Performance Estimation of Non-comparison Based Sorting Algorithms Under Different Platforms and Environments

Mentor Hamiti and Diellza Nagavci

***Abstract*—**There may be several different platforms for performance estimation of non-comparison based sorting algorithms. Understanding the relative efficiencies of algorithms designed to do the same task is very important in every area of computing. An algorithm can be analyzed in terms of time efficiency or space utilization. The efficiency, with which a sorting will be carried out, often has a big impact on the effectiveness of the program as a whole. Because platforms and surroundings are in progression and permanent change, they always need to follow these parameters. The goal of this paper is to review different non-comparison based sorting algorithms. In this occasion three different environments and computer performances are used and the obtained results are also analyzed in this paper.

***Keywords*—**algorithm, environment, non-comparison, platform, sorting.

I. Introduction

SORTING is maybe the single most important algorithm performed by computers, and certainly one of the most investigated topics in algorithmic design. One of the fundamental problems of computer science is ordering a list of items. There is a plethora of solutions to this problem, known as sorting algorithms. An algorithm can be analyzed in terms of time efficiency or space utilization. The running time of an algorithm is influenced by several factors: speed of the machine [2] running the program and language in which the program was written; Efficiency of the compiler that created the program, the size of the input and the organization of the input. Examples of sorting algorithms that run in linear time are counting sort, radix sort and bucket sort are executed in three platforms as CPU: Intel® Core i5™-M460 2.53GHz (2 Cores), RAM: 6GB, CPU: Intel® Core i3™-2100 3.10GHz (2 Cores), RAM: 4GB and also in Pentium® Dual Core – T4200 2.0GHz (2 Cores), RAM: 4GB and three environs as C++, Python and Java.

II. NON-COMPARISON SORT

A. Bucket Sort

Bucket Sort is a sorting method that subdivides the given data into various buckets depending on certain characteristic order, thus partially sorting them in the first go. Then depending on the number of entities in each bucket, it employs either bucket sort again or some other ad hoc sort. Bucket sort runs in linear time on an average. Bucket sort is stable. It assumes that the input is generated by a random process that distributes elements uniformly over the interval 1 to m. Bucket sorting algorithm is a kind of sustainable, [1] it takes data generated by a random process that distributes the same elements in the interval O (n). Bucket sort divides the intervals [0,1) in the same size intervals or bucket and then distributes them in the data bucket. Once the data are distributed uniformly and in the interval [0,1) we do not expect that each number will enter the empty bucket-mails. To gain done sorting scoring numbers in each bucket and then go to the order of bucket's listed the elements in the list.

B. Counting Sort

Counting sort is an algorithm used to sort data whose range is pre-specified and multiple occurrences of the data are encountered. It is possibly the simplest sorting algorithm. The essential requirement is that the range of the data set from which the elements to be sorted are drawn is small, compared to the size of the data set [3]. Counting sort works by determining how many integers are behind each integer in the input array A. Using this information, the input integer can be directly placed in the output array B. This type of sorting works best when data distribution is uniform. An example of efficient use of Counting Sort order can be 200 students on the basis of their results by sorting 100 or 1500 employees in connection with the filing of their birthday in a year. The drawback may occur if range m >> n (where n is the number of data while m is the range of data), the complexity will not be linear in n and thus this sort will not remain useful longer. This is because the chances of the appearance of gaps, during the sorting for those elements which do not exist in the list will cause a higher complexity of space. Because counting sort algorithm is a straightforward algorithm is quite simple and easy to be analyzed in the context of software complexity. The worst case and the average performance of counting sort algorithm is O (n + k). In order to ensure maximum efficiency, "k" should not be higher than "n". Counting Sort When compared with other sorting algorithms, appears to be easier to

implement and does not require any special structure of data to store its elements.

C. Radix Sort

A radix sort is an algorithm that can rearrange integer representations based on the processing of individual digits in such a way that the integer representations are eventually in either ascending or descending order. Integer representations can be used to represent things such as strings of characters (names of people, places, things, the words and characters, dates, etc.) and floating point numbers as well as integers. So, anything which can be represented as an ordered sequence of integer representations can be rearranged to be in order by a radix sort [4]. Most digital computers internally represent all of their data as electronic representations of binary numbers, so processing the digits of integer representations by groups of binary digit representations is most convenient. Two classifications of radix sorts are:

- Least significant digit (LSD) radix sort.
- Most significant digit (MSD) radix sort.

