Observations on Various Sorting Algorithms: Selection Sort, Bubble Sort, Insertion Sort, Quick Sort, and Merge Sort

An Analysis

Joel Simrose

October 5, 2018

Introduction

By analyzing the following sorting algorithms: selection sort, bubble sort, insertion sort, quick sort, and merge sort we are able to gain greater knowledge as to why one algorithm can be better than another. In order to classify one algorithm as being better than another, we must observe its speed, soundness, amount of space required, completeness, and tractability. This analysis will be a look at how fast each algorithm is and how different cases may impact an algorithms speed. In doing so, we are able to understand when it is best to use a specific algorithm in computer programming, as well as when not to. Some common examples of using these sorting algorithms are: bubble sort for sorting TV channels based on the length of viewing time, insertion sort is used when dealing with small or mostly sorted arrays, quick sort for sports scores based on win-loss ratio, and merge sort for databases to sort data that is too large to be loaded entirely into memory.

Background

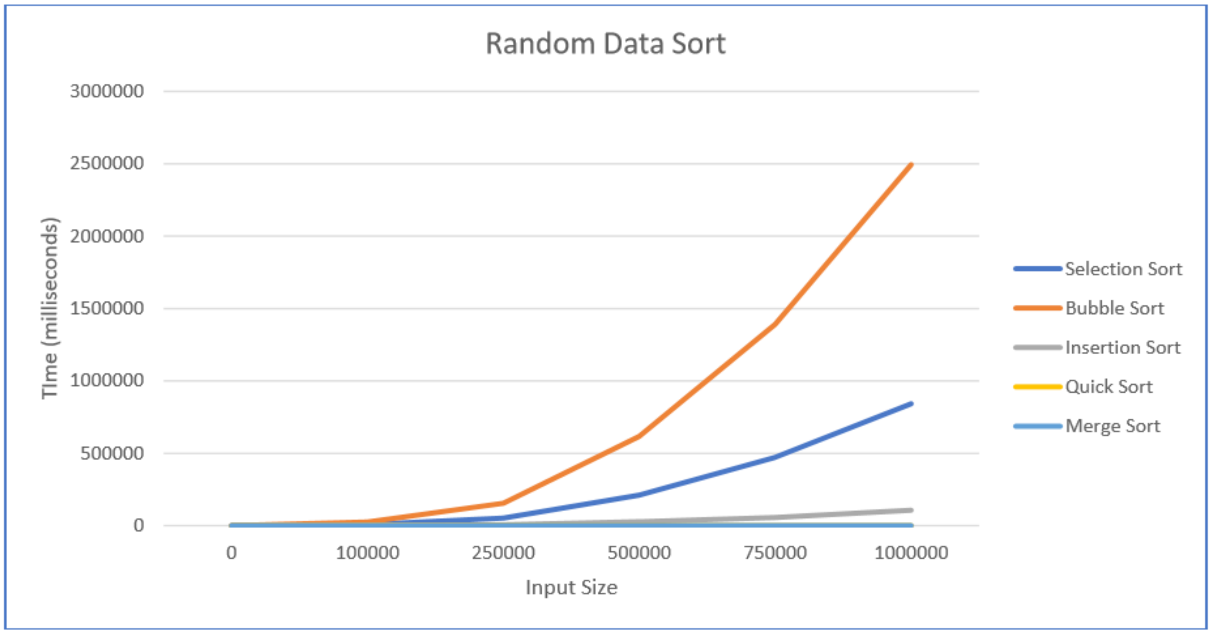
In order to gain more knowledge about these sorting algorithms, multiple scholarly articles were observed. One of these was Niraj Kumar Singh's: "A Statistical Approach to the Relative Performance Analysis of Sorting Algorithms". In this article Singh stated that "the standard quick sort works extremely well on average inputs, but its lack of robustness for certain patterns might be a performance bottleneck in certain situations." This is something we must take into consideration when choosing the best algorithm since although an algorithm may be fast, it may not be fast in every circumstance. Another article which was analyzed was Richard J. Maresh's article "Sorting Out Basic Sorting Algorithms". Here he describes bubble sort as being "the easiest sorting algorithm to understand, although many do say something about its relative lack of efficiency." Since the bubble sort algorithm is easy to understand, it is especially good for learning purposes, although it may not be the fastest algorithm due to it's lack of efficiency. A third scholarly article which I analyzed was Owen Astrachan's "Bubble Sort: An Archaeological Algorithmic Analysis". Here it stated that some books laud bubble sort because it runs in O(n) time on sorted data and works well on “nearly sorted” data. He later concludes that “insertion sort is better than bubble sort, is stable, and is the basis for more efficient shell sort.”

