JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

A COMPARATIVE STUDY ON ANALYSIS OF SORTING ALGORITHMS COMPLEXITIES ON DIFFERENT LOADS OF NUMERIC AND STRING DATA

Lavisha Gupta[1], Sarvagya Soni[2], Poshit Nagda[3], Anant Agrawal[4], Mayank Patel[5]
U.G. Scholar [1,2,3,4,] Professor[5]
Department of Computer Science and Engineering[1,3,5], Department of Computer Science and Engineering (Artificial Intelligence)[2,4]
Geetanjali Institute of Technical Studies, Dabok, Udaipur, Rajasthan, India[1,2,3,4,5]

Abstract: Sorting algorithms are the very important data structure operation. As we have millions or trillions of data stored in our memories, it is very difficult for us to find specific required data. To sort data is to arrange them in ascending or descending order so that the searching, locating or arranging of data becomes easy. Every sorting has some advantages and some disadvantages, likewise each sorting algorithm takes different time to sort the data. In this research paper we compared various sorting algorithms in respect to their execution time. The efficiency of every algorithm varies with the number of input and we have compared the efficiency of the algorithm so that we can see which algorithm is best to use based on the load. The sorting algorithms are evaluated in JAVA.

Keywords **– Bubble Sort, Heap Sort, Insertion Sort, Merge Sort, Selection Sort, Sorting Algorithm Evaluation**

I. INTRODUCTION

Sorting algorithm in computer science, is an algorithm that puts elements of a list in a certain order. Efficiently sorting a list is important for optimizing the use of other algorithms (such as search and merge algorithms) which require input data to be stored in lists. There are a lot of sorting algorithms available for sorting the given data or file. Some algorithms are efficient for some inputs and some are not. The efficiency or performance of an algorithm depends on the time and space complexity of the algorithm. The space complexity of an algorithm is the amount of memory it needs to run to completion. The time complexity of an algorithm is the amount of computer time it needs to run to completion.

Any sorting algorithm should satisfy the following properties:

(i) The output must be sorted.

(ii) It must still contain the same elements.

In the various sorting techniques using the stability and the time efficiency. In this paper we discuss the efficiency of the algorithms using the number of inputs and the time taken to sort the elements.

This paper deals with the following sorting techniques:

1. Bubble Sort
2. Selection Sort
3. Insertion sort
4. Quick Sort
5. Merge Sort
6. Heap Sort
7. Radix Sort
8. Counting Sort

II Theoretical framework

Bubble Sort: Bubble sort uses brute force method to sort the elements. Bubble sort is also called as sinking sort, is a simple sorting algorithm that repeatedly steps through the list to be sorted, it compares each pair of adjacent items and swaps them if they are in the wrong order. This algorithm is stable; since it only swaps two items if the latter one is strictly greater, so equal-valued items will stay in their original order. When the elements are already in sorted order, bubble sort takes minimum time. It can be applied to a smaller set of data Variables of the study contains dependent and independent variable. The study used pre-specified method for the selection of variables. The study used the Stock returns are as dependent variable. From the share price of the firm the Stock returns are calculated. Rate of a stock salable at stock market is known as stock price.

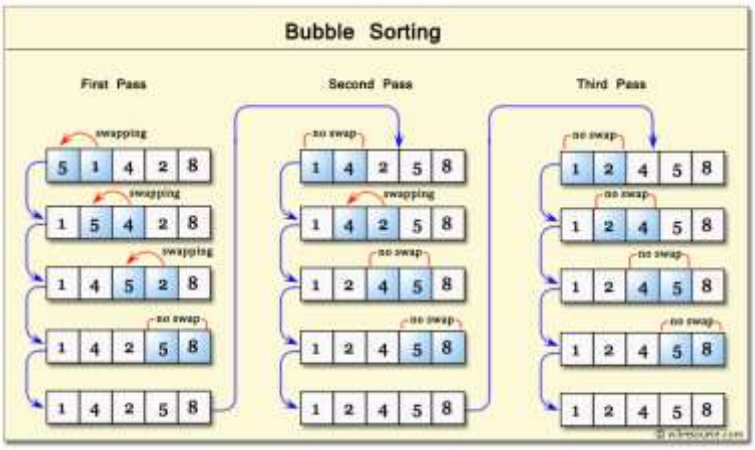

Figure 1. Working of bubble sorting

Selection sort: It is implemented using Brute force technique. The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning of the array. The algorithm maintains two sub arrays in a given array. The first subarray is the array which is already sorted. And the second is the remaining sub array which is unsorted. One of the applications of selection sort is to find the unique element in a data set.