LSD radix sorts process the integer representations starting from the least significant digit and move the processing towards the most significant digit. MSD radix sorts process the integer representations starting from the most significant digit and move the processing towards the least significant digit. The integer representations that are processed by sorting algorithms are often called "keys," which can exist all by themselves or be associated with other data. LSD radix sorts typically use the following sorting order: short keys come before longer keys, and keys of the same length are sorted lexicographically. This coincides with the normal order of integer representations, such as the sequence 1, 2, 4, 5, 6, 7, 8, 9. MSD radix sorts use lexicographic order, which is suitable for sorting strings, such as words, or fixed-length integer representations. A sequence such as b, c, d, e, g, h, i, j, ba would be lexicographically sorted as b, ba, c, d, e, f, g, h, i, j.

If lexicographic ordering is used to sort variable-length integer representations, then the representations of the numbers from 1 to 10 would be output as 1, 10, 2, 3, 4, 5, 6, 7, 8, 9, as if the shorter keys were left-justified and padded on the right with blank characters to make the shorter keys as long as the longest key for the purpose of determining sorted order.

III. TESTING IN DIFFERENT PLATFORMS AND ENVIRONMENTS

The three non-comparison algorithms that are tested for different number of CPUs will enable finding the best ratio of the volume of data to the number of cores. For example Bucket sort is implemented in three platforms, Radix Sort and Counting Sort in three platforms and in C++, Python and Java environments.

A. Bucket sort implementation CPU platform: Intel® Core i5 (TM) 2.53GHz, 6GB RAM in C++ environment

To analyze an algorithm we should provide tools which are used by an algorithm for functioning. In the general case these tools are: space memory devices, generation communications or computer hardware and execution time. Bucket sort algorithm is implemented in C++ environment executed in Visual Studio 2013.

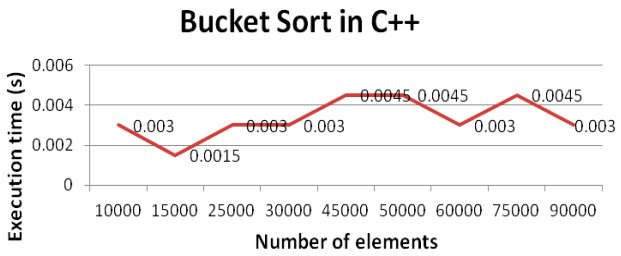

Fig. 1. Results of Bucket sort execution time

The diagram shows bucket sort execution time by the number of elements. With increasing the size and the number also increases the execution time on this platform. The best time exestuation is 0.003 seconds.

B. Bucket sort implementation CPU platform: Intel® Core i5 (TM) 2.53GHz, 6GB RAM in Python environment

Diagram for bucket sort in python environment presents results that show the curve through the highest point of the execution time in this case is thus 0.065 seconds in the range of 10000 to 100000 numbers.

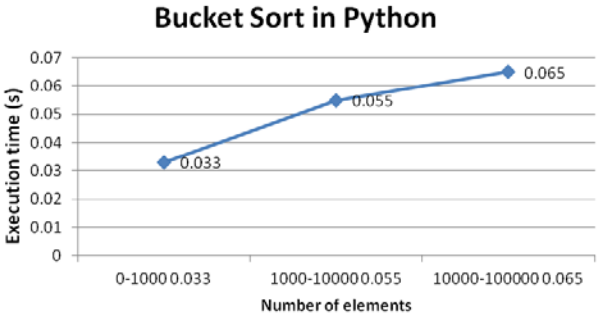

Fig. 2. Results of Bucket sort execution time in Python

C. Radix sort implementation CPU platform: Intel® Core i5 (TM) 2.53GHz, 6GB RAM in Java environment

Speed of radix sort largely depends on the inner basic operations and if operations are not efficient enough radix sort can be slower than some other algorithms such as quick sort or merge sort.

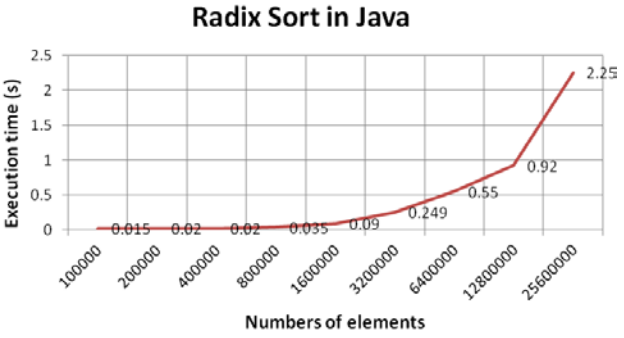

Fig. 3. Radix Sort in different platforms in Java environment

These operations include the insert delete function of the sub lists and the process of isolating the digit we want.

