A Computational Comparison of Sorting Algorithms

Ayush Gupta*

6 December 2018

Abstract

Sorting algorithms are used to sort data into order. Previous researchers have used asymptotic analysis method and considered worst running time of algorithms for comparison of sorting algorithms. This project used practical cases and considered average running time of algorithms for comparing four sorting algorithms, namely, Bubble sort, Insertion sort, Merge sort and Quick sort. The purpose of this project was to find out the difference in practical method and theoretical method of comparing sorting algorithms. Based on the results from this project, the practical method of comparison offers more insights on comparing sorting algorithms. In this research, Merge sort is found to be the best sorting algorithm considering the average number of operations for comparison. Further research can be done on comparing other algorithms using the practical method of comparison from this research. The results of this project have applications in computer science, engineering, finance and economics.

1 Introduction

A sorting algorithm is an algorithm used to sort data into an order, preferably ascending or descending order. When we are presented with a huge amount of data, then we often need to first sort it, in order to easily perform other tasks on it. For example, to find the frequency distribution of the data, the first step is to sort it so that it can be represented in an easier way. This is done by using a sorting algorithm. Some of the other uses of sorting algorithms are for searching, closest pair and convex hulls. For searching, it is always crucial to sort the data into either ascending or descending order. Binary Search, one very popular method of searching, relies on the sorting of data. Closest pair means to find a pair of numbers, among a set of numbers, which has the smallest difference between them. When we have sorted data then that closest pair is easier to find out, as it is just one of the pairs of adjacent elements among the sorted data [1]. These examples show that sorting is indeed the first important step in analyzing data and sorting algorithms are very useful for sorting. Some of the most popular sorting algorithms are Bubble sort, Insertion sort, Merge sort and Quick sort [2]. These four sorting algorithms are considered in this project.

Now that we have discussed the importance and provided examples of sorting algorithms, the next step is finding out which sorting algorithm is the best one. One of the mostly used method to compare the runtime of sorting algorithms by previous researchers is asymptotic analysis considering the worst case [3]. This method assumes that the input size of data tends towards infinity and considers the worst running time of the algorithms, where the running time is measured by the number of operations used by an algorithm to sort the data [3]. However, in real life, the input size varies according to situation and it need not be large enough to be considered infinity. Similarly, in real life, the case might not be the worst and it is needed to just consider the average running time of the algorithms because the case of average running

*Department of Mathematics, the Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong. Email: **aguptaaf@connect.ust.hk**

time is the most probable. Therefore, there is a need to consider the cases when input size is not assumed to be infinity and the average running time of the algorithms is considered, then compare the sorting algorithms for these cases. This need brings us to this project.

In this project, a study of the difference in sorting algorithms for different input sizes is conducted. This is done by computing the average number of operations, used as running time, taken by various sorting algorithms for sorting randomly generated data sets. The purpose of this project is to find out if there are differences in using the method of asymptotic analysis considering the worst case and the method of computational analysis considering average case for comparing sorting algorithms. Here, computational analysis involves comparing the performance of sorting algorithms on randomly generated data sets where the input size takes fixed finite values. The results from this project will help provide a practical method of finding which sorting algorithm is the best one, according to the size of data that we are dealing with. It will help in making the process of sorting quicker and easier for search, frequency distribution, convex hulls and closest pair [1]. The results from this project have applications in computer science, designing, engineering, finance and economics.

2 Theory

In this project, the following four sorting algorithms, Bubble sort, Insertion sort, Merge sort and Quick sort are considered.

Bubble sort is the simplest sorting algorithm that compares each pair of elements and sorts them accordingly into ascending or descending order. The working of Bubble sort is to repeatedly compare adjacent elements and swap them if they are in the wrong order. For example, if we are looking for ascending order, then in [4,2], first 4 and 2 will be compared, then they will be swapped because they are in descending or wrong order [4].

Insertion sort repeatedly performs the operation of sorting and insertion. For example, it performs sorting of the first i elements and then inserts the (i+1)th element in its correct place. The insertion of the (i+1)th element is done by comparing it with the (i)th element, if the (i)th element is smaller than the (i+1)th element and we are looking for ascending order, then the order is already correct. If the (i)th element is greater than (i+1)th element, then we compare the (i+1)th element with the (i-1)th element, this process is repeated till we find the correct position of the (i+1)th element and then place it there [3].

Merge sort and Quick sort are based on the Divide and Conquer approach. While merge sort first divides the array at middle and sorts those 2 halves and merges them, Quick sort first divides the array at a randomly selected place and then sorts those 2 parts and merges them. The process of divide and merge is recursive and the division is done till only single element arrays are left [3]. The possible benefit of Quick sort is its lesser space requirement than Merge sort. While Merge sort requires an additional array of size similar to the number of elements to be sorted, Quick sort does not need any such additional space [5]. However, in this project we are only considering the running time of algorithms for comparison.

