**Program3 Report**

Jun Hyung Park

**This Report Assumes**

* Recursive Merge sort as Usual Merge sort
* Iterative Merge sort as Improved Merge sort
* This report also used data size of 20 to 7000 increment 20 every time.
* Analysis Algorithm, the report will be using the data size 20 to 7000 to prove better time complexity.
* Analysis Algorithm, the repost will be using the data size 20 to 16700 to prove better time complexity.

**Purpose:**

Although, recursive merge sort and quick sort has same execution complexity O(nlog(n)), the actual performance is much slower. The purpose of this report is to analyze the running time of three algorithms, which are usual merge sort or merge sort implemented using recursion, quick sort, and improved merge sort, which use iterative instead of recursion by getting ratio by using each algorithms’ upper bound and lower bound.

**Pre-condition:**

The program generates size of 20 to 1000 elements, the size of element will increase by 20 every time, (20,40,60 …. 1000). For every size, it will contain random numbers as its elements and those elements will be store in the vector call items. Since the purpose of this report is to compare the running time of three sorting algorithms, copy the elements of vector items to vector items2 and items3, so, item, item2, and item3 contains same random value, which makes precise pre-condition to check running time.

**To place data to excel:**

Execute three sorting algorithms (merge sort, improved merge sort, and quick sort), and measure the start time and end time until the random elements are fully sorted by using elapsed function of initial size(20). Write down the data to “outfile”, which will generate “compare.txt”, and it will repeat the same process till size is equal to 1000.

**Hypothesis:**

The Big-O notation for two algorithms (recursive merge sort, quick sort) is O(nlog(n)), however, I did learn that merge sort will likely to take more time, since it call out itself every time and when it got call, it creates temporary array every time. So, on recursive merge sort is slower than quick sort. For the improved merge sort should be faster than recursive merge sort, since it is iterative and use only one temporary vector.

**Verify my Improved Merge sort’s correctness**

Data 1: Output of your improved merge sort program (when #items = 30)

jhpp114@LAPTOP-LDEH7SV5:/mnt/c/Users/jhpp1/Desktop/Program3\_Test$ ./a.out 30



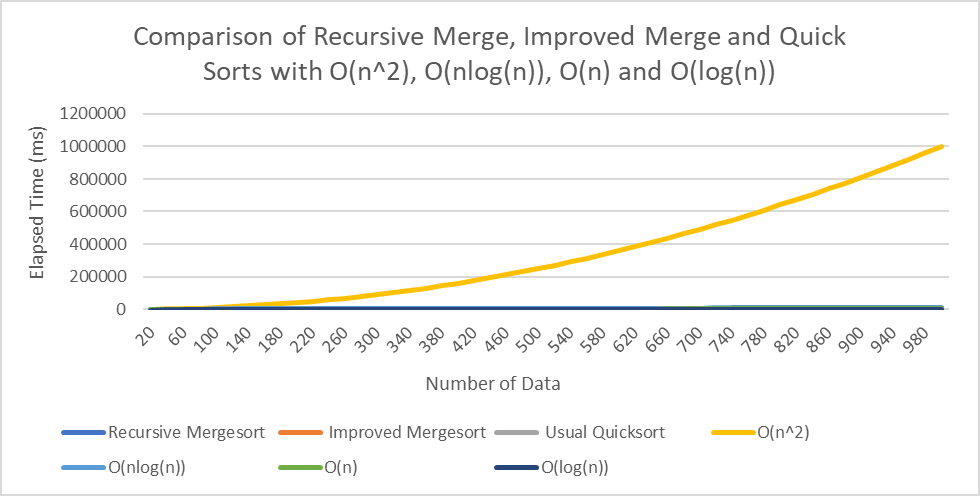
This is the data proves that my “Improved Merge Sort” algorithm works.

**Comparison of the Data for All Three Sorting Algorithms with O(nlog(n)), O(n) and O(log(n)), O(n^2).**

**Sorting Algorithms:**

* Recursive Merge sort
* Improved Merge sort (Iterative)
* Quick sort

**Figure 1: Comparison of Data for Recursive Merge Sort, Improved Merge Sort, and Quick Sort with O(n^2), O(nlog(n)), O(n), and O(log(n))**

****

Since, O(n^2) is too dominant, it is extremely hard to see what other graph does, therefore Figure 2 contains the data without O(n^2).

**Figure 2: Comparison of Data for Recursive Merge Sort, Improved Merge Sort, and Quick Sort with O(nlog(n)), O(n), and O(log(n))**

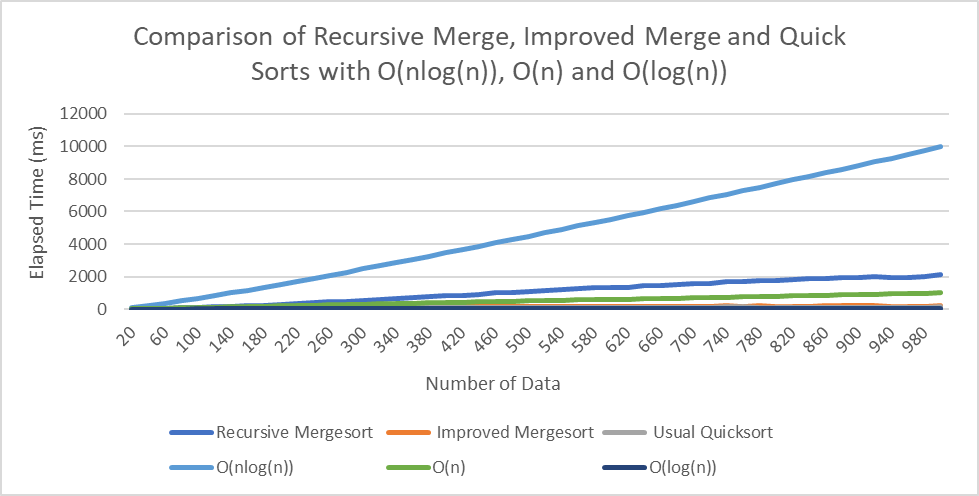
****

Figure 2 shows that O(nlog(n)), was too big that dwarfs other data, therefore, Figure 3 contains data without O(nlog(n)).

