Vol. 1, Issue 1, January 2019, pp. 1 - 18
Ayodele & Oluwade (2019). A Comparative Analysis of Quick, Merge and Insertion Sort Algorithms using Three Programming Languages I: Execution Time Analysis

A Comparative Analysis of Quick, Merge and Insertion Sort Algorithms using Three Programming Languages I: Execution Time Analysis

Oluwakemi Sade Ayodele
Department of Computer Science, Kogi State Polytechnic,
Lokoja, Nigeria
Email: Kemtemmy2009@gmail.com
&
Bamidele Oluwade
Dewade Systems Consult, Nigeria
Email: deleoluwade@dewadeconsult.com

ABSTRACT

In recent years, technological advancement and continuous improvement in computing devices has led to more study in the area of Green Computing with much emphasis on processing time and energy consumption. The study has also been precipitated by wide coverage of Internet of things (IoT), as well as increased global acceptability of mobile and wireless devices (smart device) which are smaller, smarter and ubiquitous. The aim of this paper is to carry out a comparative analysis of three sorting algorithms, namely Quick. Merge, Insertion sort, implemented in three programming languages (C, Java, Python) using execution time. The comparison of the Execution time of these three sorting algorithms was based on programming Language, data size and algorithm implementation style (Iterative and Recursive). Time stamp was used to capture the execution time of the sorting algorithm. It was noted that the data size, programming language used and algorithm implementation styles affect the execution time. Use of scripting language has clear drawbacks in terms of execution time which should be taken into consideration. The experiments carried out show that the way software is written (algorithm implementation style) and the programming language used are both very important determinants of the execution time of that software. Therefore, this paper provide information on choice of sorting algorithm type and its algorithm implementation style. This is done in order to minimize execution time of an algorithm in the current period of explosive growth in the use of smart phones and hand held devices which run majorly on battery life. This gives developers knowledge on time efficiency in software leading to choosing some codes over others based on their time performance. The paper is on data management.

Keywords: *Time, Time Efficiency, Execution time, Sorting, Comparative Analysis*

Reference Format:

Ayodele, Oluwakemi Sade and Oluwade, Bamidele (2019), A Comparative Analysis of Quick, Merge and Insertion Sort Algorithms Using Three Programming Languages I: Execution Time Analysis, *Afr. J. MIS*, Vol.1, Issue 1, pp. 1 - 18.

1. INTRODUCTION

In recent time, there is a global growing awareness of the environmental impact of computing with the emergence of a new field called Green computing. Green computing can be defined as the study of designing, manufacturing/engineering, using and disposing of computing devices efficiently and effectively in a way that reduces their environmental impact (www.technopedia.com; https://en.wikipedia.org/wiki/Green_computing).

Recent emphasis has been on green computing which focuses mainly on developing energy and power efficient devices and the use of non-toxic materials and minimizing e-waste in such design (http://trese.ewi.utwente.ni/workshops/SEAGC). Therefore, more focus has been on the hardware and less on the software aspect of green computing and green IT not minding the fact that green computing is the study and practice of using in totality computing resources efficiently. Green--ness' in the software is an emerging quality attribute that should be taken into consideration through all the stages/phases of software development process from planning to maintenance (www-03.ibm.com).

Therefore understanding the execution time and energy usage/ consumption of an algorithm has become a major issue in determining and improving the efficiency of any algorithm (http://www1.eere.energy.gov). This is particularly so because estimating the execution time, which directly affects the energy consumption, is a laborious task. The higher the execution time of an algorithm, the higher the energy consumption which directly affect the environment.

Furthermore, one of the fundamental problems of today's world is the explosive use of computing devices like the lap top, Smartphone, notebooks and other handheld devices which are typically battery driven, and the increase in price of electricity and at times non availability of electricity, so effort must be made in improving the energy efficiency of both hardware and software designs. Therefore there is an urgent need to look into reducing the execution time of an algorithm and also choosing the most appropriate algorithm depending on the use of such software.

The paper considers factors which affect the execution time of (sorting) algorithms, then compared and contrasted the results. The sorting algorithms which were experimented on are Quick, Merge and Insertion sort algorithms, while the programming languages used for the implementation are C, Java and Python. The two different algorithmic styles used are iterative algorithm and recursive algorithm.

2. RELATED WORK

(Totla, 2016) tried to improve upon execution time of the Bubble Sort algorithm by implementing the algorithm using a new algorithm. An extensive analysis has been done by author on the new algorithm and the algorithm has been compared with the traditional methods of ―Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort. Were implemented it in C language on GCC complier on Linux operating system implemented in C++ programming languages and tested for the random sequence input of length 10000, 20000, 30000. Observations have been obtained on comparing this new approach with the existing approaches of All Sorts. All algorithms were tested on random data of various ranges from small to large. It has been observed that the new approach has given efficient results in terms of execution time. Through the experimental observations that the new algorithm given in the paper is better than Selection Sort, Insertion Sort, Merge Sort, and Bubble Sort except Quick Sort for larger inputs. However, the data size used is not large enough to give a better comparison of these sorting algorithms.

(Deepthi & Birunda Antoinette Mary, 2018) conducted experiments to study how different sorting algorithms have an impact on energy consumption using C language implementation. It was discovered that both time and energy have an impact on the efficiency of these sorting algorithms, with quick sort, merge sort and shell sort found to be in the same range of time and energy consumption, followed by insertion and selection sort which is far better than Bubble sort. However, the implementation is only the C language with a non-varying small data size of 10,000. The effect of sorting large data sizes was not considered.

Rivoire et. al. (2007) proposed an external sorting benchmark for evaluating the energy efficiency of sorting for a wide range of system but focused more on hardware rather than the software domain.

Bunse et. al. (2009), in their study, focused on sorting algorithms, which are not only used directly by the user of

a device but also very often implicitly by other algorithms and show that different sorting algorithms have different energy consumptions however this work is limited to just comparison with Sorting algorithm and programming language.

(Carroll & Heiser, 2010) performed a detailed analysis of energy consumption of a smartphone, based on measurements of a physical device. How the different components of the device contribute to overall power consumption was analysed. A model of the energy consumption for different usage scenarios was developed, and showed how these translate into overall energy consumption and battery life under a number of usage patterns. The measurement of energy consumption was on smartform as a hardware but not on the installed software.

