British Journal of Mathematics & Computer Science
20(1): 1-9, 2017; Article no.BJMCS.29238
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Empirical Performance of Internal Sorting Algorithm

Faki Ageebee Silas[1*], Yusuf Musa[1] and S. Akosu Joyce[1]

[1]Department of Computer Science, Faculty of Science and Technology, Bingham University, Karu-Nasarawa State, Nigeria.

Authors' contributions

This work was carried out in collaboration between all authors. Author FAS designed the study, wrote the protocol and supervised the work. Authors YM and SAJ carried out all laboratories work and performed the statistical analysis. Author YM managed the analyses of the study. Author FAS wrote the first draft of the manuscript. Authors YM and FAS managed the literature searches and edited the manuscript. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/BJMCS/2017/29238
Editor(s):
(1) Dariusz Jacek Jakóbczak, Chair of Computer Science and Management in this Department, Technical University of Koszalin, Poland.
Reviewers:
(1) Stephen Akandwanaho, University of KwaZulu-Natal, South Africa.
(2) Chengliang Wang, Chongqing University, China.
(3) Abdul-Salam Sibidoo Mubashiru, Kwame Nkrumah University of Science and Technology, Ghana.
Complete Peer review History: http://www.sciencedomain.org/review-history/17182

Received: 30th August 2016
Accepted: 26th October 2016
Published: 9th December 2016

Original Research Article

Abstract

Internal Sorting Algorithms are used when the list of records is small enough to be maintained entirely in primary memory for the duration of the sort, while External Sorting Algorithms are used when the list of records is large enough to be maintained in physical memory hence a need for external/secondary storage for the duration of the sort. Almost all operations carried out by computing devices involve sorting and searching which employs Internal Sorting Algorithms. In this paper, we present an empirical analysis of Internal Sorting Algorithms (bubble, insertion, quick shaker, shell and selection) using sample comprising of list of randomly generated integer values between 100 to 50,000 samples. Using C++ time function, it was observed that insertion sort has the best performance on small sample say between 100 to 400. But when the sample size increases to 500, Shaker sort has better performance. Furthermore, when the sample grows above 500 samples, shell sort outperforms all the internal sorting algorithms considered in the study. Meanwhile, selection sort has displayed the worst performance on data samples of size 100 to 30,000. As the samples size grows to further to 50,000 and above, the performance of shaker sort and bubble sort depreciates even below that of selection sort. And when the sample size increases further from 1000 and above then shell sort should be considered first for sorting.

**Corresponding author: E-mail: faki.silas@binghamuni.edu.ng;*

Keywords: Internal sorting; empirical analysis; space complexity; time complexity.

1 Introduction

Sorting algorithm is an algorithm that rearranges data elements of a list in certain user defined order. Sorting can be classified in terms of the data-size and the type of memory required for a sorting computation. When the list of data-size is small enough to be accommodated at a time in the primary memory we refer to such sorting algorithm as Internal Sorting Algorithm (ISA). On the order hand, when the list of data-size is large to an extent that will require secondary memory other than primary memory are refer as external sorting [1]. ISA are faced with the limitation of memory constraint therefore, since they can only process small list capable of running at once in the primary memory. There are three categories of ISA;

(i) Selection Sort (SS) which comprises of selection and heap sort algorithms.
(ii) Insertion Sort (IS) which comprises of insertion sort algorithm and shell sort algorithm.
(iii) Exchange Sort (ES) which comprises of bubble sort algorithm and quick sort algorithms

Internal sorting algorithms (ISAs) are very prevalent in practical and real world situations. This is because most practical problems in computing require a specific arrangement of outputs. Most often you need to sort data in order to normalize it, and make querying efficient. There are many sorting algorithms and some programmers tend to develop some with the intent of getting faster algorithms [2]. The efficiency of an algorithm is estimated in terms of execution time and the amount of memory it requires. This execution time of an algorithm which is also known as time complexity is the amount of time required for the algorithm to be executed. It is worthy to note that programming language used in the implementation of an algorithm and quality of compiler does not affect time complexity of algorithms [2]. This does not rule out the challenge faced by programmers in which ISAs to use in a particular situation because some are faster than others under different situations. It therefore becomes necessary to evaluate ISAs to know their behavior on different data instances. This will provide researchers with an empirical fact and also enable programmers know the precise ISA to use in situation where primary memory is all that is needed to sort the dataset presented for sorting. Several authors have based their study on performance of external sorting algorithm using large data set. It is also important to know which ISA work on small data samples like telephone number, salary of workers, age of students, matriculation number of students etc.