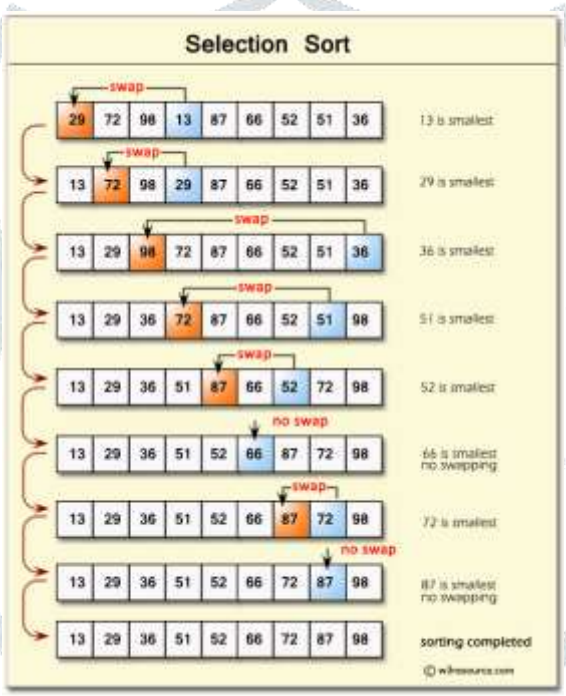

Figure 2. Working of Selection Sort

Insertion Sort: Decrease and conquer method is used to sort the elements using insertion sort. Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. The algorithm is more useful if the first few objects are already sorted, an unsorted object can be inserted in the sorted set-in proper Place.

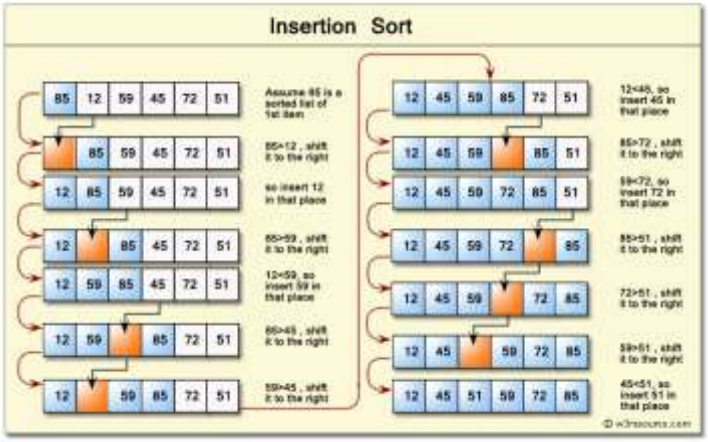

Figure 3. Working of Insertion Sort

Quick Sort: Quick-sort uses Divide and Conquer technique. It picks an element as pivot element and partitions the given array around the picked pivot element. There are many different versions of quick-sort that pick the pivot element in different ways. Always pick the first element as a pivot element. Always pick the last element as pivot elements. Pick a random element as a pivot element. Pick median as pivot element. The sorting technique is best suited to sort a large number of data. This performs better than Merge Sort and Heap Sort which have the same asymptotic time complexity O(n log n) on average but the constant factors hidden in the asymptotic time complexity for quicksort are pretty small. The algorithm is implemented in medical monitoring systems, Google pages for fast retrieval, life support or control systems etc.

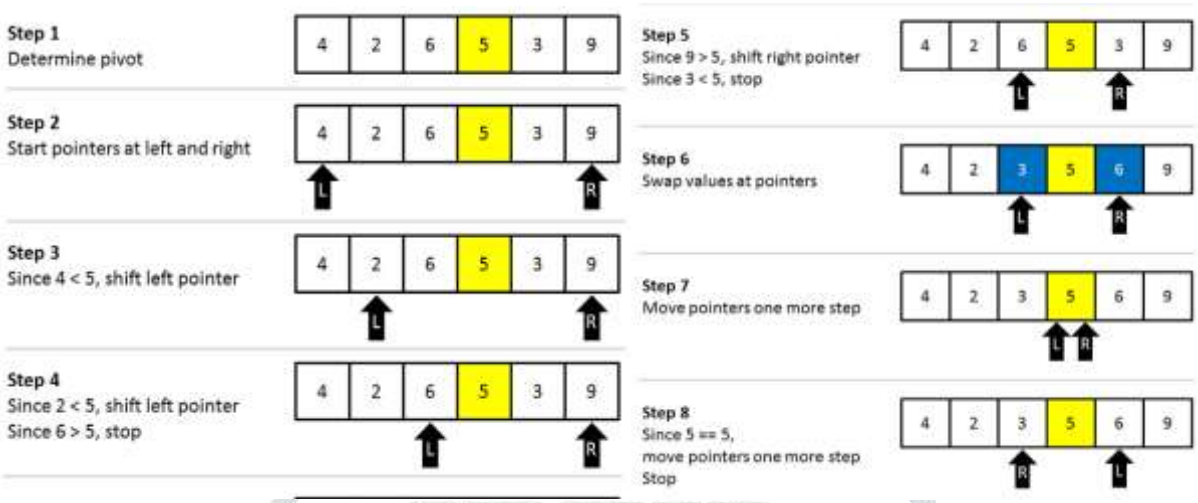

Figure 4. Working of Quick Sort

Merge Sort: Merge Sort uses Divide and Conquer algorithms. It divides the input array into two halves, calls itself for the two sorted halves and then merges the two sorted halves. The author in [2] has discussed the optimum solution of merge sort and explained the sequence in detail.