**Figure 3: Comparison of Data for Recursive Merge Sort, Improved Merge Sort, and Quick Sort with O(n), O(log(n)).**

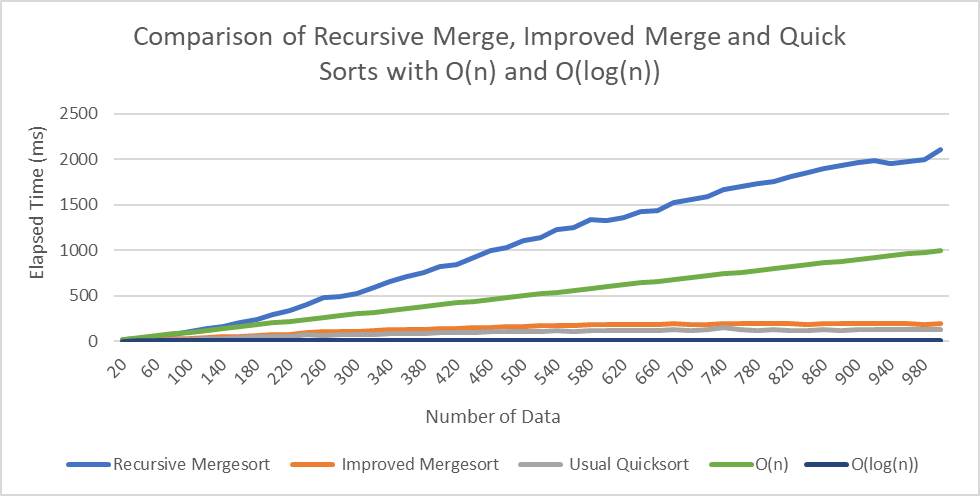
****

Figure 3 shows that among the three sorting algorithms “Recursive Merge Sort” is the most inefficient algorithms, however, it is hard to see whether “Quick Sort” is more efficiency than “Improved Merge Sort”. Therefore, figure 4 contains data without Recursive Merge sort.

**Figure 4: Comparison of Data for Improved Merge Sort and Usual Quick Sort with O(n) and O(log(n))**

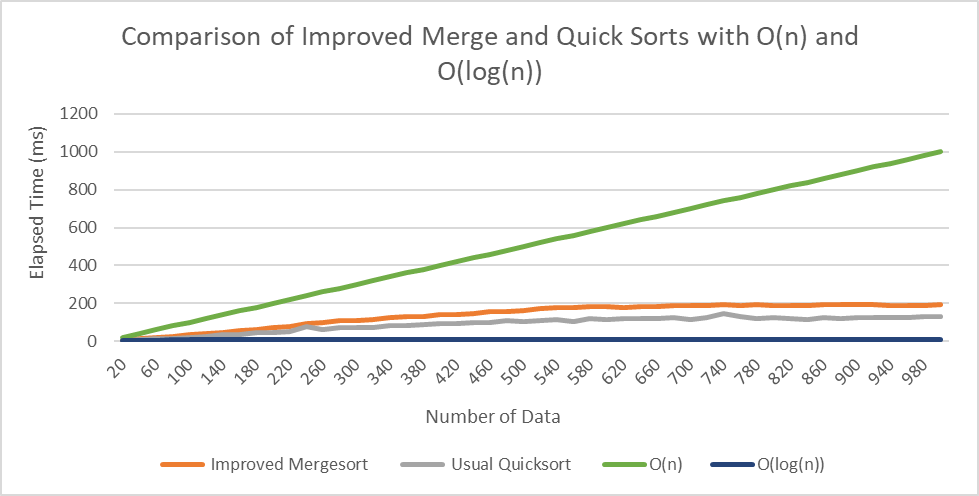
****

Figure 4 shows that “Quick Sort” is more efficient compare to “Improved Merge Sort”.

**Key Points from Figure 1 through Figure 4**

Among Recursive Merge Sort, Improved Merge Sort, and Quick Sort

* Recursive Merge Sort is the most inefficient.
* Quick Sort is the most efficient.
* Improved Merge Sort slower than Quick Sort but much faster than Recursive Merge Sort.

**Algorithm Analysis**

Each of the Algorithms, Recursive Merge Sort, Improved Merge Sort, and Quick Sort will be analyze by getting ratio to identify the time complexity. To get ratio, O(n^2), O(nlog(n)), O(n), and O(log(n)) will be used.

**Important Thing to Consider:**

The size of data will be 20 to 7000 increment by 20 every loop, because by size of 20 to 1000 would not give me precise result.

The size of data of 20 to 16700 increment by 20 every loop is also contained in this Algorithms Analysis to measure and analyze with better condition.

**Algorithm Analysis for Recursive Merge Sort**

To identify recursive merge sort algorithm’s time complexity is by getting ratio, which can be obtain by dividing O(n^2), O(nlog(n)), O(n), and O(log(n)) with Recursive Merge Sort elapsed time. If the result with dividing by O(x) is constant, then O(x) is the time complexity for Recursive Merge Sort.