This study therefore, investigates and evaluate the performance of ISAs using empirical analysis not the Asymptotic Complexity Analysis since all the algorithms are internally sorted algorithms [3]. Though, a little discussion was done on asymptotic complexity analysis (Table 1) just to alternate the limitations of empirical analysis which is system dependent, [4] more complexity analysis techniques are available in [5], [6], [7]. The data sets used in the study are randomly generated integers ranging between 100 to 50,000.

2 Review of Related Literature

Over the years, Empirical analysis has become a crucial part of the study of algorithms [5]. This is because sorting data is essential to computer program and different sorting algorithms works in different ways with advantages and disadvantages based on data input size and other architectural parameters. [8] considered features in the process of analyzing an algorithm empirically, these features include; Correctness of the algorithm, work done, and space used, simplicity or clarity and optimality. Correctness of an algorithm proves the relationship between the precondition and post conditions taking on the characteristics of the input as expected, execute it with desired output. Amount of work done measured the efficiency of the algorithm by comparing two or more algorithm on their performance in solving same problem. The amount of space used is a measure of memory space required for execution of an algorithm. Being an internal sorting algorithm where all data reside in the resource memory, this memory is affected as data increase in size. Simplicity or clarity easily makes implementation, debugging and modifying of program while optimality depicts the best possible algorithm that solves a particular problem [1], [9], [10], [11]. Being exposed to different sorting techniques is a common task in programming. This makes it necessary for programmers to know which algorithm best suit a particular situation. Since the study only focus on empirical evidence of ISA, and different computers and operating systems vary widely in how they keep track of CPU time in with ISA [12,13], we consider running the experiment on a system while taking record of CPU execution time for systems.

Space complexity was considered in base on two kind of memory usage patterns; “in-place” sorting algorithms that do not requires extra space to accomplish its task, therefore maintains a memorize O(1). While “out-play” sorting algorithm requires extra space O(n) or even O(log n) [14]. A stable algorithm preserves relative order or position for duplicate array values. E.g. Insertion, bubble and shaker algorithm [14]. Space complexity is a measure of the amount of working storage an algorithm needs. It can also be seen as the essential memory cells that are needed by an algorithm. This affect the performance of any running ISA [14]. Time complexity which determines how long an algorithm is ran must be considered for algorithm performance evaluation. Practically, the better the time complexity of an algorithm is, the faster the algorithm will carry out his work. There is often a time-space-tradeoff involved in most practical situation. One then, has to make a compromise to exchange computing time for memory consumption or vice versa, depending on which algorithm one chooses and how one parameterizes it.

The study focus on empirical evidence of ISA which were tested on windows 8.1, with Intel Pentium (R), CPU speed of 2.80GHz, 4GB RAM and 64bits OS system type.

Being exposed to different sorting techniques as a common task in programming, this makes it necessary for programmers to know which algorithm best suit a particular situation [15]. This study only focus on empirical evidence of empirical analysis which entails; (1) Understanding the theoretical analysis. (2) Decide on what features are to be measured: in this study we measured Time and space complexity, stability and adaptability. (3) What is the most appropriate hardware to run the measurement on: we used the processor speed, internal memory size and operating system type. (4) Which is the most appropriate implementation language: we chose C++ to implement ISAs. (5) Which is the most appropriate data structure to use, (6) implement the algorithms for comparison, (7) implement some form of timing device. (8) Create the input dataset necessary to produce the measurement we need. (9) Measure the performance of the algorithm on the different input dataset created to meet the aim of the analysis. (10) interpret the result and relate to the theoretical analysis[16,17].

It is worth knowing that, at the course of performing empirical analysis of ISA, the execution time of implementing a given algorithm depends on the CPU processing capacity, compilation rate of the Compiler, programming language used in the implementation, algorithm construction and implementation, memory access time of input/output and whether the operating system is multitasking of single tasking [17]. Due to the nature of computing device in use, the CPU processing capacity and Memory determine how fast algorithm can be executed [18]. CPU depends heavily on the computing resources ability to process floating point (if available) and integers. It is worth of note that the general performance of CPU depends on the platform of the computing resources [19]. The rate of compilation of a compiler has a tremendous effect on the performance of algorithm [20]. Different compilers and versions compile codes at different rate. Programming language affect algorithm based on specific criteria. Example, C programming may be better in terms of execution based on its closeness to machine language while Java is better for web application and C# for GUI design [15]. Input/output and operating system also determine how fast an algorithm can be executed.