Based on this graphic we can conclude that radix sort in 25600000 elements had a worst case of exestuation, otherwise the best case is 0.015 seconds in 100000 elements.

D. Counting sort implementation CPU platform: Intel® Core i3 (TM)2100 3.10 GHz, 4GB RAM in Python environment.

Counting sort is implemented in Python environment, this non-comparison algorithm is stable. The best case of exestuation time is 0.009 seconds.

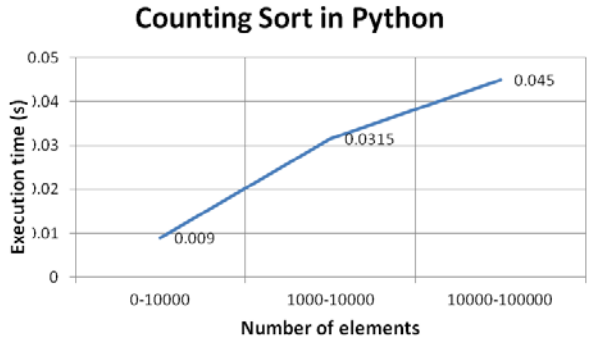

Fig.4. Counting Sort in different platforms in Python environment

E. Radix sort implementation CPU platform: Intel® Core i3 (TM)2100 3.10 GHz, 4GB RAM in Java environment.

Implementation of Radix Sort in Java environment with CPU platform: Intel® Core i3, has different results, the best case is 0.022 seconds. The Java was complied in Eclipse.

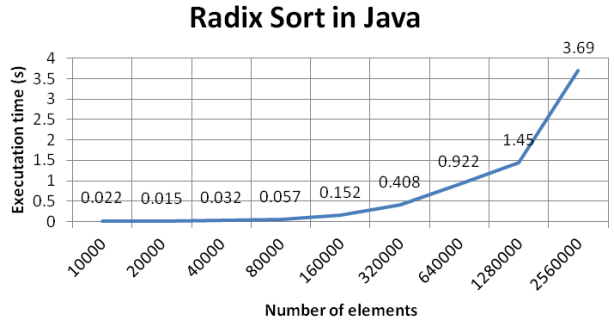

Fig. 5. Radix Sort in different platforms in Java environment

F. Bucket sort implementation CPU platform: Pentium® Dual Core 2GHz, 4GB RAM in C++ environment.

Bucket sort algorithm is implemented in platform CPU: Pentium® Dual Core 2GHz, in the C++ environment complied in Visual Studio 2013. As we can see in the figure the worst exestuation time is 27.852 seconds, but the best case is 0.434 seconds. We can conclude that the execution time depends in the number of elements.

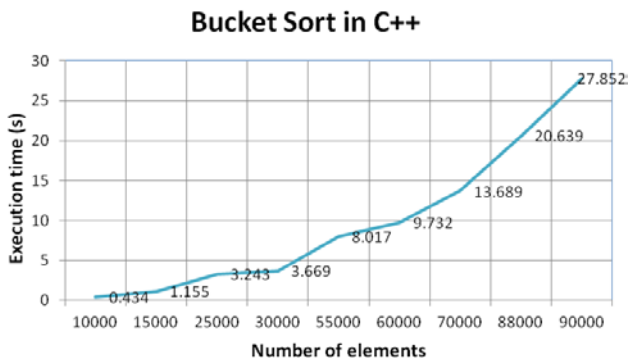

Fig. 6. Bucket Sort in different platforms in C++ environment

G. Radix sort implementation CPU platform: Pentium® Dual Core 2GHz, 4GB RAM in Java environment.

Radix Sort is implemented in java environment and in platform with Pentium® Dual Core 2GHz. The code is compiled in Eclipse. Based in the figure we can conclude that the best case of execution is 0.02, increasing the number of elements the execution time will score 12.25 seconds.