(Jain, Molnar & Ramzan, 2005) defined an algorithm complexity model theoretically for energy. The complexity model gave analytical explanation using staple algorithms such as sorting and random number generation to verify the model. However, the effect of program structure on the power and energy consumption was not taken into consideration.

(Rashid, Ardito & Torchiano, 2015) carried out an experiment on an ARM based device, measuring the energy consumption of different sorting algorithms implemented in different programming languages to analyze the energy consumption of algorithms implementation. However, the impact of program structure on energy consumption was not evaluated.

(Elkahlout & Maghari, 2017) conducted a comparative study to evaluate the performance of three algorithms; comb, cocktail and count sorting algorithms in terms of execution time. Java programming was used to implement the algorithms using numeric data on the same platform conditions. Among the three algorithms, it was found out that the cocktail algorithm has the shortest execution time; while counting sort comes in the second order. Furthermore, Comb comes in the last order in term of execution time. In the work, different sorting algorithms were used, only Java was used for comparison and energy consumptions of the sorting algorithms were not determined.

(Aremu et. al., 2016) presented a comparative study of three sorting algorithms, namely median, heap, and quick sort using CPU time and memory space as performance index to adopt the most efficient sorting technique in the development of job scheduler for grid computing community. These were implemented with C-language; while the profile of each technique was obtained with G-profiler. However, energy efficiency of these sorting algorithms was not considered.

(Ali et. al., 2016) studied the working of five sorting algorithms (bubble, selection, insertion, merge, quick sorts) and compared them on the basis of computational complexities, usage of memory and other computer resources, stability, number of swaps and number of comparisons. Quick sort was said to be the faster sorting algorithm while bubble sort is the slowest algorithm. Nothing was discussed on energy efficiency as parameter for comparing these sorting algorithms.

3. METHODOLOGY

3.1 Research Questions (RQ)

The following research questions have been formulated for this study to help the stated objectives above:

RQ1: Does Data Size have effect on Execution Time?
RQ2: Does Programming languages used have effect on Execution time of an Algorithm?
RQ3: Does Algorithm implementation style (Iterative and Recursive) have effect on Execution time of an Algorithm?

3.2 Implementation/Experimentations

3.2.1 Algorithm Implementation Languages

The three sorting algorithms (Quick, Merge and Insertion Sorts) were run using three different programming languages (Java, C and Python Languages). This is to:

i. Determine how implementation language affects execution time of a sorting algorithm.
ii. Determine complexities of the sorting algorithms when changing the implementation language.

Merge, Quick and Insertion sort algorithms were implemented using Java, C and Python Programming Languages. Merge and Quick sort were implemented using both recursive and iterative programming styles while Insertion was implemented using only iterative programming structure. The choices of programming languages were categorized based on:

- Virtual machine based languages: Java (compiled and interpreted language)
- Native Languages: C(Compiled language)
- Scripting Languages: Python (Interpreted language)

The Experiments were conducted on the same computer configuration, and running Window 7. The sorting algorithms were run on a laptop with the following configuration, HP 630 Notebook PC, 4GB RAM, Intel(R) Core(TM) i3 @2.53 processor.

3.2.2 Algorithm implementation styles

Two different algorithm implementation styles were used on the same Algorithm for the three Sorting Algorithms (Quick, Merge and Insertion) to compare its energy consumption:

i. Iterative Algorithm
ii. Recursive Algorithm

Recursion and iteration both repeatedly executes the set of instructions. Recursion is when a statement in a function calls itself repeatedly. The iteration is when a loop repeatedly executes until the controlling condition becomes false. The key difference between recursion and iteration is that recursion is a mechanism to call a function call a within the same function while iteration is to execute a set of instructions repeatedly until the given condition is true. Recursion and Iteration are major techniques for developing algorithms and building software applications.

3.3 Data Size

In the algorithm execution, five different integer elements are sorted (100,000; 200,000; 300,000; 400, 000 and 500, 000), See table 3.2. The data was randomly generated by importing random function from the language(s) libraries. The Execution time for each sorting algorithms for each data sizes were measured. To reduce measurement error, each data size being considered was executed five times, the average captured and recorded for use.

3.4 Measurement

Time Stamp

Time stamp was placed directly above the called function containing the sorting algorithm and another time stamp was placed directly below the called sorting function. This is to ensure that the execution time captured is solely for the sorting algorithm. Execution time was derived by subtracting the start time from end time.

Execution Time

The Execution Time T is the time that it takes for an algorithm to execute. Execution time for sorting was measured using system clock imported from programming language's libraries. The algorithm for execution time is as shown in Figure 3.1.

System. cuurentTimeMillis(), time(Null) and datatime, timeit.default_timer() were time functions used for Java, C and Python Programming Languages respectively. Execution Time was derived by subtracting Start time from end time. Measured times were in seconds.

3.5 Software Setup

Merge sort, Insertion sort and Quick sort algorithms were implemented using C, Java and Python. Dev C++ and Net beans Integrated Development Environments were used for C and Java programming languages respectively. Python 3.6.0 was used in the implementation. The sorting algorithms were implemented using recursive and iterative styles. Each of the algorithms was implemented in three programming languages, Merge sort and Quick sort algorithms were implemented using the iterative and recursive version for each programming languages. Data sizes ranges from one hundred thousand (100,000) to five hundred thousand (500,000) at an interval of one hundred thousand. The integers for each data size were randomly generated using the rand() function. The sorting algorithms were placed in functions and classes and called in the main function and class. Time stamp was placed directly above the called function containing the sorting algorithm and another time stamp was placed directly below the called sorting function. This is to ensure that the execution time captured is solely for the sorting algorithm. Execution time was derived by subtracting the start time from end time.

Immediately, data size was entered and ok was clicked to start sorting. The execution time was displayed in millisecond. The experiment was repeated five times for each algorithm using one data size and one programming structure (iterative or recursive). The average of the five experiments was recorded as the value for the data size.