However, before, analyze the time complexity for Recursive Merge Sort, the upper bound and lower bound had to be identified.

Figure 5 will display the graph that contains data of Recursive Merge Sort and O(n^2), O(n\*log(n)), O(n) and O(log(n)) size of data to 7000 increment by 20

**Figure 5: Comparison of Recursive Merge Sort With O(n^2), O(n\*log(n)), O(n) and O(log(n)) (with Data 20 to 7000 increment by 20)**

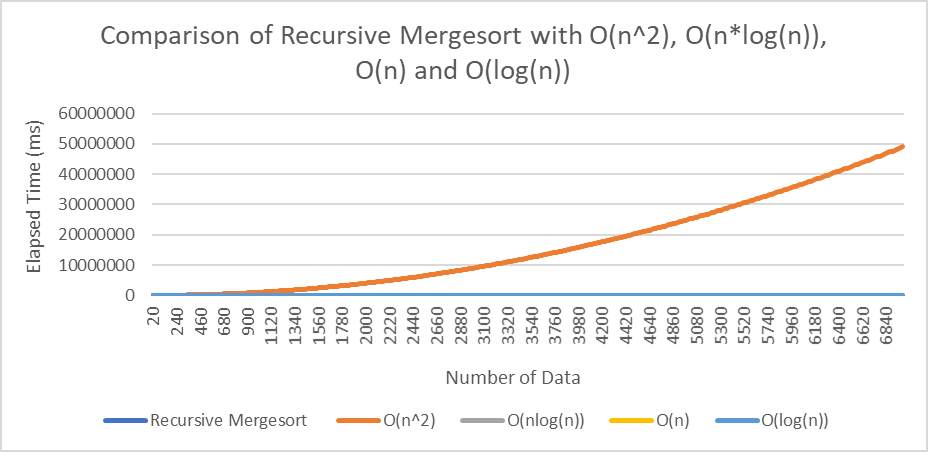
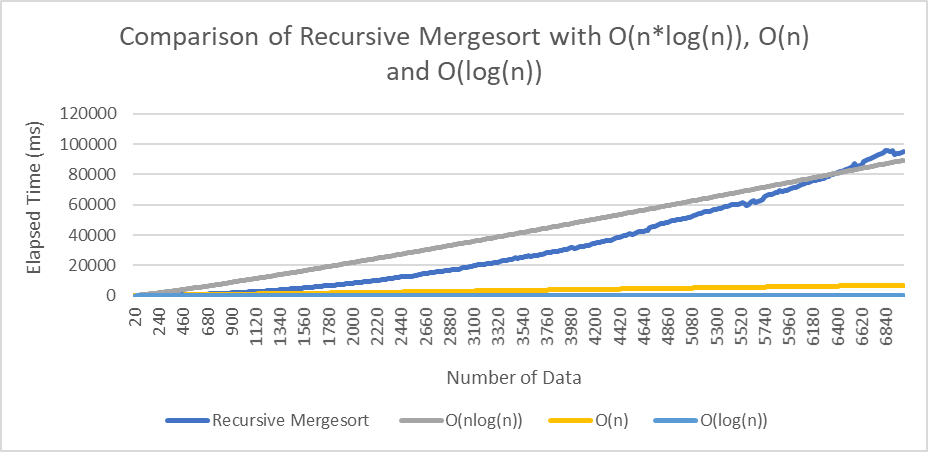
****

Figure 5 shows that the dominant is O(n^2), since O(n^2) is so big, it is hard to see the where other data lies on. So, Figure 6 contains all data but removed O(n^2)

**Figure 6: Comparison Recursive Merge Sort With O(nlog(n)), O(n), and O(log(n)) (with Data 20 to 7000 increment by 20)**



In figure 6, notice that Recursive Merge Sort has longer time complexity compare to O(nlog(n)) at the end, which means as size of data increase Recursive Merge sort is slower than O(nlog(n)).

**Key Points Figure 5 and Figure 6**

* Recursive Merge Sort is faster than O(n^2) when the size of data is equal to 20 to 7000 incremented by 20
* Recursive Merge Sort is eventually getting slower than O(n\*log(n)) when the size of data is equal to 20 to 7000 incremented by 20

Therefore, the upper bound for Recursive Merge Sort is O(n^2) and lower bound for Recursive Merge Sort is O(nlog(n)). Figure 7 will contain Comparison of O(n^2) / Recursive Merge Sort or (O(nlog(n)) / Recursive Merge Sort.

**Figure 7: Recursion Merge Sort Time Complexity data (O(n^2) / Recursive Merge Sort and O(nlog(n)) / Recursive Merge Sort) with Data 20 to 7000 increment by 20**

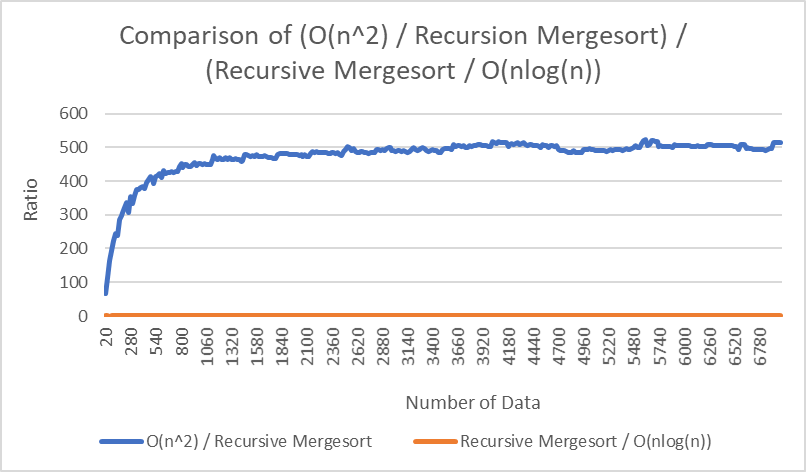


Figure 7 shows that the result of O(n^2) / Recursive Merge Sort is too big, it is hard to observe the data of Recursive Merge sort / O(n\*log(n)). So, figure 8, contains the data only Recursive Merge sort / O(n\*log(n)).