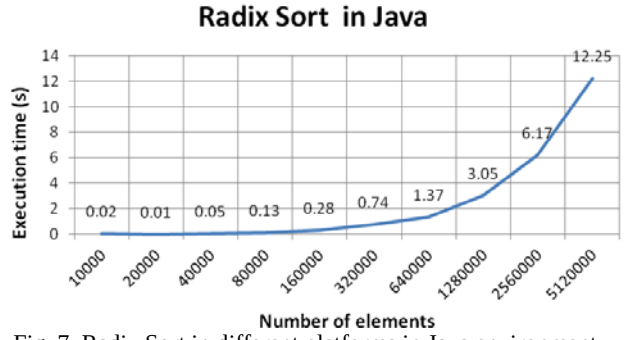

Fig. 7. Radix Sort in different platforms in Java environment

IV. Analysis of the Results Obtained Different Platforms and Environments

Non-comparison algorithm Bucket, Radix and Counting sort are tested in configurations with various performance and by that we conclude which algorithm is executed most quickly. Also we can reach a point where we see the results obtained by each algorithm without comparisons. These three algorithms implemented in three programming languages selected for this study will serve as a benchmark of the results obtained by different configurations, including computers with processors i5, i3 and Pentium dual-core.

A. Bucket Sort in different platforms in C++ environment

As we can in the Figure Bucket sort algorithm is implemented in three different platforms featuring distinction at the time of execution, depending on the characteristics of the computer.

ISBN: 978-1-61804-320-7

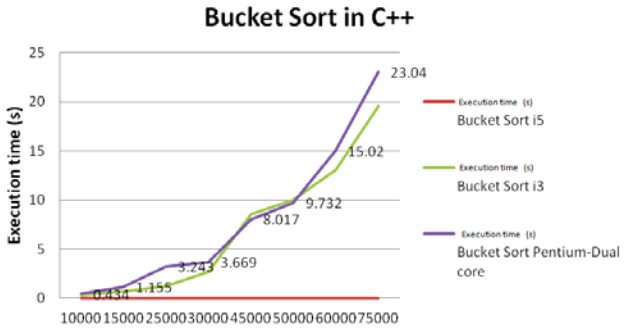

Fig. 8. Bucket Sort in different platforms in C++ environment

B. *Bucket Sort in different platforms in Python environment*

As in the previous case, we conclude that the worst implemented algorithm is in the Python programming language in surroundings Pentium Dual Core where the duration of the performance is 68.78 seconds against times significantly faster computers with processors i3 and i5. Below is the report in tabular and graphical form.

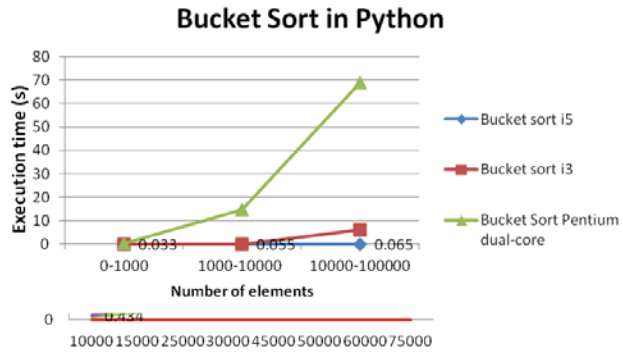

Fig. 9. Bucket Sort in different platforms in Python environment

C. *Bucket Sort in different platforms in Java environment*

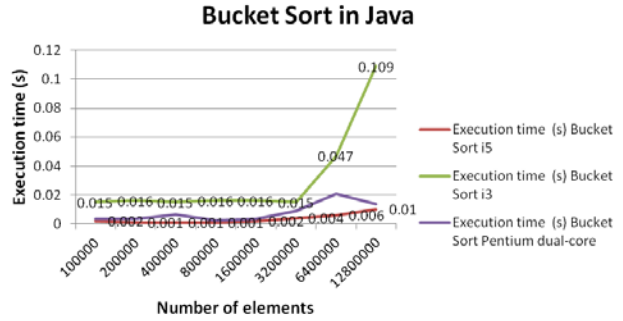

Fig. 10. Bucket Sort in different platforms in Java environment

In the diagram above we can see the results showing the best case and worst case. As best case is the platform with i5 processor which for a short time e executes all data elements in the Java programming language and as a worst case according to the results is the performance of the computer with Pentium Dual Core processor. The best case then algorithm execution is i5 processor.

D. *Radix Sort in different platforms in C++ environment*

Radix sort algorithm is implemented in the vicinity of compiled C ++ in Visual Studio 2013 in various performance processors i5, i3, and Pentium dual-core. Based on the results that are obtained we can conclude which Radix sort algorithm performance is better and which worse. Bellow is the results of comparisons between different performances of Radix sort algorithm.