Microsoft Excel was used to compute the Average Execution Time (Table 3.1 is a screen shot)

4. RESULTS

The results in the paper have been tabulated and can be seen in Table 4.1 – Table 4.8. These results are on:

(i) Comparison of the average execution time of iterative quicksort algorithm implementations in C, Java and Python programming languages (Table 4.1).
(ii) Comparison of the average execution time of iterative mergesort algorithm implementations in C, Java and Python programming languages (Table 4.2).
(iii) Comparison of the average execution time of iterative insertionsort algorithm implementations in C++, Java and Python programming languages (Table 4.3).
(iv) Comparison of the average execution time of recursive quicksort algorithm implementations in C, Java and Python programming languages (Table 4.4).
(v) Comparison of the average execution time of recursive mergesort algorithm implementations in C, Java and Python programming languages (Table 4.5).
(vi) Comparison of the average execution time of iterative and recursive quick, merge and insertion sort algorithm implementation in C (Table 4.6).
(vii) Comparison of the average execution time of iterative and recursive quick, merge and insertion sort algorithm implementation in Java (Table 4.7).
(viii) Comparison of the average execution time of iterative and recursive quick, merge and insertion sort algorithm implementation in Python (Table 4.8).

5. DISCUSSION OF RESULTS

5.1 Impact of Data Size on Execution Time (Execution Time versus Data Size)

From Figure 4.1 to Figure 4.8 data size and average execution time of Quick sort, Merge Sort and Insertion for the Iterative and recursive Algorithm implementation styles. As the data size increases from 100,000 to 500,000 at a range of 100,000, the execution time also increases.

Checking the trend in the shape of the graph, there is a continuous linear flow of the graphs in figures 4.1 through 4.8 which implies that as the data size increases the execution time also increases.
Therefore the following conclusions were made:

1. The Average Execution Time increases with increase in data size
2. Irrespective of the Algorithm implementation style used, statement in (1) above is upheld.
3. Irrespective of the Programming language used, statement in (1) above is upheld.
4. Average Execution time of recursive implementations of all the sorting Algorithms in all programming languages used is higher than its equivalent iterative implementations.

It is worth noting that Sorting Algorithms have looping structures. If the loops iterate n times, then the time for all n iterations is C1. N where C1 indicates number of times the computation takes place in one loop iteration. The value of C1 is dependent on programming language used, compiler or interpreter used in translating source code to machine code, programming structure, algorithm implementation style and other factors. While the C1 has negligible effect on computing the asymptotic complexity, the impact is evident in actual implementation of the algorithms.

The impact of C1 on the execution time for each algorithm implemented accounts for differences in behavior. Same algorithm implemented with different programming languages and algorithm implementation styles have slight differences in execution time.

Obviously, C1 is not negligible but key factors to be considered in algorithm implementation.

However, it is interesting to see the differences in execution time between iterative and recursive variants of the same algorithms. These lead to the assumption that the use of recursion might improve performance, but will definitively increase execution time. An Iterative algorithm will be faster than the Recursive algorithm because of overheads like calling functions and registering stacks repeatedly. Many times the recursive algorithms are not efficient as they take more space and time. The results show that the iterative version of the Sorting Algorithms used are more energy efficient than the recursive version.

5.2 Impact of Programming Languages
Observations:
We consider Tables 4.1 - 4.8 and Figure 4.1 - figure 4.8. For instance, Figure 4.1 shows the graph of comparison of average execution time of iterative Quick Sort Algorithms Implementations in C, Java and Python programming languages. Python programming language has the highest execution time followed by C language and then Java language. The difference in the execution time of python language both to Java and C languages is very high.

Therefore, six implementations from three groups of programming languages were analyzed; figures 6 – 8 show the results of the experiments. The following were observed:

1. The execution time by same algorithm varies from one programming language to the other.
2. The execution time of the scripting language (Python) is significantly higher than the virtual based language (Java) and the native code language (C).

It was noted that the high execution time of the scripting language can be attributed to the fact that there is the need to interpret and then execute the algorithm. This additional step clearly has a higher value in terms of Execution time.

Programming languages are diverse in terms of design and specifications. The experiment on programming languages with this diversity gives us hints as to why execution time varies from one programming language to another.

5.3 Impact of Algorithm implementation Style (Iterative and Recursive)
Observations:
From figure 4.8, the Insertion sort algorithm using the Iterative implementation style, implemented in python programming language, has the highest execution time. As the data size increases, the execution time also increases. It should be noted as earlier on stated that insertion sort algorithm was only implemented using the iterative algorithm implantation style. In all other sorting algorithms used implemented in the three programming languages, the execution time of the recursive version is higher than the iterative.

Checking the trend in the shape of the graph, the following were observed:
1. The execution time of an algorithm is dependent on the algorithm implementation style.

2. Generally, iterative implementation of sorting algorithms are more energy efficient than equivalent recursive implementation.

5.4 Relationship between Execution time, Data Size and Programming language used.
From Figures 4.1- 4.8, it can be seen that data size, algorithm implementation Style and programming language affects the execution time of an Algorithm. It can be noted that the same sorting algorithm has different execution time with varying data sizes, when implemented in different programming language or/and when implemented in different algorithm implementation styles. The higher the data size the higher the time taken for sorting. Varying the parameters values (Data size, algorithm implementation style, programming language) impacts the time of execution of an algorithm with different evolution patterns based on which parameters are varied.

6. CONCLUSION

The execution time of different sorting algorithms were analysed via relative implementation in three distinct languages, and two algorithm implementation styles over an average of five data sizes. The data size, programming language, the implementation algorithm style are factors that affect execution time of any software. Varying the parameters values (Data size, Programming language and algorithm implementation style impact the time complexity with different evolution patterns, based on the parameters are varied.

This paper therefore provides the basic information in choosing a specific sorting implementation over another in order to minimize the execution time of such an algorithm, most especially in any handheld battery driven devices.