3 Overall Analysis of Time and Space Complexity with Stability and Adaptability of ISA

When comparing ISAs, there are four factors that needs to be considered, these factors include; Time complexity analysis is an effective way of comparing algorithms through their time complexity, which depends upon the number of comparisons made when an algorithm runs at magnitude of O(f(n)). Where n is the number of items in the list to be sorted [4,21].

Table 1 shows that Quick sort has better performance in its best case (Ω(nlogn)) and average case (Θ(nlogn)) runtime compared to the rest of the ISAs. Shell sort is next with best case of Ω(nlogn). It is therefore obvious that selection sort will not be a preferred choice over other sorting algorithms with its best case ($\Omega(n^2)$), average case ($\Theta(n^2)$) and worst case ($O(n^2)$) respectively. Therefore, empirical analysis would be handy to provide a better comparability outcome.

Table 1. Time/space complexity and stability/adaptability of ISAs [4,22,23]

		Time complexity					
		Best Case	**Average case**	**Worst case**	**Auxiliary Space complexity**	**Stability**	**Adaptability**
ISA	Insertion Sort	Ω(n)	$\Theta(n^2)$	$O(n^2)$	O(1)	Yes	Yes
	Quick Sort	Ω(n log n)	Θ(n log n)	$O(n^2)$	O(n)	No	Yes
	Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	O(1)	No	No
	Bubble Sort	Ω(n)	$\Theta(n^2)$	$O(n^2)$	O(1)	Yes	Yes
	Shaker Sort	Ω(n)	$\Theta(n^2)$	$O(n^2)$	O(1)	Yes	
	Shell Sort	Ω(n log n)	$\Theta(n^2)$	$O(n^2)$	O(1)	No	Yes

4 Empirical Analysis

Empirical analyses are based on executing the algorithms on a computer system rather than using asymptotic notations. To achieve that, we run the experiment on windows 8.1, with Intel Pentium (R), CPU speed of 2.80GHz, 4GB RAM and 64 bits OS system type.

The data samples selected were inserted into an array of size from 100 to 50,000 elements. In each implementation, "CLOCKS_PER_SEC" the function in C++ library (in the time.h) was used to estimate the execution time of ISA as shown in Fig. 1. The C++ clock function returns a value expressed in clock ticks, which are unit of time of a constant [13].

```
#include <time.h>
for(i =1; i<n; i++){
    arr[i]= rand()%n+1; //data generator
}clock_t begin =clock(); //counting sorting time begins
    //sorting code here
clock_t end = clock();//counting sorting time ends
double cpu_time_to_sort = ((double) (end – begin))/ CLOCKS_PER_SEC;
```

Fig. 1. The code snippet for C++ clock function

The clock_t is the Data type of the value of the numbers of clock ticks that is returned by the clock function. The CLOCKS_PER_SEC is a macro type that holds the number of clock ticks per second measured by clock function

Each of the algorithms was run for 100 trials and an average was calculated by dividing the sum of execution time of trials by the number of trials. The time it takes to sort each sorting algorithm in respect to the data was recorded and represented as shown in Table 2.

5 Experimental Results and Interpretation

Table 2 presents the outcome of the test conducted on each of the ISA with various data categories.

Table 2. Time comparison (sec) for ISA on sizes (100 to 50,000) of data samples

		Data sample size							
		100	**500**	**1,000**	**1,500**	**5,000**	**10,000**	**30,000**	**50,000**
ISAs	Insertion	**0.015***	0.046	0.094	0.140	0.451	0.938	3.317	6.25
	Quick	0.016	0.047	0.141	0.156	0.516	0.937	2.796	5.35
	Selection	**0.047****	**0.187****	**0.312****	**0.468****	**1.453****	**2.829****	**8.484****	14.172
	Bubble	0.016	0.047	0.094	0.140	0.547	1.328	6.734	**15.985****
	Shaker	0.016	**0.048***	0.097	0.156	0.546	1.252	5.94	13.532
	Shell	0.016	0.047	**0.093***	**0.141***	**0.453***	**0.906***	**2.743***	**4.552***

Taking a critical look at Table 2, each data sample column has its lowest (*) and highest (**) execution time value in bold and italics. The lowest values represent best execution time and the fastest algorithm in respect to that particular data sample size, while the highest values represent the worst execution time and the slowest algorithm in respect to that particular data sample size. The result on Table 2 shows that Selection sort is the slowest ISA since it has the highest value all through the data samples. But at 50,000 data sample, bubble sort became slower than selection sort since it has the largest value. This means bubble sort performance depreciates with increase in data sample size. On the other hand, Shell sort has the least execution time all through the data samples making it the fastest ISA both with small and large data samples. The result of behavior of different ISA in respect to input sample size is presented in Fig. 3.