The python codes of the 4 sorting algorithms considered by this project are borrowed form the site geeksforgeeks and then modified to count the number of comparisons giving the number of operations, which is used as the running time of the algorithms. [6]

In this project, the number of operations is counted by counting the number of comparisons, between any pair of elements, made by the sorting algorithms. This is done because the cost of number of comparisons generally dominates the cost of all the other operations. This method is also used in previous researches related to comparison of sorting algorithms [3]. The term input size refers to the number of elements to sort at a time by an algorithm. All logarithmic operations consider base equal to 2 for this project.

3 Methodology

3.1 Computing the average number of operations

This project uses python programming language to compute the average number of operations taken by an algorithm to sort a set of data of input size n. For practically comparing the average number of operations, different data sets for various input sizes, $n = 100$, 1000, 10000, 100000, 1000000, are constructed. This is done by generating m arrays of n integers each, where each integer is randomly selected from the range 1 to n. Here, m is used for offering multiple trials to the algorithms to sort arrays of n integers each. The maximum value of m considered by this project is 100000 for $n = 100$ and the general value of m considered is 1000. This gives us several data sets of size $m * n$. Then the algorithms are run one by one on a data set and the number of operations taken by an algorithm to sort each array of n elements is computed. Then the total number of operations taken by an algorithm to sort all the m arrays of n elements each is calculated. By dividing the total number of operations by m, we get the average number of operations taken by an algorithm to sort n elements. This is done for all the algorithms on a data set. The process is repeated for different data sets having different values of n. This gives us the average number of operations taken by different algorithms to sort n elements.

As a very simple example, assuming we have $n = 3$ and $m = 2$, then the elements range from 1 to 3, an example of such data set can be [1,2,3] and [3,1,2], then an algorithm will be run on each of [1,2,3] and [3,1,2] and the number of operations will be noted, then the total of the number of operations will be calculated, later this total will be divided by $m = 2$ to give the average number of operations on $n = 3$ for that algorithm. This process will be repeated on this data set for all the algorithms giving the average number of operations taken by each of the algorithms for sorting 3 elements. This process will be repeated for all data sets giving the average number of operations taken by each algorithm for sorting n elements, for several values of n.

3.2 Comparing the results with that of previous researchers

The results from this project, of average number of operations for each algorithm for various input sizes, are then compared with the results of previous researchers. Based on the results of this project, it is attempted to construct an approximate formula to represent the average number of operations taken by an algorithm to sort n elements for any value of n. For example, such a formula already exists for Bubble sort, the formula is number of operations needed is $n(n-1)/2$ for sorting n elements [3].

3.3 Comparing the algorithms among themselves

The average number of operations taken by various algorithms for different input sizes is used to compare the sorting algorithms among themselves. This is done to find out which algorithm, among the 4 sorting algorithms, is the best in terms of number of operations for a specific input size.

4 Numerical Results

4.1 Bubble sort vs Insertion sort

From the results of this project, it is observed that Insertion sort takes about 50% to 52% of the value of $n(n-1)/2$ for average number of operations, where n denotes the input size (number of elements to sort) and takes the values n = 100, 1000, 10000, 100000, 1000000. It can be concluded that Insertion sort takes approximately $n(n-1)/4$ as average number of operations for input size n. As proved theoretically by previous researchers, this project also observes that

Bubble sort takes $n(n-1)/2$ as average number of operations [3]. This shows that Insertion sort takes approximately half the average number of operations than that taken by Bubble sort for the same input size n. Since smaller value of number of operation denotes smaller running time or better performance. Thus, based on the number of operations, Insertion sort is approximately twice as better than Bubble sort sorting algorithm.

Previous researchers identified $O(n^2)$ as the number of operations taken by either of Bubble sort as well as Insertion sort for input size n tending to infinity and the worst running time case. While, this holds true for practical cases as well, the value of $O(n^2)$ is insufficient to establish the exact differences in running time of Insertion sort and Bubble sort. In this research, using the computational value of the average number of operations taken in the practical n case, the difference between Insertion sort and Bubble sort is practically established.

4.2 Merge sort vs Quick sort

The following table 1 illustrates the average number of operations taken by Merge sort and Quick sort for sorting N (or n) elements:

Table 1: Merge sort and Quick sort results

N or n (number of elements to sort)	Merge sort (MS) Average number of operations (rounded up to 2 decimal places)	Quick sort (QS) Average number of operations (rounded up to 2 decimal places)	Previous researchers $nlog(n)$ (rounded up to 2 decimal places) logarithmic to base 2	QS/MS (Average number of operations)
100	97.99 (14.75%)	651.70 (98.09%)	664.39	6.65
1000	997.92 (10.01%)	11023.47 (110.61%)	9965.78	11.05
10000	9997.84 (7.52%)	156137.49 (117.51%)	132877.12	15.62
100000	99997.96 (6.02%)	2022887.18 (121.79%)	1660964.05	20.23
1000000	999998.01 (5.02%)	24557640.44 (123.21%)	19931568.57	24.56

Previous researchers have identified $O(nlog(n))$ as the number of operations taken by either of Merge sort and Quick sort [3]. Thus, in this project, as it can be seen from the table 1, $nlog(n)$ is used as a standard value for comparing the practical values of number of operations taken by Merge sort and Quick sort. The percentage values in Merge sort and Quick sort columns are with respect to $nlog(n)$ in the previous researchers column. It denotes the variation of the practical value of number of operations taken by Merge sort and Quick sort with respect to $nlog(n)$. The smaller the value of number of operations, the smaller the running time, which means better performance by the algorithm.

