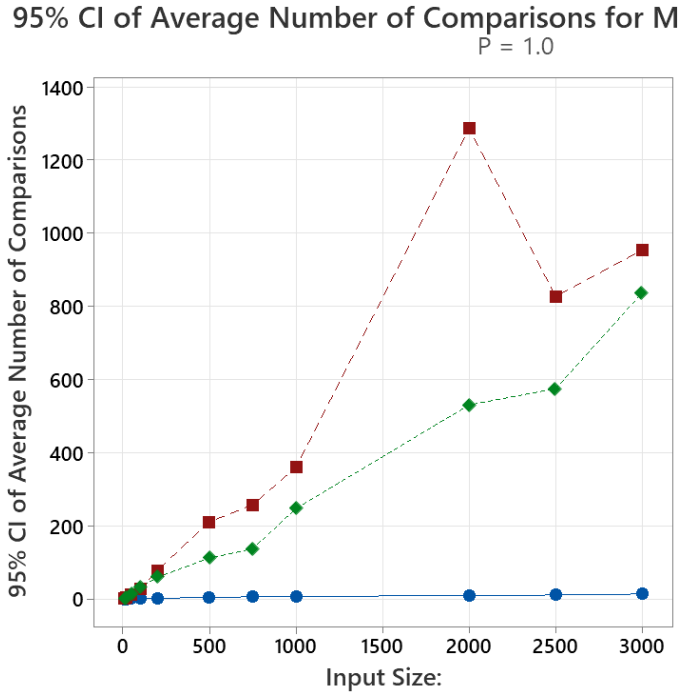
All three algorithms show to have confidence intervals that slowly increase as input size increases. This indicates that even if Average CPU Times increase with an increase in input size, the true mean of the CPU Time for Merge Sort, Deterministic Quick Sort, and Random Quick Sort will occur in a smaller range.