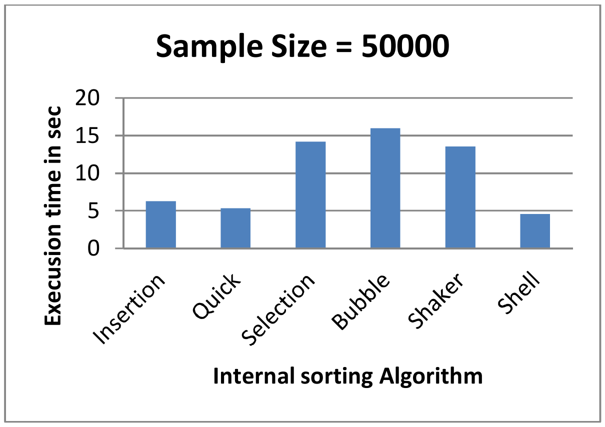

Fig. 2. Execution time performance between ISAs over 50,000 data samples

Fig. 2 validates the fact that the performance of bubble sort, followed by shaker sort compared to selection sort depreciates as data sample size increases from 50,000 and above. However, shell sort remains the best choice of ISA when working with large sample size.

The relationship between the time taken for sorting and the amount of element is nonlinear as seen in Fig. 4. Comparing the ISA, a clear variation of execution time can be seen. The performance of the entire data sample used in the study (100, 500, 1000, 1500, 10,000, 30,000, and 50,000) for each of the ISA is shown in Fig. 3.

The study observes that insertion sort has the best performance for small data. Therefore, Insertion sort is much faster in computing less random numbers of small size than any of the ISA considered in the study. However, shell sort outperforms insertion sort as the data sample increases.