From the above table 1, since the percentage value of Merge sort keeps decreasing from 14.75% to 5.02% as N increases from 100 to 1000000, it can be concluded that the performance of Merge sort keeps improving with respect to $nlog(n)$ as the value of input size (N or n) increases. Similarly, since the percentage value of Quick sort keeps increasing from 98.09% to 123.21% as N increases from 100 to 1000000, it can be concluded that the performance of Quick sort keeps decreasing with respect to $nlog(n)$ as the value of the input size (N or n) increases.

Similarly, from the last column of the above table 1, the ratio of number of operations taken by Quick sort with respect to Merge sort keeps increasing from 6.65 to 24.56 as N increases from 100 to 1000000. This shows that there is an enormous improvement in the performance of Merge sort with respect to that of Quick sort with the increase in the input size. Based on these results, Merge sort performs better that Quick sort when only the number of operations taken to sort is considered for comparison.

4.3 Bubble sort & Insertion sort vs Merge sort & Quick sort

As it can be seen from the sections 4.1 and 4.2, Bubble sort and Insertion sort have an approximate running time of $n(n-1)/2$ and $n(n-1)/4$, whereas Merge sort and Quick sort have an approximate running time around $nlog(n) * 0.1$ and $nlog(n) * 1.1$. Since the growth of the logarithmic function is slower than the growth of the polynomial function, and the value of $n(n-1)/4$ is bigger than the value of $nlog(n) * 1.1$ for $n = 100$, the average running times of Bubble sort and Insertion sort are greater than the average running times of Merge sort and Quick sort. Overall, only considering the number of operations as the running time for comparison, the performances of Merge sort and Quick sort are better than the performances of Bubble sort and Insertion sort based on the results from this project, as well the the results from previous researches [3].

5 Conclusion

5.1 Brief summary

This project practically computed the average running time of Bubble sort, Insertion sort, Merge sort and Quick sort sorting algorithms for different input sizes ranging from 100 to 1 million. The results of this project show that by comparing the algorithms in the practical case of finite input size and average running time, we can get better insights on which algorithm is better. The results are more detailed than the theoretical case of n tending to infinity and worst running time. For example, while considering the number of operations as the running time for comparison of algorithms, Insertion sort seems twice as better than Bubble sort, Merge sort seems better than Quick sort, Merge sort and Quick sort seem much better than Bubble sort and Insertion sort. Among Bubble sort, Insertion sort, Merge sort and Quick sort sorting algorithms, Merge sort seems to be the best in terms of number of operations needed to sort elements. However, it must be noted that these results are valid on the data sets randomly generated in this project and the results may vary based on the data needed to be sorted. Overall, more insights can be obtained from the practical method used in this project to computationally compare sorting algorithms considering average case than the theoretical method of Asymptotic analysis considering the worst case used in researches in this field [3].

5.2 Limitations

There can be other case specific ways to implement algorithms which would affect their running time. Similarly, there can be different results obtained based on the type of data, for example if the data is already well sorted or sorted in parts, then Insertion sort might work much better than the other three algorithms considered in this project. Due to this, python uses Timsort sorting algorithm as default, which is a combination of Merge sort and Insertion sort sorting algorithms [7].

In some cases, parameters other than number of operations can also matter, for example, the space complexity or the space needed by an algorithm. If the space complexity matters more than number of operations in some projects, then Quick sort is considered better than Merge sort by previous researchers [5].

5.3 Future implications

Based on the results from this project, if we just consider the number of operations for comparing sorting algorithms then among the 4 sorting algorithms, the increasing order of performance is Bubble sort, Insertion sort, Quick sort and Merge sort, with Merge sort found to be the best sorting algorithm.

In future, the method of computational comparison of algorithms used by this research can be used to compare other algorithms and provide a better understanding of the differences in performances of algorithms.

References

[1] `https://www8.cs.umu.se/kurser/TDBA77/VT06/algorithms/BOOK/BOOK/NODE30.HTM`

[2] K. Ali, A Comparative Study of Well Known Sorting Algorithms, Retrieved from `http://www.ijarcs.info/index.php/Ijarcs/article/view/2903/2886`

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, & C. Stein, *Introduction to Algorithms, Third Edition*, The MIT Press, 2009.

[4] `https://www.geeksforgeeks.org/bubble-sort/`

[5] `https://www.geeksforgeeks.org/quick-sort-vs-merge-sort/`

[6] `https://www.geeksforgeeks.org/sorting-algorithms/`

[7] `https://www.geeksforgeeks.org/timsort/`