Method

For this analysis, the coding language Java was used. Throughout all algorithms, the underlying data structure used was an array. To perform these tests, a Lenovo Y40-80 laptop was used which has an Intel Core i7-5500U CPU @ 2.4GHz x 2 processor, as well as 16 gigabytes of RAM. The operating system used was Linux Mint 18.1. In order to get a data point for a graph, three data points were averaged.

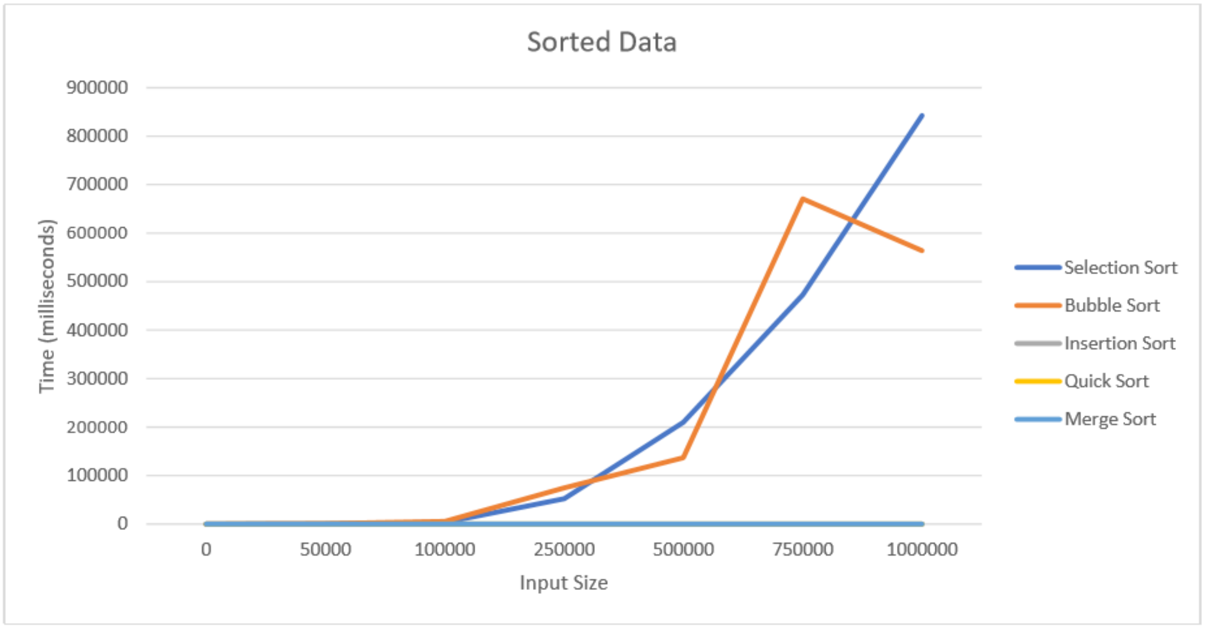
Results

Figure 1



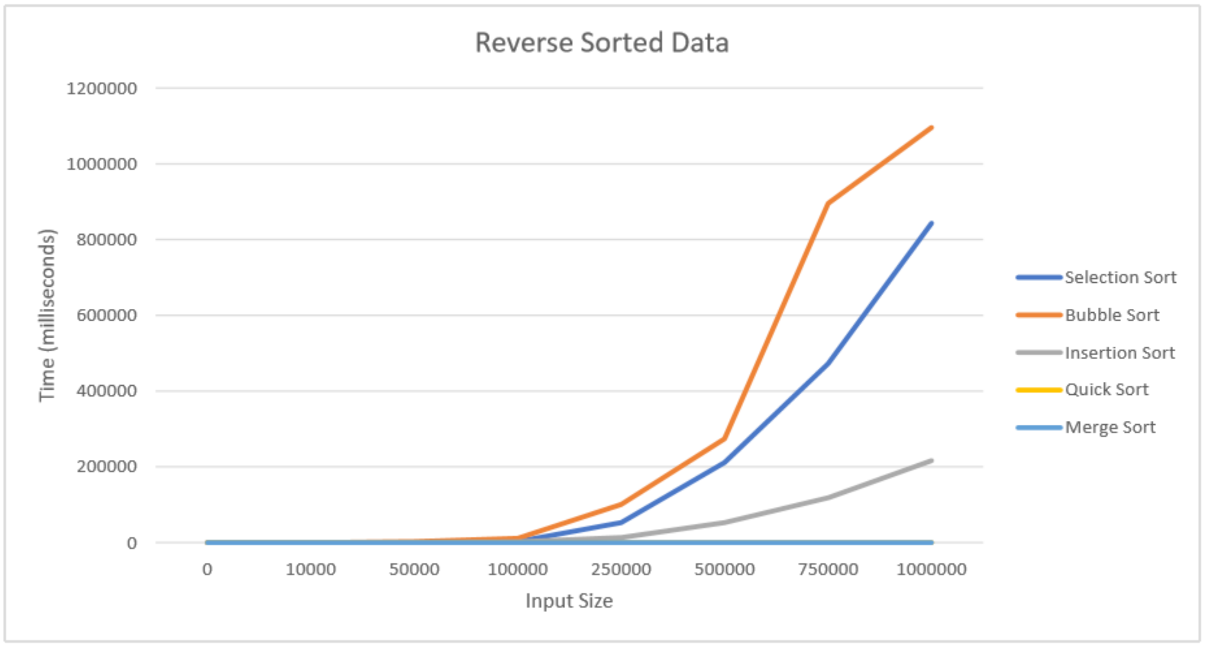
Timing tests of the five sorting algorithms for random data are shown in Figure 1.

Figure 2



Timing tests of the five sorting algorithms for sorted data are shown in Figure 2.

Figure 3



Timing tests of the five sorting algorithms for reverse sorted data are shown in Figure 3.

Conclusion

After analyzing the sorting algorithms selection sort, bubble sort, insertion sort, quick sort, and merge sort we are able to conclude that quick sort and merge sort are usually the best algorithms. As stated in the introduction, for this analysis we are classifying the best algorithm as being the fastest. Quick sort has an expected running time of O(nlogn) and a worst-case running time of O(n^2). Merge sort on the other hand has an expected running time of O(nlogn) as well as a worst-case running time of O(nlogn). As observed during testing, quick sort and merge sort performed the best with small differences in running times. Quick sort is most suitable for a randomized data set. During testing a source of error that was experienced was when a sort with a larger data size took less time. This may have