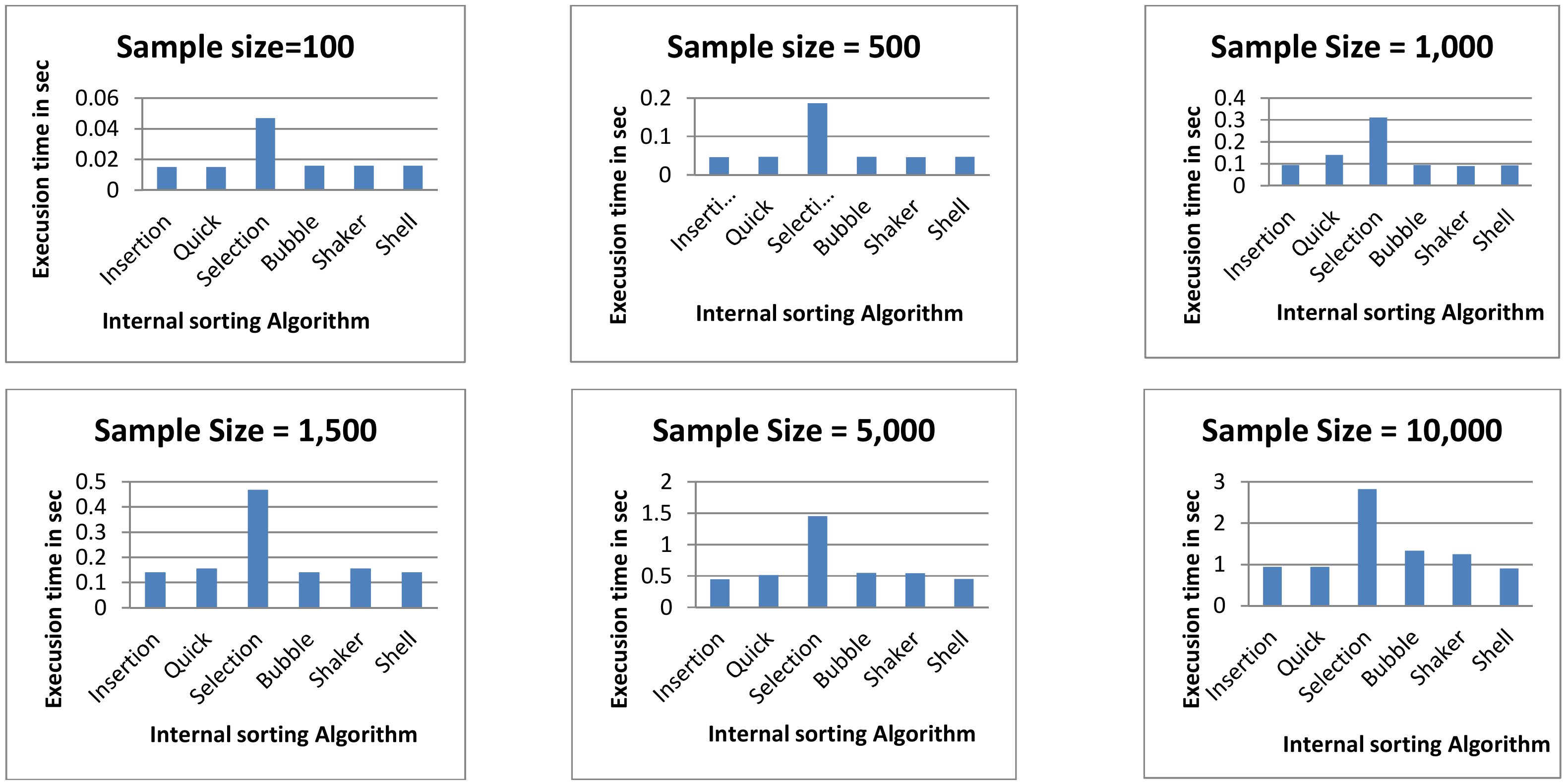

Fig. 3. Comparison of ISA across data sample sizes

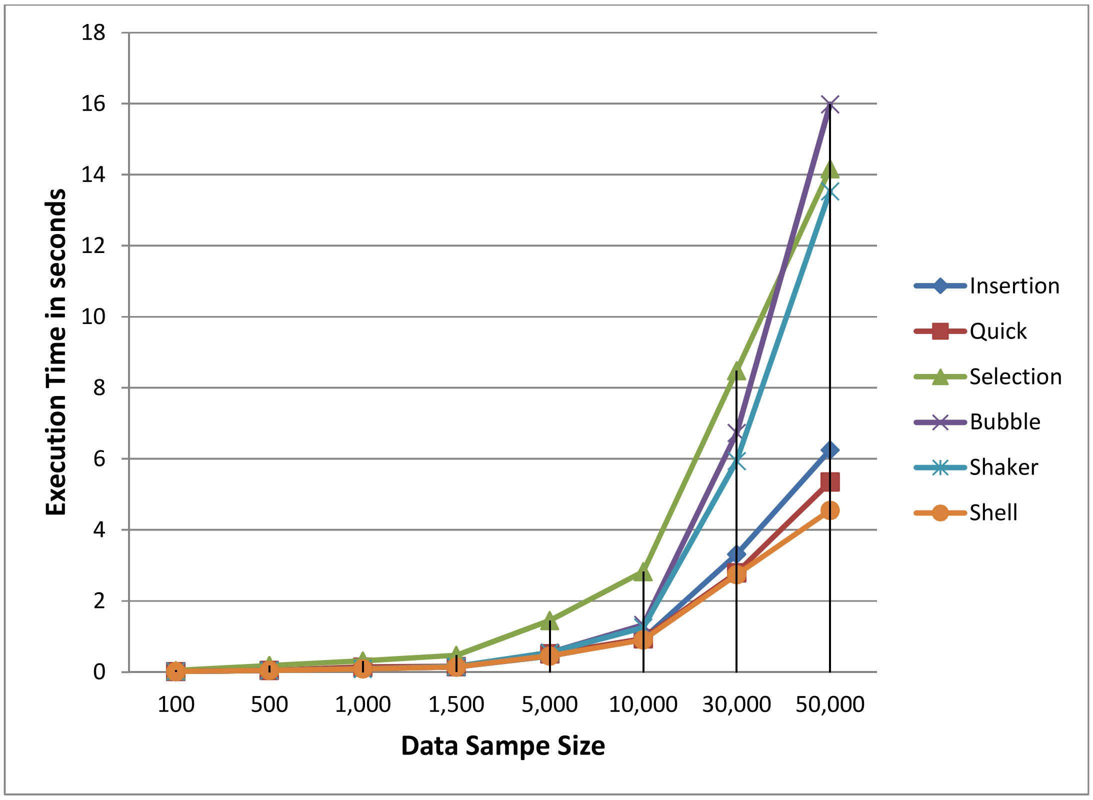

Fig. 4. General overview of growth rate based on data size and execution time in sec of ISAs

6 Further Study

This study was carried out with a single computing device. In the future, researchers could use different computing resources with varying computing speed to compare the effect of processor speed on these data samples. Also, only integers were used as data sample; it is the interest of the researchers to know what will happen to character arrays in respect to internal sorting in the future.