**Figure 8: Recursion Merge Sort Time Complexity data (Recursive Merge sort / O(n\*log(n))) (with Data 20 to 7000 increment by 20)**

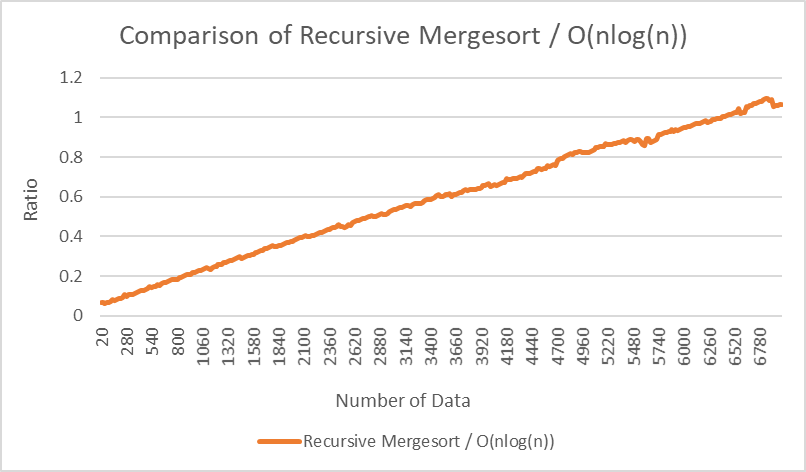
****

Figure 8 shows that the result which is the ratio of (Recursive Merge Sort / O(nlog(n)) shows that it is linear, which means it is not constant.

Therefore, I conclude that Recursive Merge Sort’s worst-case time complexity is O(n^2) as data increase.

**Algorithm Analysis for Improved Merge Sort**

To Analyze the algorithm analysis for Improved Merge Sort, the constant ratio is required. To obtain the constant ratio, dividing Improve Merge Sort elapsed time with its’ upper bound and below bound.

To analyze Improved Merge Sort’s upper bound and lower bound, figure 9 will contains data of Improved Merge Sort with O(n\*log(n)), O(n), and O(log(n)).

**Figure 9: Comparison of Improved Merge Sort with O(nlog(n)), O(n), and O(log(n))** **(with Data 20 to 7000 increment by 20)**

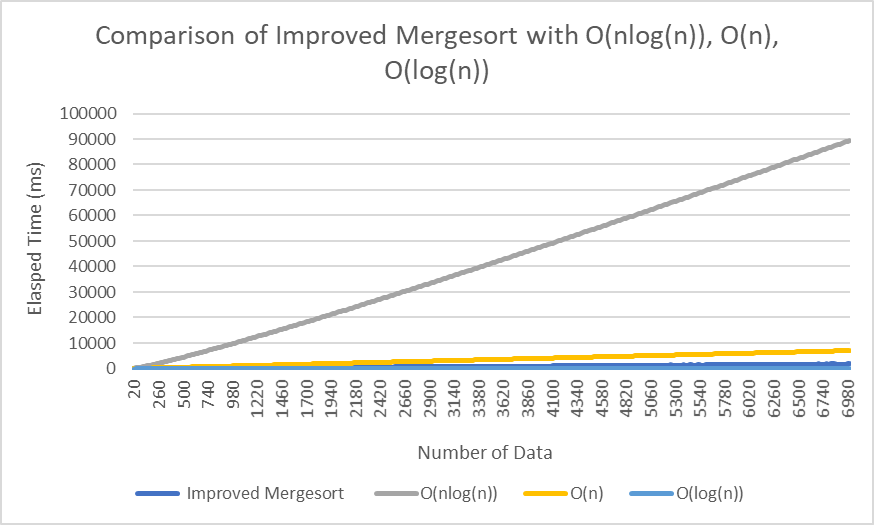


Figure 9 shows that O(nlog(n)) is too big which makes too hard to observe other data, therefore figure 10 will contains data exclude O(nlog(n)).

**Figure 10: Comparison of Improved Merge sort with O(n) and O(log(n))**

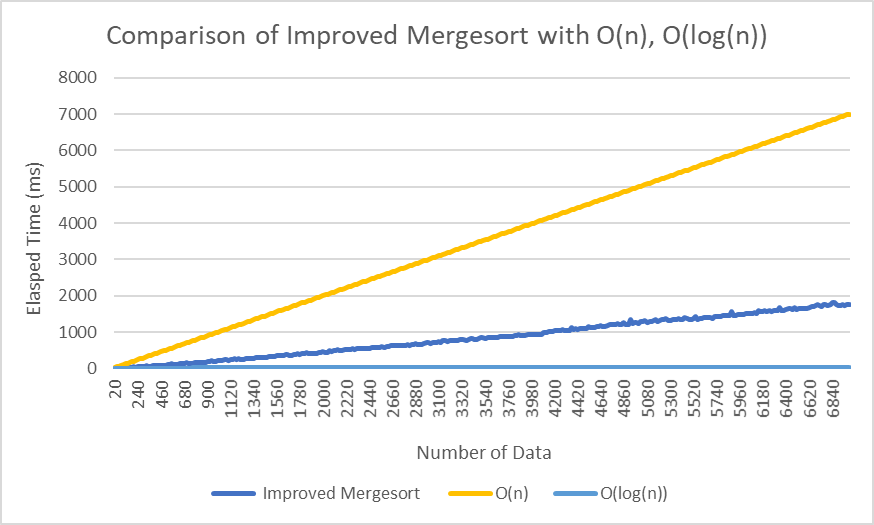


Figure 10 shows that Improved Merge sort is slower that O(log(n)) and by Figure 9 and Figure 10, the upper bound can be either O(nlog(n)) and O(n), therefore, to identify Improved Merge Sort requires of ratio with O(nlog(n)), O(n), and O(log(n)).