been due to the randomization and how the first set of random numbers may have been closer to the best case and the second set of numbers may have been closer to the worst case which may have caused the calculations take longer. Another source of error was that although quick sort is best with a randomized list, it still appeared to work very well for the sorted and reverse sorted lists as well. This may resolve itself if larger data sizes are used for testing.

Works Cited

Niraj Kumar Singh and Soubhik Chakraborty. 2012. A statistical approach to the relative performance analysis of sorting algorithms. In *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology* (CCSEIT '12). ACM, New York, NY, USA, 57-62.

Richard J. Maresh. 1985. Sorting out basic sorting algorithms. *SIGCSE Bull.* 17, 4 (December 1985), 54-64.

Owen Astrachan. 2003. Bubble sort: an archaeological algorithmic analysis. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education* (SIGCSE '03). ACM, New York, NY, USA, 1-5.

Gootooru, N. (n.d.). Program: Implement quick sort in java. Retrieved from https://www.java2novice.com/java-sorting-algorithms/quick-sort/

Insertion Sort in Java - Javatpoint. (n.d.). Retrieved from https://www.javatpoint.com/insertion-sort-in-java

Merge Sort. (2018, August 30). Retrieved from https://www.geeksforgeeks.org/merge-sort/

Quicksort Sorting Algorithm in Java. (n.d.). Retrieved from https://javarevisited.blogspot.com/2014/08/quicksort-sorting-algorithm-in-java-in-place-example.html

Selection Sort in Java - Javatpoint. (n.d.). Retrieved from https://www.javatpoint.com/selection-sort-in-java