REFERENCES

[1] http://www.techopedia.com>definition. Retrieved 13/01/2018

[2] (Call for paper on) Green Computing, In Software Engineering Aspects of Green Computing In the *28th Annual ACM Symposium on Applied Computing*, March 18-22, 2013, Coimbra, Portugal. [online]; http://trese.ewi.utwente.ni/workshops/SEAGC. Retrieved 07/11/2016

[3] Data center energy consumption trends. http://www1.eere.energy.gov/Femp/Program/dc_energy_consumption.html. Retrieved 11/11/2017

[4] Green IT: Why Mid-Size Companies are investing Now- IBM. [online]; www-03.ibm.com/press/attachments/GreenIT-final-mar4.pdf. Retrieved 14/11/2017

[5] Bunse, C., Höpfner, H., Roychoudhury, S., Mansour, E (2009). Energy Efficient Data Sorting Using Standard Sorting Algorithms. In *International Conference on Software and Data Technologies* ICSOFT 2009: Software and Data Technologies pp. 247-260 [online] https://link.springer.com/chapter/10.1007/978-3-642-20116-5_19. Retrieved on 4/11/2017

[6] Rivoire, S; Sharh, M.A; Ranganathan, P; and Kozyrakis, C. (2007), "Joulesort: A balanced energy efficiency benchmark", in *Proceedings of the 2007 ACM SIGMOD international conference on management of Data*, ser.SIGMOD '07. New York, NY, USA: ACM, 2007, pp. 365-376. [Online]. Available: http://doi.acm.org/10.1145/1247480.1247522. Retrieved 18/11/2016

[7] Bunse, Christian; Hopfner, Hagen; Mansour, Essam and Roychoudhury, Suman (2009),"Exploring the Energy Consumption of Data Sorting Algorithms in Embedded and Mobile Environments" In *Mobile data Management Systems, Services and Middleware, 2009. MDM '09 Tenth International Conference* on May 2009, pp 600-607. Retrieved [Online] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.543.8109&rep=rep1&type=pdf on 29/09/2017.

[8] Carroll, Aaron and Heiser, Gernot (2010). "An analysis of power consumption in a smartphone", In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference, USENIXATC'10*, pages 21–21, Berkeley, CA, USA, 2010.USENIX Association, pp 4,36. Retrieved [online] https://ori-nuxeo.univlille1.fr/nuxeo/site/esupversions/5dd79154-0514-469fbd60-899cf9f57035

[9] Jain, R; Molnar, D; and Ramzan, Z. (2005). "Towards understanding algorithmic factors affecting energy consumption: Switching complexity, randomness and preliminary experiments," In *Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing*, ser. DIALM-POMC '05, New York, NY, USA: ACM 2005, pp 70-79, [online]. https://pdfs.semanticscholar.org/.../249c47bdf85538b03b314c3f0fb70b0518c4.pdf. Retrieved 15/01/2017.

[10] Rashid, M; Ardito, L; Torchiano, M. (2015). "Energy Consumption Analysis of Algorithm Implementations", In *2015 proceedings of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (ESEM). [online]. Available: http://doi.ieee/computersociety.org/10.1109/esem.2015.7321198. Retrieved 04/11/2016

[11] Elkahlout, Ahmad, and Maghari, Ashraf Y. A. (2017). "A comparative Study of Sorting Algorithms Comb, Cocktail and Counting Sorting", *International Research Journal of Engineering and Technology* (IRJET) e-Volume:04, Issue:01 Jan 2017; https://www.researchgate.net/publication/314753240_A_comparative_Study_of_Sorting_Algorithms_Comb_Cocktail_and_Counting_Sorting [accessed Nov 18 2017].

[12] https://en.wikipedia.org/wiki/Green computing. Retrieved on 2/11/2017

[13] Deepthi T; Birunda Antoinette Mary, J. (2018). "Time and Energy Efficiency: A Comparative Study of Sorting Algorithms Implemented in C", *International Conference on Advancements in Computing Technologies* (IJFRCSCE) -ICACT 2018, Volume 4, Issue 2, 25–27; http://www.ijfrcsce.org. Retrieved 20/5/2018

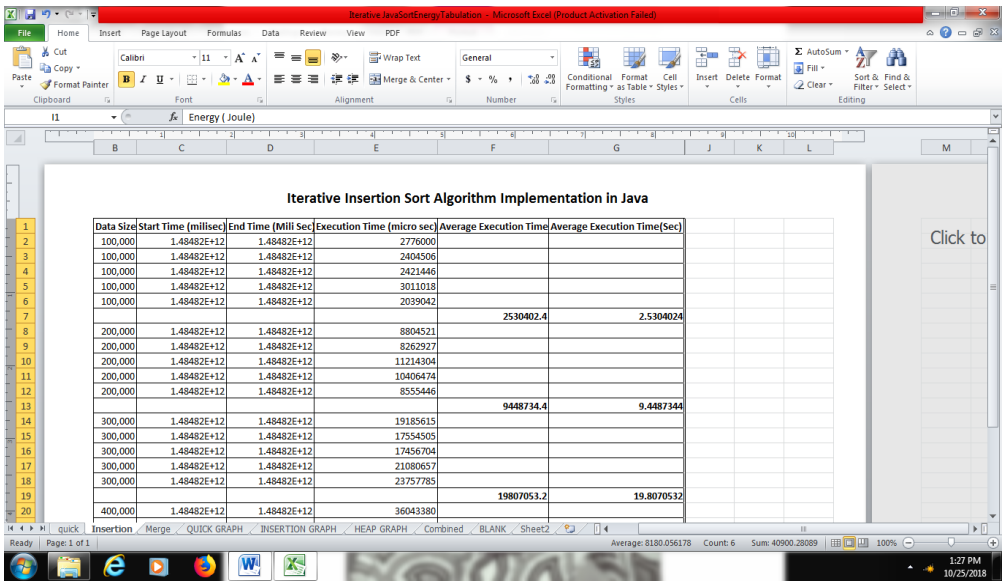
[14]Aremu, D.R; Adesina, O.O; Makinde, O. E; Ajibola, O; and Agbo-Ajala, O. O. (2016). "A Comparative Study of Sorting Algorithms", *African Journal of Computing & ICT*, Vol 6, No. 5, pp 199-206.

[15] Ali, Waqas; Islam, Tahir; Rehman, Habib Ur; Ahmed, Izaz; Khan, Muneeb; and Mahmood, Amna (2016) "Comparison of Different Sorting Algorithms", *International Journal of Advanced Research in Computer Science and Electronics Engineering* (IJARCSEE), Volume 5, Issue 7, [Online] ijarcsee.org/index.php/IJARCSEE/article/view/554/526.