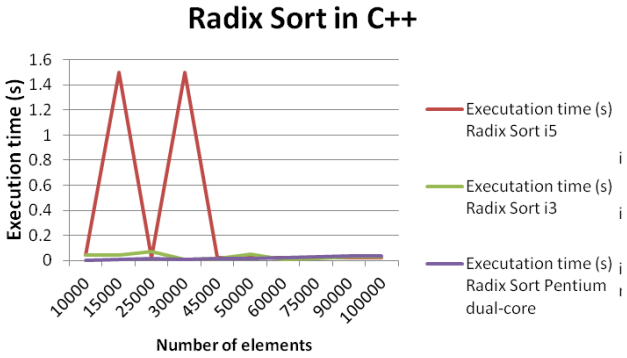

Fig. 11. Radix Sort in different platforms in C++ environment

E. *Radix Sort in different platforms in Python environment*

Radix sort algorithm is implemented in the environment of the compiled Python “Python GUI” in various performance i5, i3, and Pentium dual-core. Based on the obtained results we can conclude for the best and worst performance of Radix sort algorithm. Below are the results of comparisons between different performances of Radix sort algorithm.

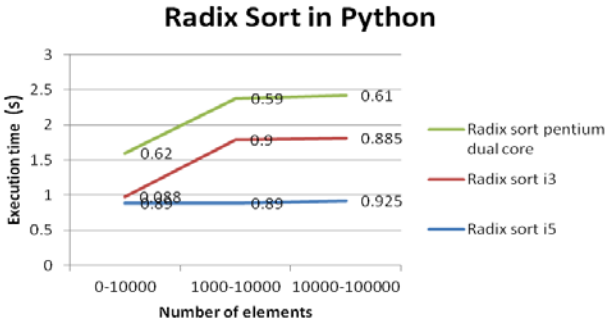

Fig. 12. Radix Sort in different platforms in Python environment

F. *Bucket Sort in different platforms in Java environment*

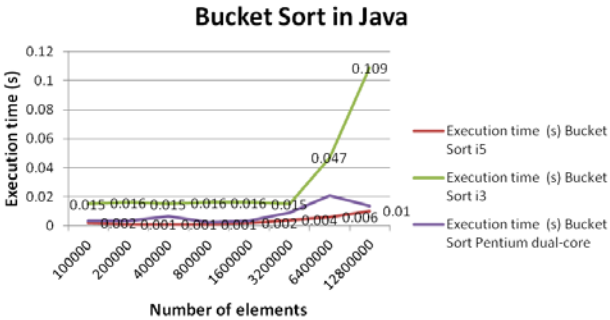

Fig. 13. Bucket Sort in different platforms in Java environment

ISBN: 978-1-61804-320-7

Bucket sort algorithm is implemented in three different platforms i5 processor performance, and Dual-Core i3. Based on these results it was concluded which of these algorithms is most appropriate for the respective performance.

G. Counting Sort in different platforms in C++ environment

Counting sort algorithm is implemented in C ++ environment, which is compiled in Visual Studio 2013 in three different platforms i5 processor performance, and Dual-Core i3. Based on these results it was concluded which of these algorithms is most appropriate for the respective performance.

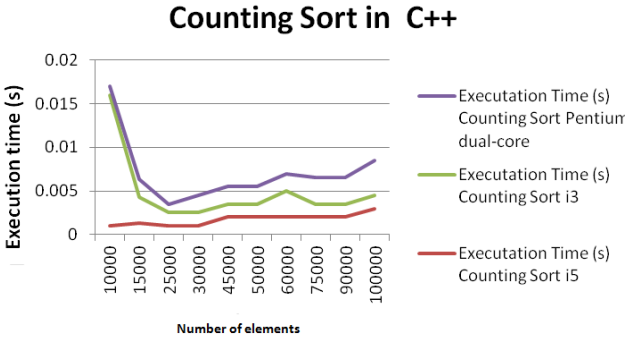

Fig. 14. Counting Sort in different platforms in C++ environment

H. Counting Sort in different platforms in Python environment

Counting sort algorithm is implemented in Python environment which is compiled on the Python GUI and on three different platforms i5 processor performance, and Dual-Core i3. Based on these results it was concluded which of these algorithms is most appropriate for the respective performance.