e. P = 1.0

i. Average Number of Comparisons

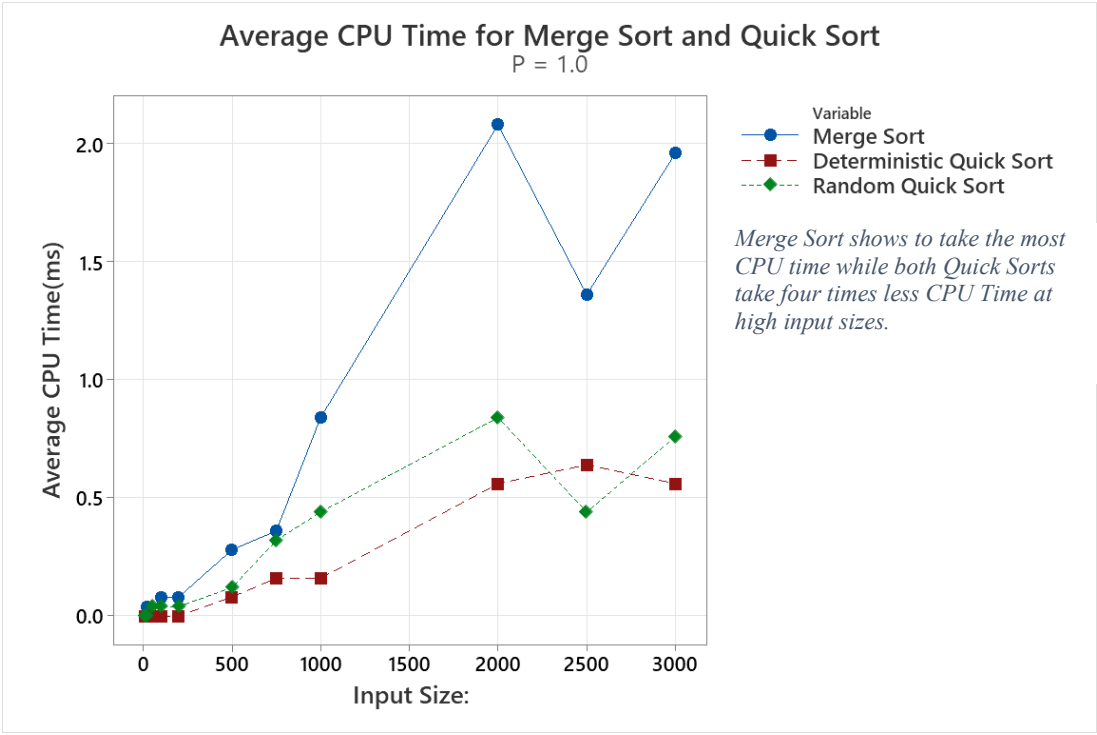


Random Quick Sort barely beats Merge Sort with the smallest Average Number of Comparisons as the input size increases. Deterministic Quick Sort may have a similar growth to Merge Sort and Random Quick Sort

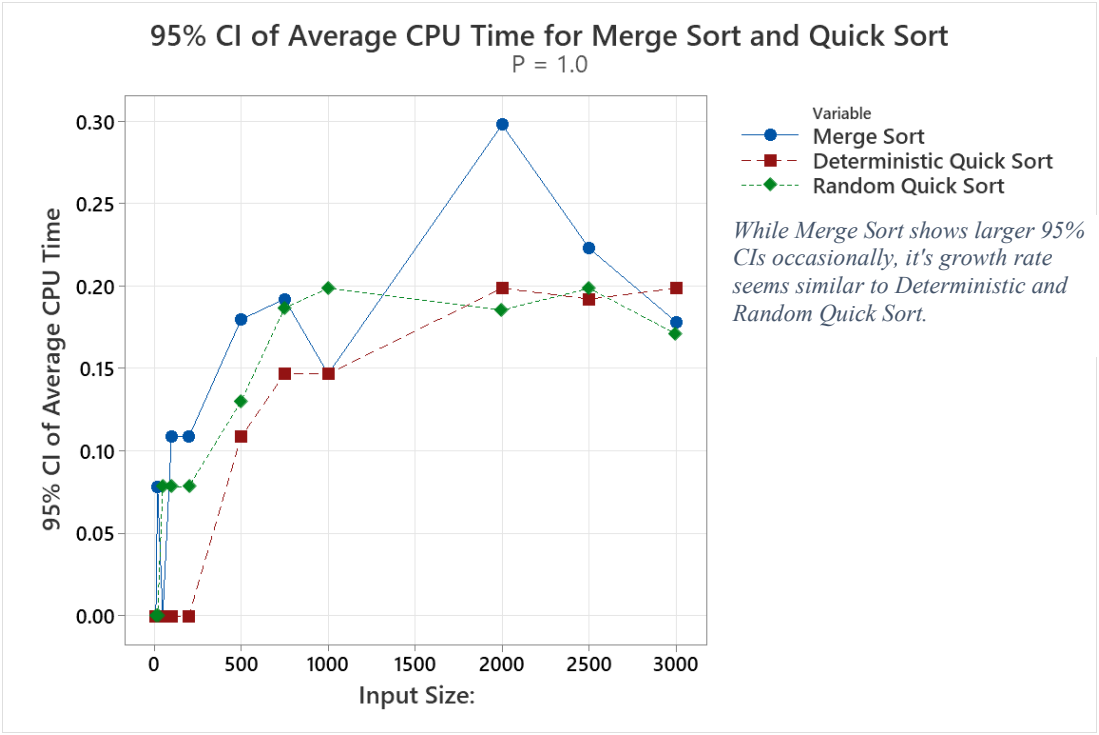


Deterministic and Random Quick Sort seem to tend toward larger confidence intervals as input size increases. Merge Sort is very consistent with its Average Number of Comparisons for every instance, which may be why its 95% CI doesn't increase much with the input size.

ii. Average CPU Time (ms)



Merge Sort shows to take the most CPU time while both Quick Sorts take four times less CPU Time at high input sizes.



While Merge Sort shows larger 95% CIs occasionally, it's growth rate seems similar to Deterministic and Random Quick Sort.