Matthew Horrocks
September 16th, 2020

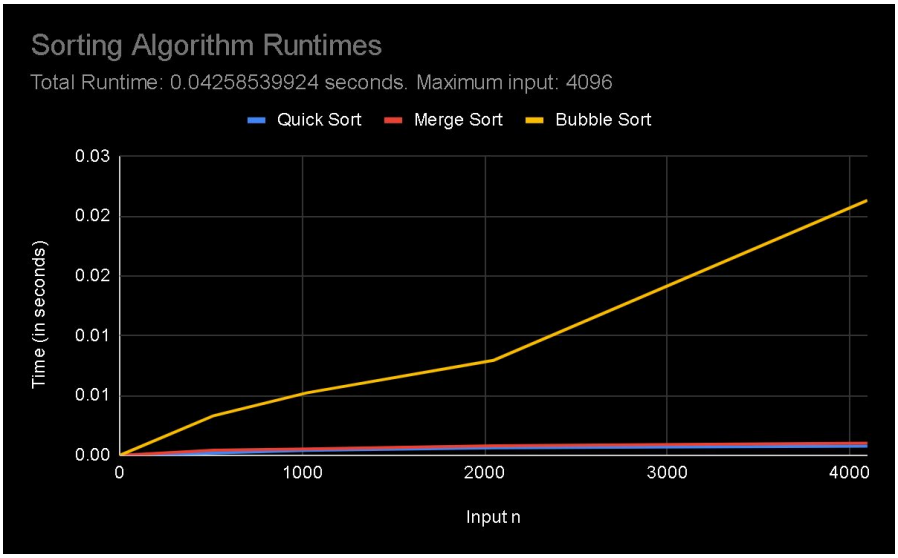
Time Complexity/Runtime Analysis of Sorting Algorithms

On the question of the efficiency of a sorting algorithm, the difference between a merge, quick, and bubble sort function creates a small difference in performance; apparent in both small and larger input sizes. This can be demonstrated through algorithms that sort through pseudo-random numbers of up to input size n. The three algorithms that are presented here run very differently; merge sort has a time complexity of O(n*logn) in it's best and worst case due to it's recursion from dividing the array into two subproblems (taking the middle index), while taking linear time to merge the two subproblems. The time complexity requires the same amount of computing power whether or not the array is sorted, which means it wouldn't be a preferred method of searching arrays. while the polynomial-time function maintains a constant-time iterative cycle without the recursion. For the quick sort algorithm, the time complexity is deterministic. The worst case scenario is much worse (O(n^2)) in comparison to the best and average case scenarios (O(nLogn)). When the partition process begins at either the largest or smallest chosen pivot point (or even worse, if it was already sorted), this recursively becomes the worst-case scenario. Average-case and best-case solutions are to choose the middle point, or an element at random. However, according to the data I collected, quick sort is by far the most efficient sorting algorithm when the index is considered as a parameter. If I were given the choice to choose between merge, quick, and bubble sort to implement some type of sorting technique to sort 2 gigabytes with 200 megabytes of main memory, I would choose the quick sort method with a modification in index parameters. This is because then it will be stable and it will only use more space than necessary to store the required recursive method calls.

Matthew Horrocks
September 16th, 2020

Before we discuss the results of the three algorithms, you can implement it yourself. The source code lies within one Java file - which runs independently from any package or other imports (utilizing only your system time). It was completely written in Java, so you can use any type of Java IDE you prefer (IntelliJ, Eclipse, Netbeans.. etc.) to import the file into. I have created a total of seven separate functions. All seven functions are called at least indirectly. Running the file should default to running the main function. The time functions are all within main. To change the input amount, modify the number of the declared integer array variables. Without changing this number, both programs will take inputs up to 4096 (this will take longer than my results, depending on your computer).

Below, you will see a chart that includes the running time (in seconds) and the input (n) of all three sorting functions. As you can see, the merge sort function behaves similarly to the quick sort function. The bubble sort function takes much longer to compute than the other two.



Matthew Horrocks
September 16th, 2020

From the data, we can conclude that the quick and merge sort functions will always be more performant than its bubble sort counterpart. Thus the quick sort and merge sort function will be faster in larger inputs..