[16] Totla, Jyoti (2016), "Review on Execution Time of Sorting Algorithms - A Comparative Study", International *Journal of Computer Science and Mobile Computing*, Vol. 5, Issue.11, pp. 158-166.

```
Start_Time        Invoke _System_clock
          Call_sortingAlgorithm_class/method
          End_Time   ←  Invoke_System_Clock
          Execution_Time = End_Time -  Start_
Time
```

Figure 3.1: The algorithm for execution time



Iterative Insertion Sort Algorithm Implementation in Java

Data Size	Start Time (milisec)	End Time (Mili Sec)	Execution Time (micro sec)	Average Execution Time	Average Execution Time(Sec)
100,000	1.48482E+12	1.48482E+12	2776000		
100,000	1.48482E+12	1.48482E+12	2404506		
100,000	1.48482E+12	1.48482E+12	2421446		
100,000	1.48482E+12	1.48482E+12	3011018		
100,000	1.48482E+12	1.48482E+12	2039042		
				2530402.4	2.5304024
200,000	1.48482E+12	1.48482E+12	8804521		
200,000	1.48482E+12	1.48482E+12	8262927		
200,000	1.48482E+12	1.48482E+12	11214304		
200,000	1.48482E+12	1.48482E+12	10406474		
200,000	1.48482E+12	1.48482E+12	8555446		
				9448734.4	9.4487344
300,000	1.48482E+12	1.48482E+12	19185615		
300,000	1.48482E+12	1.48482E+12	17554505		
300,000	1.48482E+12	1.48482E+12	17456704		
300,000	1.48482E+12	1.48482E+12	21080657		
300,000	1.48482E+12	1.48482E+12	23757785		
				19807053.2	19.8070532
400,000	1.48482E+12	1.48482E+12	36043380		

Table 3.1: Computation of the Average Execution Time of Iterative Insertion Sort Algorithm

Table 4.1: COMPARISON OF AVERAGE EXECUTION TIME OF ITERATIVE QUICK SORT ALGORITHM IMPLEMENTATIONS IN C, JAVA AND PYTHON

DATA SIZE('000)	C	JAVA	PYTHON
100	0.0594	0.056	1.7203154
200	0.125	0.0656	3.6812568
300	0.1372	0.075	5.6833226
400	0.2652	0.0874	7.8595448
500	0.3902	0.103	9.3320522

Table 4.2: COMPARISON OF AVERAGE EXECUTION TIME OF ITERATIVE MERGE SORT ALGORITHM IMPLEMENTATIONS IN C, JAVA AND PYTHON

DATA SIZE (000)	C	JAVA	PYTHON
100	0.0914	0.0532	2.709
200	0.1892	0.0748	4.8174
300	0.2406	0.12	7.2094
400	0.3032	0.1344	9.7769
500	0.4388	0.148	12.4384

Table 4.3: COMPARISON OF AVERAGE EXECUTION TIME OF ITERATIVE INSERTION SORT ALGORITHM IMPLEMENTATIONS IN C++, JAVA AND PYTHON

DATA SIZE (000)	JAVA	C	PYTHON
100	2.530402	15.2446	2134.752
200	9.448734	59.088	8799.9277
300	19.8070532	125.8078	18235.92516
400	32.7248226	227.1638	32938.84238
500	49.9475218	346.2752	57156.7263

Table 4.4: COMPARISON OF AVERAGE EXECUTION TIME OF RECURSIVE QUICK SORT ALGORITHM IMPLEMENTATIONS IN C, JAVA AND PYTHON

DATA SIZE('000)	C	JAVA	PYTHON
100	0.0556	0.068961	1.52022
200	0.0834	0.0926188	3.450263
300	0.1134	0.0941966	5.030150
400	0.16	0.1246214	7.149799
500	0.2094	0.1377148	7.606842

Table 4.5: COMPARISON OF AVERAGE EXECUTION TIME OF RECURSIVE MERGE SORT ALGORITHM IMPLEMENTATIONS IN C, JAVA AND PYTHON

DATA SIZE(‘000)	C	JAVA	PYTHON
100	0.042	14.5594	15.2446
200	0.0858	39.228	59.088
300	0.136	86.9696	125.8078
400	0.1956	156.3012	227.1638
500	0.2374	242.195	346.272

Table 4.6**:** COMPARISON OF AVERAGE EXECUTION TIME OF ITERATIVE AND RECURSIVE QUICK, MERGE AND INSERTION SORT ALGORITHMS IMPLEMENTATIONS IN C

		Average Execution Time of three Iterative and Recursive sorting Algorithms implemented in C			
Data Size ('000)	Iterative Insertion	Iterative MergeSort	Recursive MergeSort	Iterative Quick sort	Recursive QuickSort
100	15.2446	0.0914	0.042	0.0594	0.0556
200	59.088	0.1892	0.0858	0.125	0.0834
300	125.8078	0.2406	0.136	0.1372	0.1134
400	227.1638	0.3032	0.1956	0.2652	0.16
500	346.2752	0.4388	0.2374	0.3902	0.2094

Table 4.7: COMPARISON OF AVERAGE EXECUTION TIME OF ITERATIVE AND RECURSIVE QUICK, MERGE AND INSERTION SORT ALGORITHMS IMPLEMENTATIONS IN JAVA

		Average Execution Time of three Iterative and Recursive sorting Algorithms implemented in Java			
Data Size ('000)	Iterative Insertion	Iterative MergeSort	Recursive MergeSort	Iterative Quick sort	Recursive QuickSort
100	2.5304024	0.0532	14.5594	0.056	0.068961
200	9.4487344	0.0748	39.228	0.0656	0.0926188
300	19.8070532	0.12	86.9696	0.075	0.0941966
400	32.7248226	0.1344	156.3012	0.0874	0.1246214
500	49.9475218	0.148	242.195	0.103	0.1377148

Table 4.8: COMPARISON OF AVERAGE EXECUTION TIME OF ITERATIVE AND RECURSIVE QUICK, MERGE AND INSERTION SORT ALGORITHMS IMPLEMENTATIONS IN PYTHON