**Key Points Figure 9 and Figure 10**

* Improved Merge sort is faster than both O(n\*log(n)) and O(n)
* Improved Merge sort is slower than O(log(n))

Therefore, the upper bound for Improved Merge sort would be O(n\*log(n)) and O(n) and the lower bound for Improved Merge sort would be O(log(n)). Figure 11 will contain comparison of O(n\*log(n)) / Improved Merge sort, O(n) / Improved Merge sort, and O(log(n)) / Improved Merge sort.

**Figure 11: Improved Merge Sort Time Complexity Data**

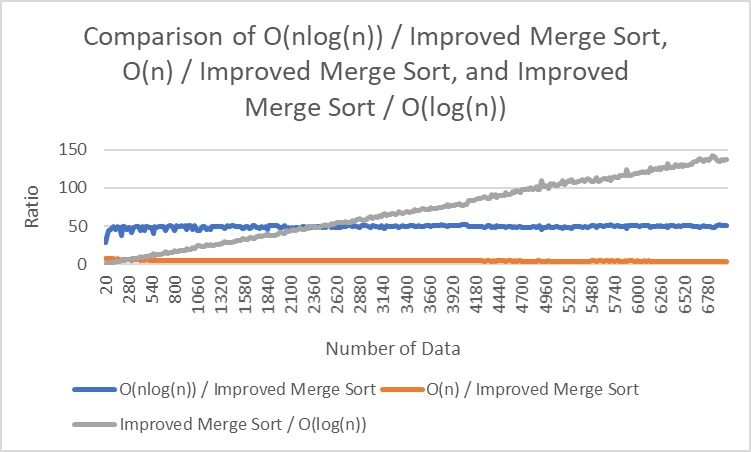


Figure 11 shows one thing clearly, which is Improved Merge Sort / O(log(n) is not constant. However, both O(nlog(n)) / Improved Merge Sort and O(n) / Improved Merge Sort look constant, therefore, figure 12 will contain only O(n) / Improved Merge Sort and figure 13 will contain only O(n\*log(n)) / Improved Merge Sort.

**Figure 12: Improved Merge Sort Time Complexity Data, O(n) / Improved Merge Sort**

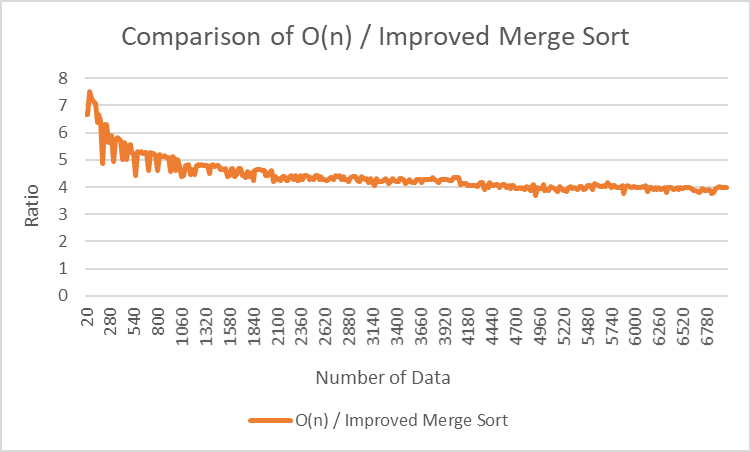
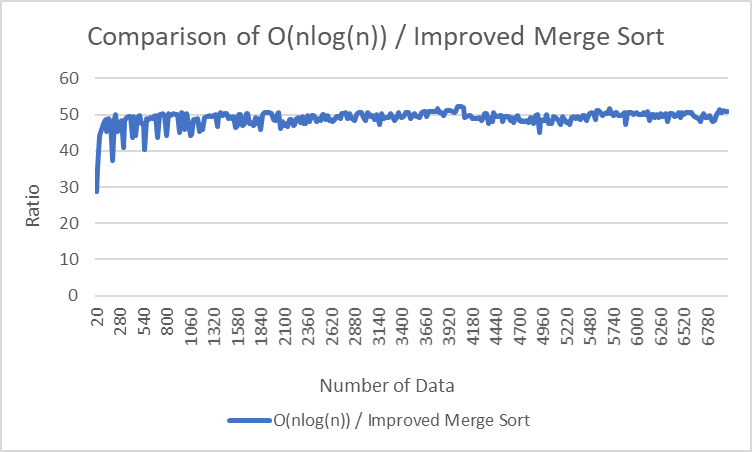
****

Figure 12 shows that the ratio of O(n) / Improved Merge Sort gave me constant.

**Figure 13: Improved Merge Sort Time Complexity Data O(nlog(n)) / Improved Merge Sort**



**Key Points Figure 12 and Figure 13**

* Both Ratio of O(nlog(n)) / Improved Merge Sort and O(n) / Improved Merge Sort seems constant in data size of 20 to 7000 increment by 20

Interestingly, both figure 12 and figure 13’s ratio result seems constant. So, through figure 14, I decided to increase the size of data to 16700, which will show the result differently.

