Alex Davis & Tanyan Mars

Professor Jacob Gillenwater

Data Structures

03 March 2025

Comparing Merge Sort to Insertion Sort

When it comes to deciding what sorting algorithm to choose, two main questions come to mind. What is the variable type, and how fast do you want it to run? These two main questions limit the number of choices/preferences you have. For example, when comparing merge sort and insertion sort, factors depend on input size, memory, and how fast you want it to run. The question then becomes: how do you decide on which algorithm to use for the given situation? The answer is simple, you must compare the advantages of both and their tradeoffs.

When comparing merge sort and insertion sort, the first thing we decided to look at was the type of variable. This will play a huge role in how fast the algorithms run. To see the main difference between the two algorithms without having to create multiple different classes for them, we made the algorithms generic typing so that the original variable typing doesn’t influence the algorithm. Next, we gave them two distinctively different types being received from an outside file. These two variable types are integers and book objects. Each test case was ran 3 times and averaged out. First, we will start with the integers because of their simplicity compared to the book objects. When it came to the integers, the merge sort was much quicker, having the smaller run time (See in figure 1).

Figure

In the image you clearly see that in almost every occasion the insertion sort is much slower than merge sort. When comparing the book object, you still see the same outcome (See in figure 2).

Figure

While looking at these graphs it seems that the book comparison was better for both sorting algorithms, however this isn’t the case. If we change the style of the graph, you can clearly see that the integer type is a lot easier to handle for these algorithms (See in figure 3).

Figure

In that graph you can clearly see that both the merge and the insertion sort were more efficient dealing with integers compared to books. The best-case scenario for the insertion sort and the merge sort was AlmostInOrder\_10, because there is less to change in the list. Both algorithms were less than a millisecond for the sorted list. The worst-case scenario for the insertion sort is obviously ReverseOrder\_10000 using the books. The same is said for merge sort but is still clearly seen that merge sort is better and more efficient in every case. This would make sense because of both algorithms big O notation. The merge sort has an average big O notation of O(n\*logn) while the insertion sort is O(n^2). However, I refrain from letting these notations be the entire truth of the matter. The fact is that the process at which these sorting algorithms work is the main cause of their time efficiency. Merge sort splits the array up until almost every item is on its own, then groups them back together one at a time in the correct order. However, insertion sort works by taking each element of the list and placing in its correct position relative to the already sorted part of the array.

The two main questions when deciding on what sorting algorithm to use are how fast you want your program to run and what type of variables you are using. However, when it comes down to comparing merge sort and insertion sort, merge sort beats out insertion sort in most cases. However, in certain cases insertion sort may be the better option. Such as, cases around 100 variables insertion sort outperformed or was on the same level as merge sort (see figure 4). This is why it is important to cater the algorithm you choose to your needs and test them to see which is best for your case.

Figure