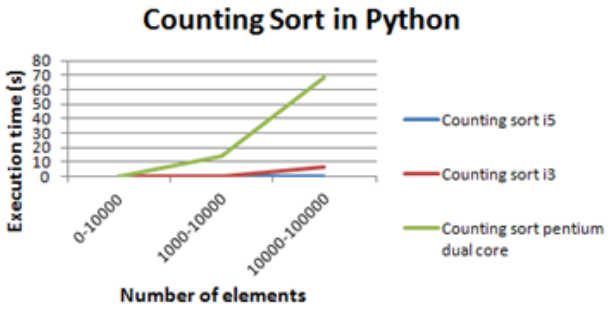

Fig. 15. Counting Sort in different platforms in Python environment

I. Counting Sort in different platforms in Java environment

Counting sort algorithm is implemented in Java environment which is compiled in Eclipse in three different platforms i5 processor performance, and Dual-Core i3. Based on these results it was concluded which of these algorithms is most appropriate for the respective performance.

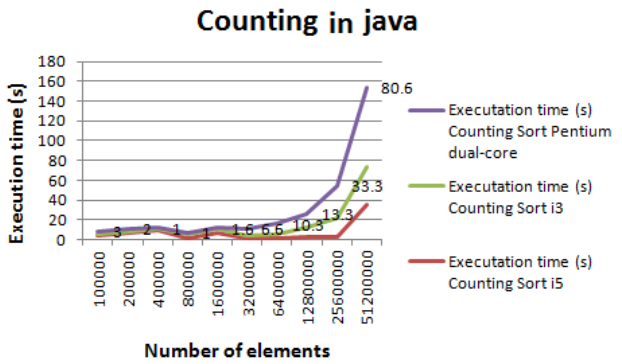

Fig. 16. Counting Sort in different platforms in Java environment

V. Conclusion

In this paper, we have studied and analyzed about non-comparison based sorting algorithms. We analyzed the time complexity of each algorithm with time taken by each step of algorithm in different platforms and environments.

The main objective to analyze the performance of sorting algorithms without comparisons focused on various localities, including programming languages, such as C ++, Python, Java, by analyzing the programs for taking their time for execution. Initially, the description of the algorithm for ranking, then the assessment of sorting algorithms, sorting algorithms classification without comparisons, which represents the complexity of algorithms without comparison, as variable memory devices it increases the number of data. However the essence of this work has been the performance estimation of non-comparison based sorting algorithms under different platforms and environments specifically the implementation of algorithms Bucket, Counting and Radix Sort platforms with processors i5, i3 and Pentium Dual-Core in various localities such as C ++, Python and Java. From the results obtained we can conclude that the algorithm ran in Python environment Counting sort is the best in i5 platform, followed by sort and bucket Radix sort algorithm as last. The obtained results in i3 platform, shows that the first algorithm for this platform is Counting sort, followed by Radix sort and Bucket sort as last. In Pentium Dual-Core platform best algorithm is Counting sort, second is Radix sort and third is Bucket sort. For Java environment in i5 platform the most appropriate algorithm is Bucket sort, the second is the Counting sort and final is Radix sort. For i3 performance in Java environment as first algorithm is Bucket sort, the second is Counting sort and final remains Radix sort. In Pentium Dual-Core performance in Java environment, the best algorithm for this case is Counting sort, second is Bucket sort and last is Radix sort. The results obtained from the implementation of non-comparison algorithms in different platforms have different results. But what is most important is that non-comparison sorting algorithm has the best rating in C++ environment compared with sequential equivalent solutions on platforms Java and Python. After platforms were tested implementations are achieved where the execution time is faster in C++ environment than in Python and Java environment.

ISBN: 978-1-61804-320-7

REFERENCES

[1] Spirakis. (2013). Algorithms and Complexity.

[2] Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Second Edition 2012 (Prentice Hall of India private limited), New Delhi-110001.

[3] Parag Bhalchandra, Proliferation of Analysis of Algorithms with application to Sorting Algorithms, M.Phil

[4] Dissertation, VMRF, India, July 2011.

[5] Mishra, A. B., 2013. Comparison of Sorting Algorithms based on Input Sequences International. Journal of Computer Applications.

[6] Soltys, M., 2012. An Introduction to the Analysis of Algorithms. s.l.:s.n.

[7] Mahfooz Alam, A. C., 2014. Sorting Algorithm: An Empirical Analysis. International

[8] Journal of Engineering Science and Innovative Technology (IJESIT).

[9] Comparison of Sorting Algorithms based on Input Sequences International Journal of Computer Applications (0975– 8887)Volume 78 – No.14, September 2013.