7 Conclusion

In this paper, we have tested six ISAs on random data sample of sizes from 100 to 50,000. We apply the six ISA on each of the data samples sizes and compare their performance in each case. And also we have found out best and worst case with respect to execution time. All the ISA implemented in C++ language. The programs were designed on CodeBlocks 10.5 IDE with C++ 5.02 compiler and executed on Intel Pentium (R) processor, and the programs running at 2.80GHz clock speed.

Insertion sort should be considered ahead of every other ISAs when data sample is small (say less than 100 data sample). But when the data sample increases from 100 to 500, shaker sort should be considered ahead of insertion sort. And when the sample size increases further from 1000 and above then shell sort should be the best choice for sorting.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Brassard G, Bratley P. Fundamentals of algorithmic. ed: Prentice-Hall, Inc. Pearson Education, Upper Saddle River; 1996.

[2] Aliyu AM, Zirra P. A Comparative analysis of sorting algorithms on integer and character arrays. The International Jornal of Engineering and Science. 2013;25-30.

[3] Cormen TH, Leiserson CE, Rivest RL. Clifford Stein. Introduction to Algorithms; 2001.

[4] Ocampo JP. An empirical comparison of the runtime of five sorting algorithms. International Baccalaureate Extended Essay; 2008.

[5] Sedgewick R, Flajolet P. An introduction to the analysis of algorithms. Addison-Wesley; 2013.

[6] Purdom Jr PW, Brown CA. The analysis of algorithms. Holt, Rinehart & Winston; 1985.

[7] Singh NK, Chakraborty S. Smart sort: Design and analysis of a fast, Efficient and robust comparison based internal sort algorithm. arXiv preprint arXiv:1204.5083; 2012.

[8] Baase S. Computer algorithms: Introduction to analysis and design. ed: Addison Wesley, Reading, Massachusetts; 1988.

[9] Corman TH, Leiserson CE, Rivest Ronald L. Introduction to algorithms. The MITelectricalengineering and computerscienceseries. ed: The MIT Pressl 1990.

[10] Neapolitan RE, Naimipour K. Foundations of algorithms. Jones & Bartlett Learning; 2010.

[11] Manber U. Introduction to algorithms: A creative approach. Addison-Wesley Longman Publishing Co., Inc.; 1989.

[12] Loosemore S. Oram A, Drepper U. The GNU C library. System & Network Applications; 1999.

[13] Loosemore S, Stallman RM, McGrath R, Oram A, Drepper U. The GNU C library reference manual. Free Software Foundation; 2001.

[14] Karunanithi AK. A survey, Discussion and comparison of sorting algorithms. Department of Computing Science, Umea University; 2014.

[15] Alomari Z, Halimi OE, Sivaprasad K, Pandit C. Comparative studies of six programming languages. arXiv preprint arXiv:1504.00693; 2015.

[16] Sanders I. Empirical analysis of algorithms is easy (or is it?). Citeseer; 2001.

[17] Brunskill D, Turner J. Understanding algorithms and data structures. McGraw-Hill Companies;1996.

[18] Patterson DA, Hennessy JL, Computer organization and design: The hardware/software interface: Newnes; 2013.

[19] Heineman GT, Pollice G, Selkow S. Algorithms in a nutshell: A practical guide. O'Reilly Media, Inc.; 2016.

[20] Team J. Julia Programming Language. (2013, 13/10/2016).

[21] Lafore R. Data structures and algorithms in Java; 2003.

[22] Faujdar N, Ghrera SP. Analysis and testing of sorting algorithms on a standard dataset. In Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference. 2015;962-967.

[23] Al-Kharabsheh KS, AlTurani IM, AlTurani AMI, Zanoon NI. Review on sorting algorithms a comparative study. International Journal of Computer Science and Security (IJCSS). 2013;7:120-126.