**Figure 14: Improved Merge Sort Time Complexity Data O(nlog(n)) / Improved Merge Sort and O(n) / Improved Merge Sort (With Data Size 16700)**

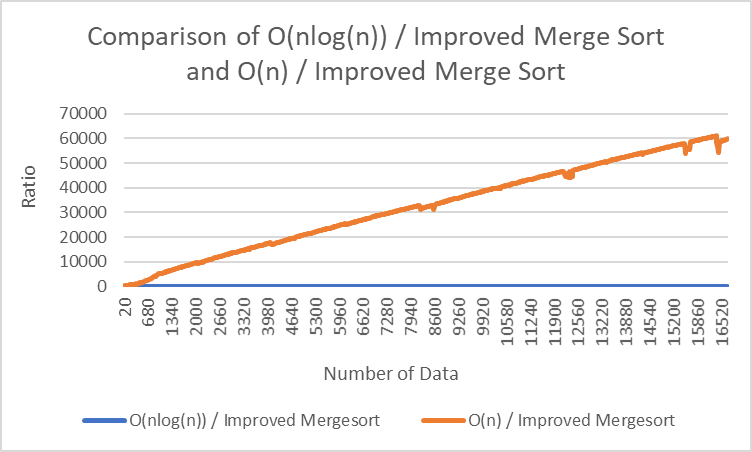
****

Figure 14 shows that O(n) / Improved Merge Sort as increase size until 16700 of data seems to have behavior of linear. Since, O(n) / Improved Merge Sort is too big, it is really hard to see O(nlog(n)) / Improved Merge sort is constant or not. So, Figure 15 will display only O(nlog(n)) / Improved Merge Sort.

**Figure 15: Improved Merge Sort Time Complexity Data (O(n\*log(n)) / Improved Merge Sort Only with Data Size 16700)**

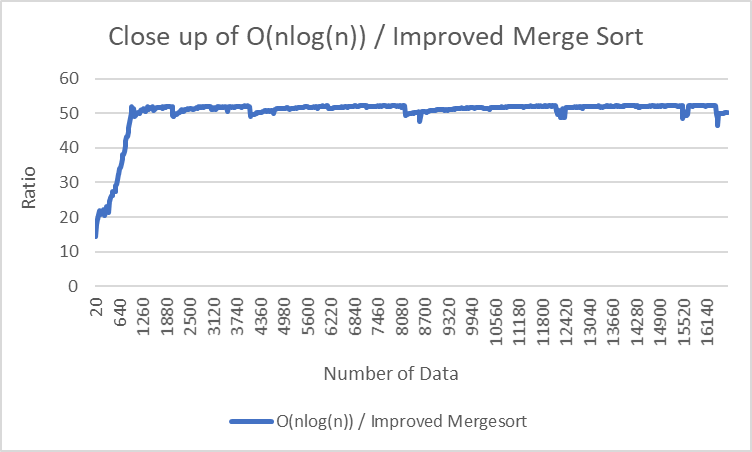
****

Figure 15 shows that the ratio of O(nlog(n)) / Improved Merge Sort is maintaining the ratio of constant even though the size of data increase dramatically. Therefore, I conclude the time complexity of Improved Merge Sort is O(n\*log(n)).

**Algorithm Analysis for Quick Sort**

To analyze the complexity of Quick Sort, it required to obtain the ratio and observe the ratio. To obtain ratio, upper bound and lower bound is required. So, figure 16 contains only Quick Sort with O(n \* log(n)), O(n) and O(log(n)).

**Figure 16: Comparison of Quick Sort with Run Time with O(nlog(n)), O(n), and O(log(n))**

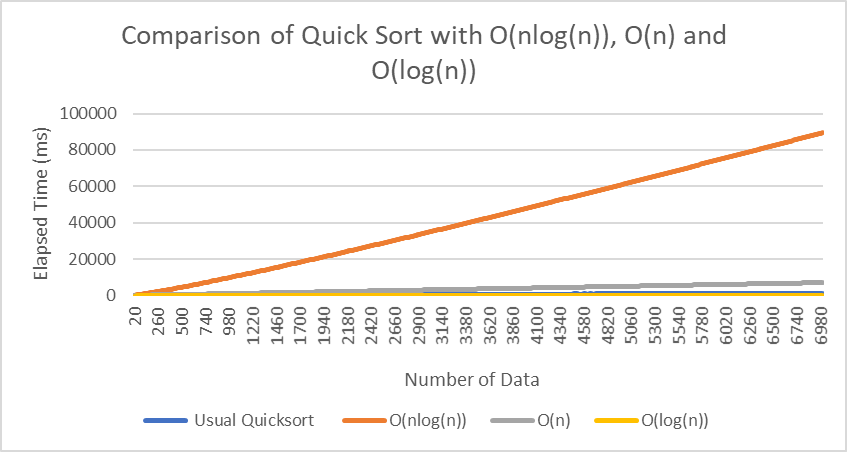
****

Figure 16 shows that O(nlog(n)) is clearly one of the upper bound of Quick Sort, however since O(nlog(n)) is too big, which make hard to observe other data. Therefore, figure 17 will contain data that exclude O(nlog(n)) to observe other data better.

