Introduction

Merge and quick sort have the same time complexity of O (n log n). Through this experiment we will explore how all the copying operations within merge sort slows it down as compared to quick sort. By measuring the amount of time it takes each function to sort an array of varying sizes we can get a better of understanding of each sort methods performance. Our goal is to write an iterative version of the merge sort function and determine its run time complexity.

Procedure

The program driver randomly generates numbers to create three array copies of size 20 and measures the time it takes to sort the same array in the merge, improved merge and quick sort functions. With each pass the size of the array increases by 20 (i.e., size = 20, 40, 60, 80…). Results were then plotted on a graph along with O(n), O( log n) and O( n log n) to compare their performance. Each function was then compared against its upper and lower bound to determine which time complexity it is closest to.

Data Comparison of Three Sort Functions

In the first graph we can see the three functions along with O(n), O (log n), and O( n log n).

Noted here, as the size of the array increases, the amount of time it takes for the function to sort the data increases. Merge rises quickly after the size is more than 300. In stark contrast we can see that the improved merge sort and quick sort increase very slowly over the increase in size of the array.

Figure 1: Merge Sort, Improved Merge Sort, Quick Sort, O(n), O(log n) and O( n log n) Data

Data Comparison of Merge Upper and Lower Boundaries.

In this figure we can see merge sort marked in blue. Above it we see in yellow O (n log n), we will use this in further analysis as the upper bound. From below we will compare with O(log n) from the lower bound.

Figure 3 shows a better view of their behavior when the array size is smaller.

Figure 2: Merge Sort, O(n), O (log n), O(n log n) Data Up to Array Size 1000

# Analysis of Merge Sort Time Complexity, Recursive

Here we see the plot of Merge sort over a size array of 200 bound at the top by O(n) in grey and below by O(logn) in the yellow. Figure 4 shows that Merge/O(n log n ) has linear relationship much like we expect to see when the time complexity is true. The proportional relationship forms a smooth line, something we do not see in Figure 3 when checking merge over the upper bound of O(n).

Figure 2: Merge Sort, O(n), O (log n), O(n log n) Data Up to Array Size 200

Figure 3: O(n)/Merge

Figure 4: Merge/O(n log n)

Analysis of Improved Merge Sort Time Complexity, Iterative

From above Improved Merge in orange up bound by O(n) and O(n log n) and below by O(log n). It is expected that this version of sort will be linear when comparing to O(log n). It was necessary to change the horizontal x-axis to better visualize the relationship between the improved merge function and O(n) above and (logn) below.

Figure 5: Improved Merge Sort, O(n), O (log n), O(n log n) Data Up to Array Size 1000

Figure 6: Improved Merge Sort, O(n), O (log n), O(n log n) Data Up to Array Size 200

On further inspection things change unexpectedly. The upper bound plots of in figures 7 and 8 demonstrate that there is no linear relationship between improved merge and its upper bonded plot lines. Looking at Improved merge and its lower bound of O(log n) we can see somewhat of a linear relationship. This demonstrates we have the expected improvement of the function. It was expected to have behavior more similar to O(n log n).

Figure 7: **O(n)/Improved Data Plot, Array Size Up to 1000**

Figure 8: **O(n log n)/Improved Data Plot, Array Size Up to 1000**

Figure 7: **Improved/O(logn) Data Plot, Array Size Up to 1000**

Analysis of Quick Sort Time Complexity

Quick sort is looking very similar to improved merge in its lower bound. I will compare the quick sort to both O(n log n) and O(n) for thoroughness here as well even though it looks closer to O(n) at its upper bound.

Figure 8: ***Quick Sort, O(n), O(log n), O( n log n) Data Array Size 1000***

Figure 9: ***Quick Sort, O(n), O(log n), O( n log n) Data Up to Array Size 200***

Analysis of quick sort function required looking at the data over a smaller subset. Including more data points caused the charts to look like a jagged curve, showing no linear relationship. It was curious to see that the graph looked to crowed to form any substantial conclusion from the data.

Figure 10: Quick/O(log n) Data over array size 20-200

Figure 11: O(n)/Quick Data over array size 20-200

Figure 12: O(n log n)/Quick Data over array size 20-200

Discussion

A few challenges came up during this exercise. Each sort function was looked at over different array lengths to determine how each graph should be compared to the other time complexities. While generating the data points over size 1000 proved to be faster the smaller number of data points resulted in graphs with serrated shape. In order to get a more accurate reading of the information I had to go back and generate larger arrays to collect more time data.

Once that task was completed it was more productive to the process of analysis. Based on the upper and lower bound comparisons it was made clear where the linear relationships between the elapsed times and the different time complexities of O(n), O(log n), and O( n log n).

Conclusion

Based on the analysis of the elapsed time analysis of the different sort methods it can be concluded that quick and merge sort are truly of time complexity of O( n logn). Improved sort was slightly slower than quick sort showing a linear relationship with the time complexity of O (log n) as seen in figure 7.