		Average Execution Time of three Iterative and Recursive sorting Algorithms implemented in Python			
Data Size ('000)	Iterative Insertion	Iterative MergeSort	Recursive MergeSort	Iterative Quick sort	Recursive QuickSort
100	2134.752	2.709	15.2446	1.7203154	1.52022
200	8799.9277	4.8174	59.088	3.6812568	3.450263
300	18235.92516	7.2094	125.8078	5.6833226	5.030150
400	32938.84238	9.7769	227.1638	7.8595448	7.149799
500	57156.7263	12.4384	346.272	9.3320522	7.606842

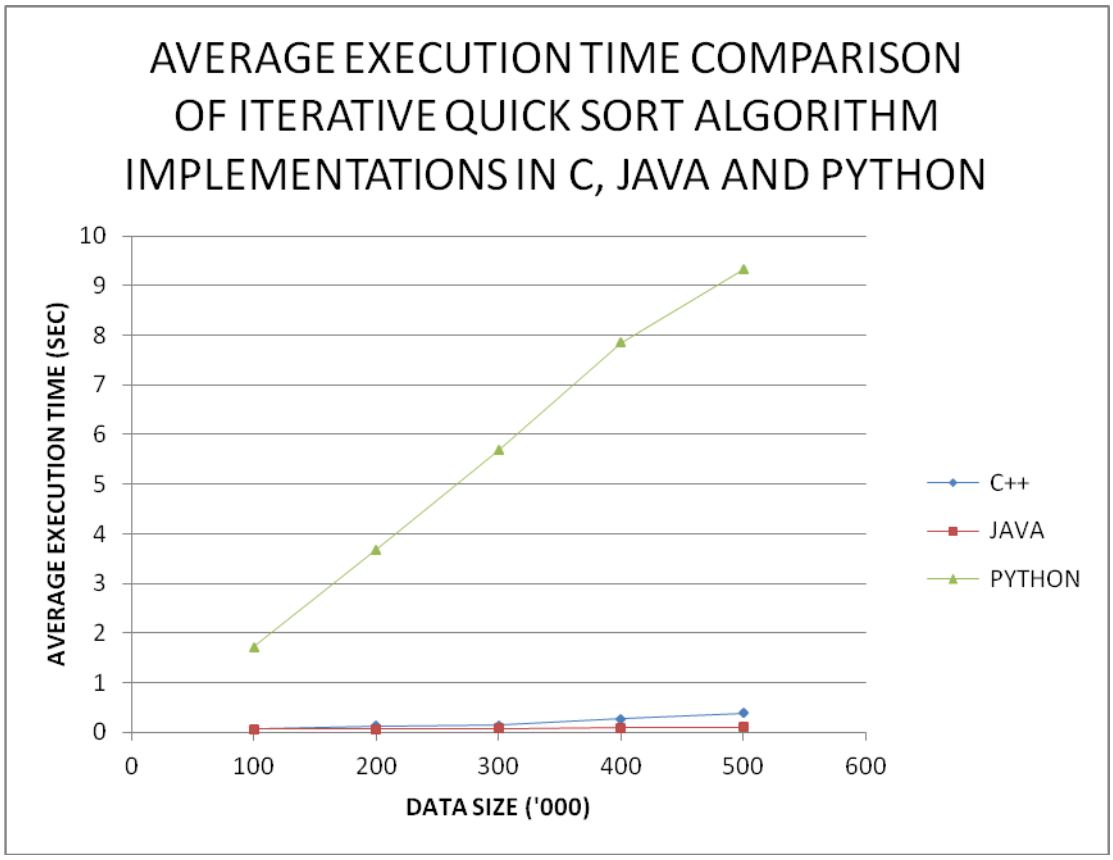


Figure 4.1: Average Execution Time Comparison 0f iterative Quick Sort Algorithms Implementations in C, Java and Python

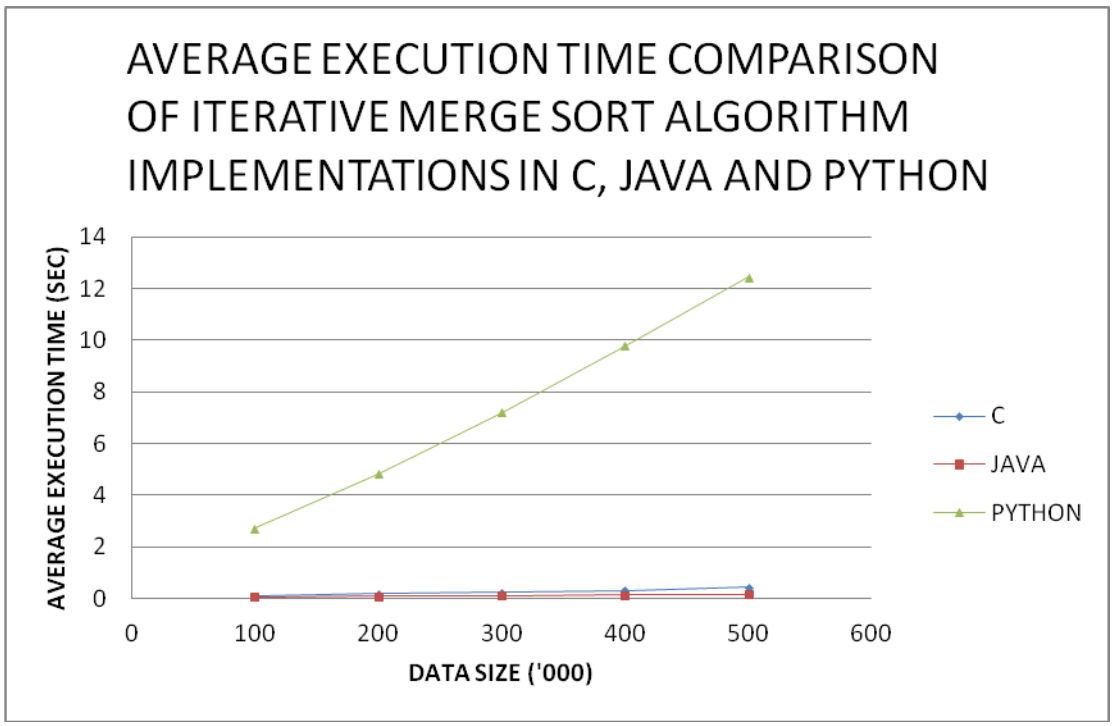


Figure 4.2: Average Execution Time Comparisons of Iterative MegeSort Algorithm Implementations in C, Java and Python.