**Figure 17: Comparison of Quick Sort with Run Time with O(n) and O(log(n))**

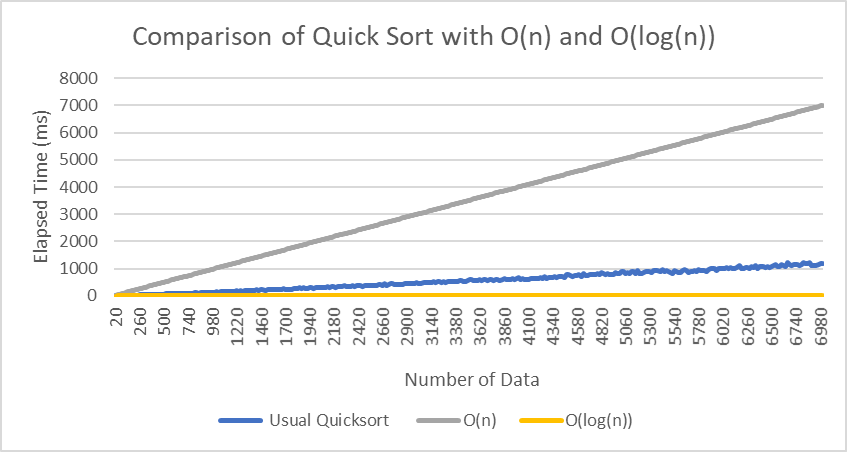


Figure 17 shows that Quick sort run times is between O(n) and O(log(n)), therefore, O(log(n)) is clearly the lower bound of Quick sort.

**Key Points Figure 16 and Figure 17**

* Both Run time O(n\*log(n)) and O(n) are upper bound of Quick sort
* O(log(n)) is lower bound of Quick Sort

Therefore, to measure the rate, O(n\*log(n)) / Quick sort, O(n) / Quick sort, and O(log(n)) / Quick sort is required. The figure 18 contains comparison of O(n\*log(n)) / Quick sort, O(n) / Quick sort and O(log(n)) / Quick sort.

**Figure 18: Quick Sort Time Complexity Data with O(n\*log(n)), O(n), and O(log(n))**

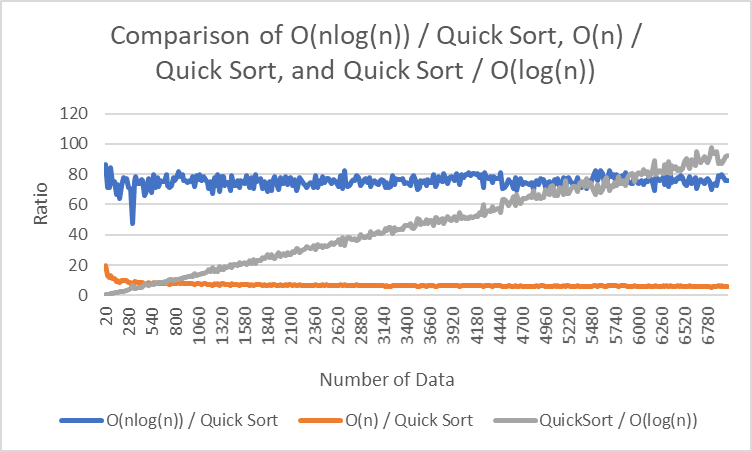
****

Figure 18 shows that Quick Sort / O(log(n)) is clearly linear, however the result of O(nlog(n)) / Quick sort and O(n) / Quick sort data seems constant. Therefore, observe the data individually was required. So, figure 19 will contain only O(nlog(n)) / Quick sort and figure 20 will only contain O(n) / Quick sort.

**Figure 19: Quick Sort Time Complexity Data, O(nlog(n)) / Quick Sort Only**

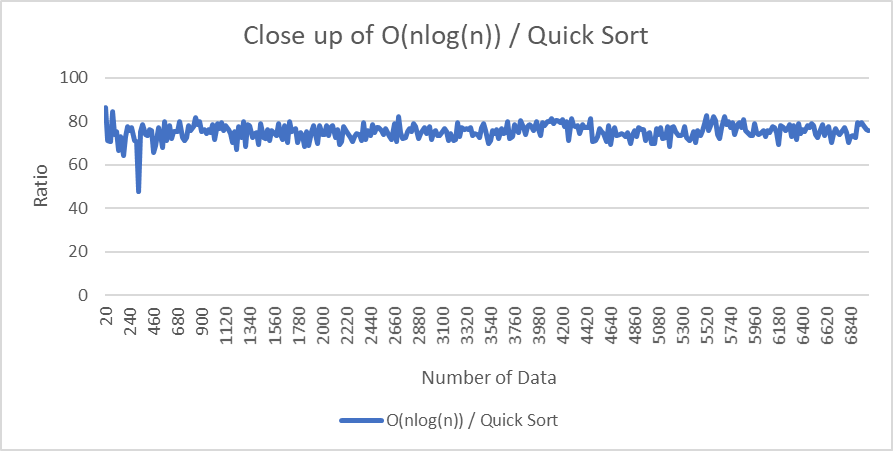
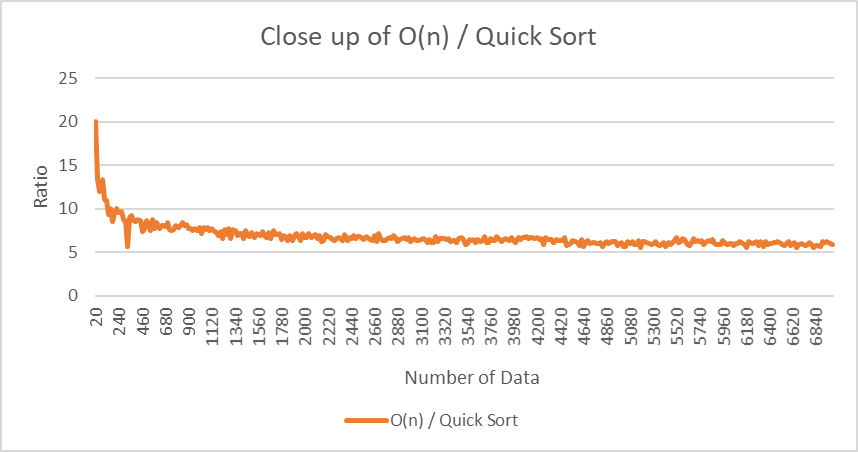


Figure 19 shows that the result of O(nlog(n)) / Quick Sort is constant.

**Figure 20:** **Quick Sort Time Complexity Data, O(n) / Quick Sort Only**



From figure 20 with size of data 7000, it is hard to identify it is constant or decreasing.

Since both the result of figure 19 and figure 20 look constant, to get better and precise data result, figure 21 will contain O(nlog(n)) / Quick sort and O(n) / Quick Sort with data size of 16700.

**Figure 21: Quick Sort Time Complexity Data, O(nlog(n)) / Quick Sort and O(n) / Quick Sort (Data size 16700)**

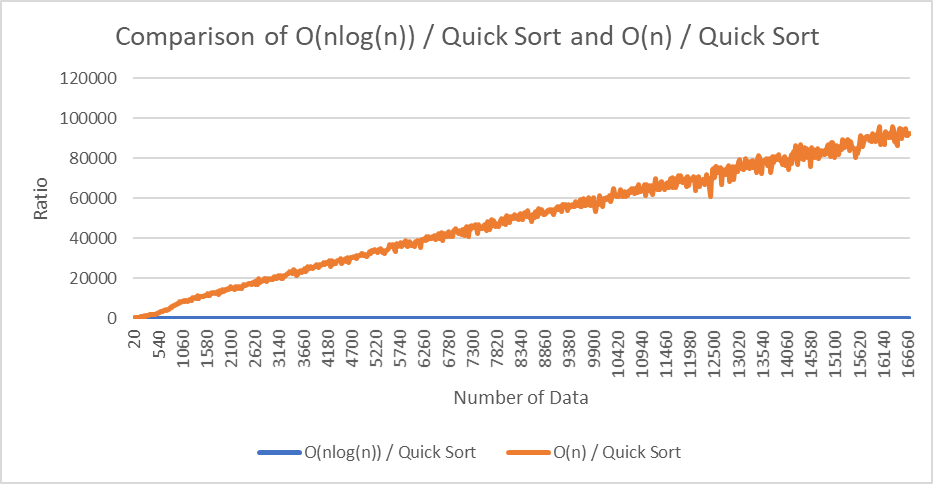
****

Figure 21 shows that with size of data 16800, the result of O(n) / Quick Sort is linear. However, it is hard to say that the O(nlog(n)) / Quick Sort is constant, therefore, to observe O(nlog(n)) / Quick sort’s ratio, figure 22 contains only O(nlog(n)) / Quick Sort with size of data 16800.

**Figure 22: Quick Sort Time Complexity Data O(nlog(n)) / Quick Sort Only (With Size of Data 16800)**

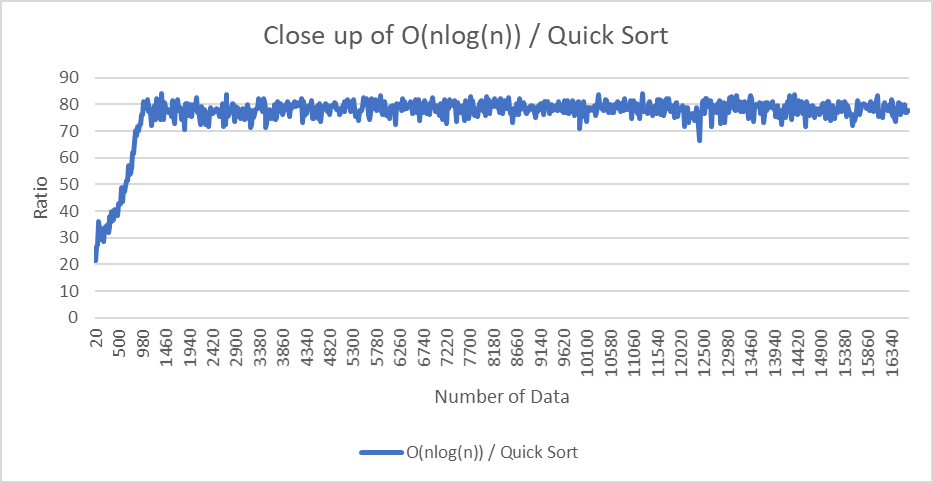
****

Figure 22 shows that the result of O(nlog(n)) / Quick Sort is constant with data size of 16800, which means even though the size of data increase, the ratio of O(nlog(n)) / Quick Sort is always constant. Therefore, I conclude that Quick Sort Time complexity is O(nlog(n)).

**Conclusion:**

First, I had to prove that the Improved Merge Sort works by sorting 30 elements. Secondly, from figure 1 through 4, comparing efficiency among Recursive Merge Sort, Improved Merge Sort, and Quick Sort, I concluded that the most inefficient among these three sorting algorithms was Recursive Merge Sort and the most efficient algorithm was Quick Sort, Improved Merge Sort showed amazing process and performance compare to Recursive Merge Sort. By analyzing the sorting algorithms’ time complexity by ratio which obtained from dividing with O(n^2), O(nlog(n)), O(n), O(log(n)), the data 20 to 1000 was too small, therefore, I had to increase data size to 7000 and even 16800 to analyze the time complexity of algorithm precisely. Finally, I concluded the time complexity of Recursive Merge Sort is O(n^2), Improved Merge Sort is O(n\*log(n)), and Quick Sort is O(n\*log(n)).