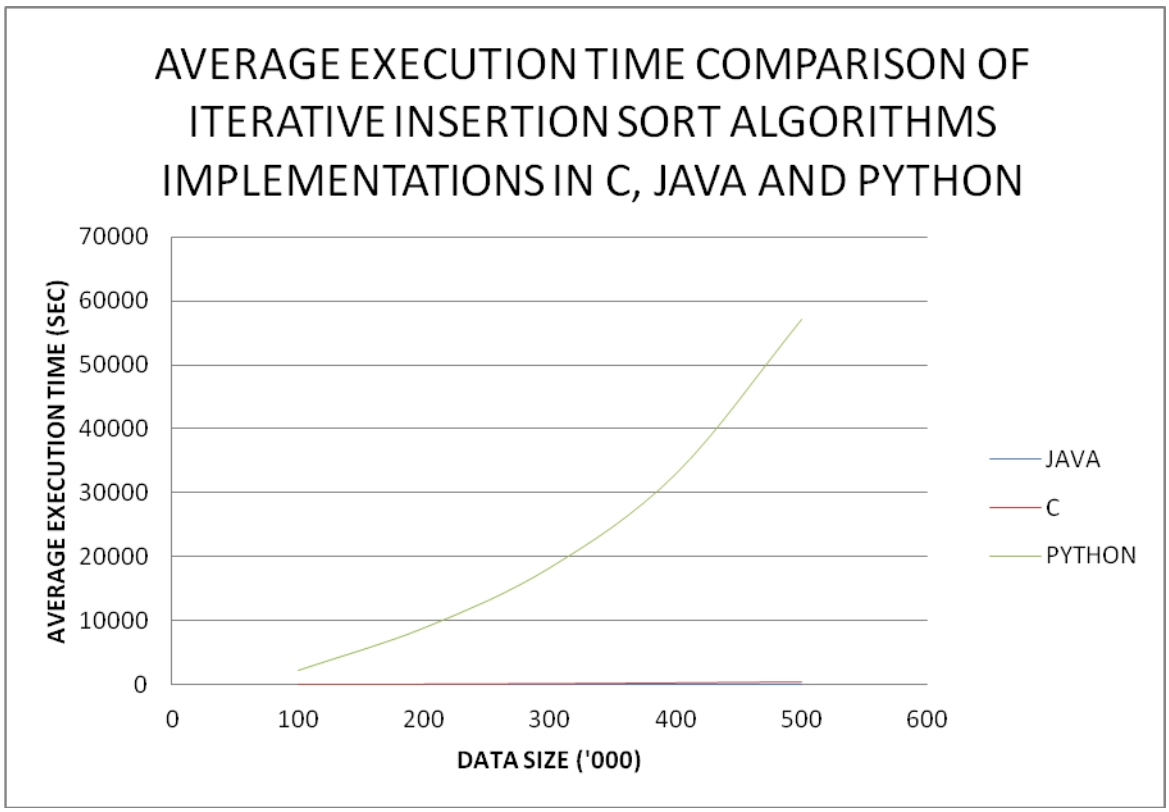


Figure 4.3: Average Execution Time Comparisons of Iterative Insertion Sort Algorithm Implementations in C, Java and Python

Figure 4.4: Average Execution Time Comparison 0f Recursive Quick Sort Algorithms Implementations in C, Java and Python

Figure 4.5: Average Execution Time Comparison 0f Recursive Merge Sort Algorithms Implementations in C, Java and Python

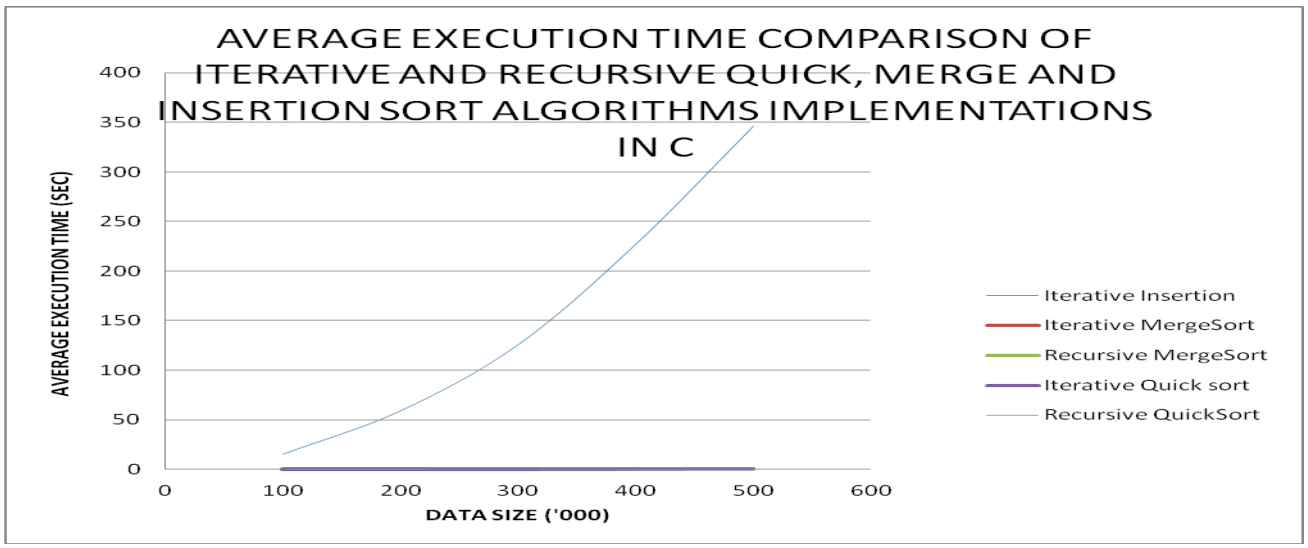


Figure 4.6: Average Execution Time Comparison of Iterative and Recursive Quick, Merge And Insertion Sort Algorithms Implementations in C

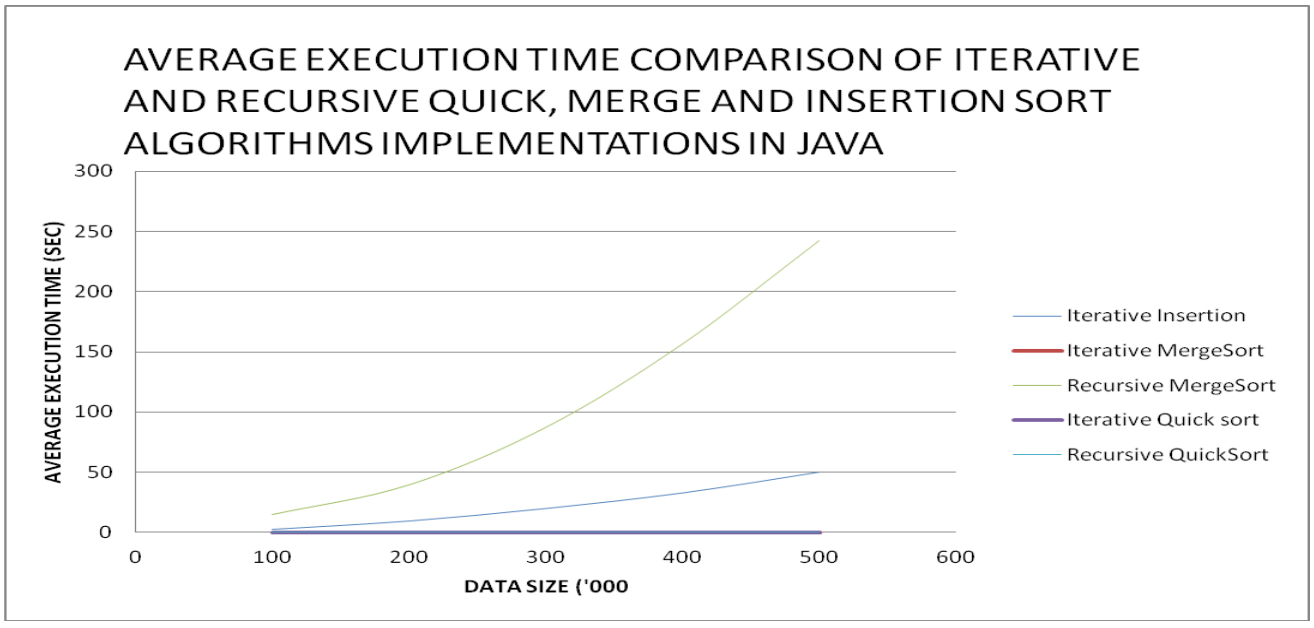


Figure 4.7: Average Execution Time Comparison of Iterative and Recursive Quick, Merge And Insertion Sort Algorithms Implementations in Java

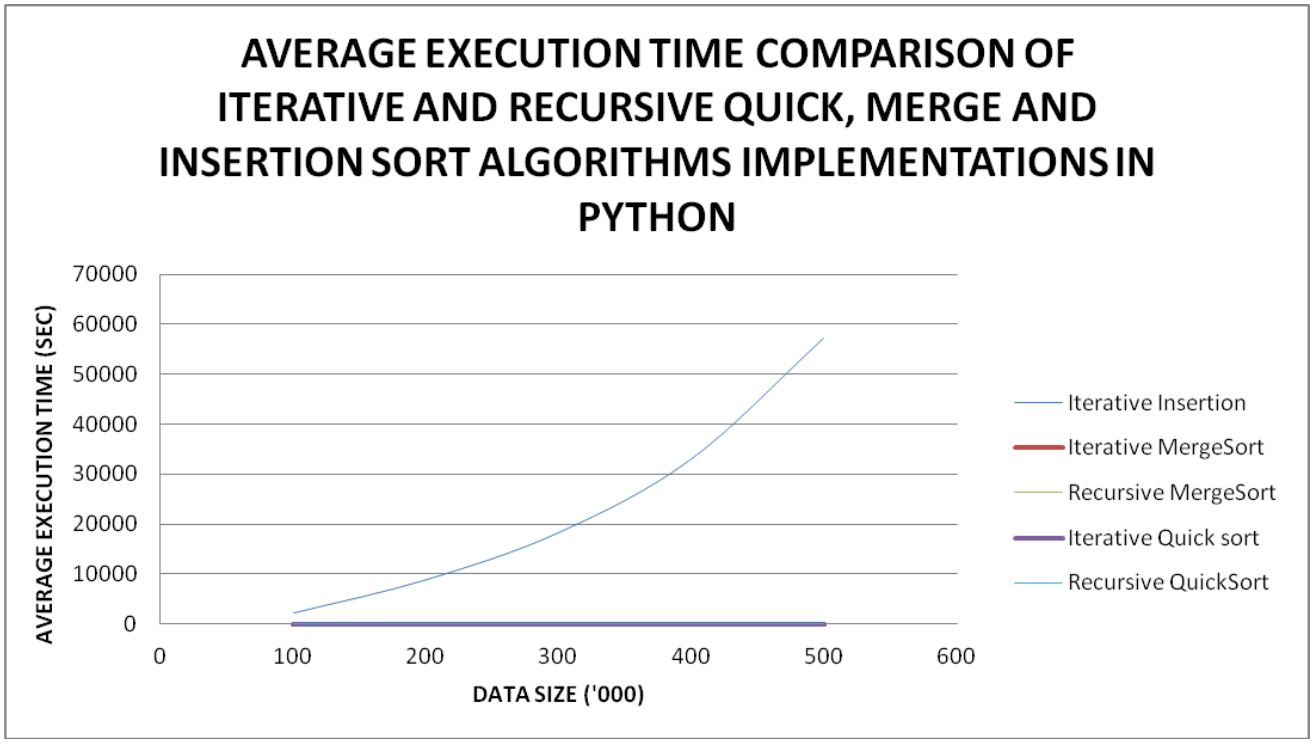


Figure 4.8: Average Execution Time Comparison of Iterative and Recursive Quick, Merge And Insertion Sort Algorithms Implementations in Python