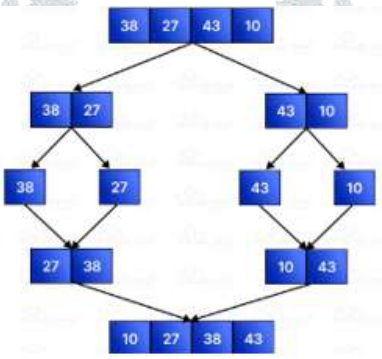

Figure 5. Working of Merge Sort

Heap Sort: The method used by the heap sort, a comparison-based sorting algorithm, is to divide an array into sorted and unsorted regions, repeatedly reducing the size of the unsorted part by removing the largest element and shifting it to the sorted region. Relying on the binary heap data structure, it creates a max-heap from the array, a binary tree in which the parent nodes are always bigger than the offspring. The last element in the array is then swapped for the largest element, and the heap is then re-qualified. Heap sort is effective for handling huge datasets and comes with a worst-case time complexity of O(nlogn).

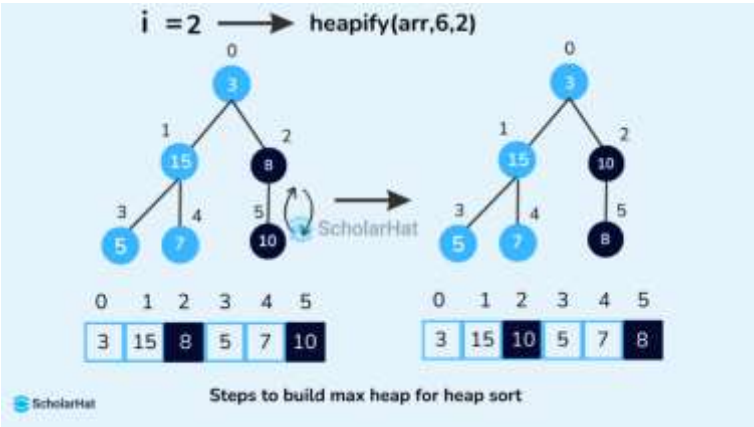

Figure 6. Working of Heap Sort

Radix Sort: A non-comparison-based sorting technique called radix sort sorts items one digit at a time, beginning with the least important digit. When working with integer values, it groups the elements according to their digits, then sorts each group according to the next digit, and so on, finishing with the highest significant digit. Radix sort is effective for handling huge data sets since it possesses a worst-case time complexity of O(kn), wherein k is the greatest number of digits in the elements. It works well when sorting elements that have a defined amount of numbers or when sorting elements according to a certain characteristic

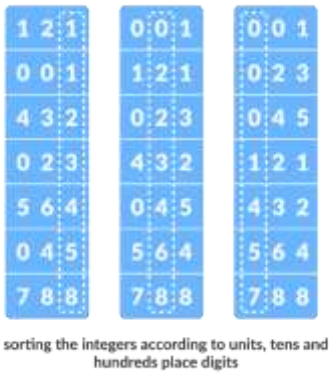

Figure 7. Working of Radix Sort

Counting Sort: Counting sort is an integer sorting algorithm with linear running time complexity. For some integer K, counting sort assumes that each element is an integer in the range 1 to K. It works by counting the number of occurrences of each element in the input, usually called keys, and store this information in another array, we need to modify the array to identify the location of each key value in the output sequence, use the prefix sum on the counts. Counting sort is not a comparison sort, and it preserves the relative order of the elements with equal keys.

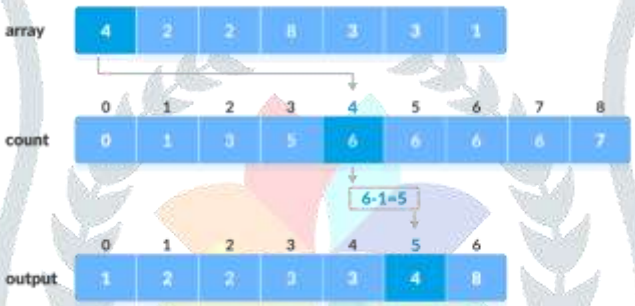

Figure 8. Working of Counting Sort

III. RESULTS AND DISCUSSION

3.1 FOR NUMERIC DATA:

For Numeric data we take 10 elements in Low Load, 100 elements in Average Load, 1000 elements in Intermediate Load and 10000 elements in High Load and the performance was recorded for each sorting algorithm shown in the graph of following cases.

3.1.1 BEST CASE:

In the best case, we take an ordered (sorted) data set for observing performance of different algorithms and by analyzing the performance of each sorting algorithm. We finalize the result that selection sort performs better in every case when data is in ordered form.

Table 1: Complexity time of various Sorting Algorithm for Numeric Ordered Data

S. NO.	SORTING ALGORITHMS	NUMBER OF ELEMENTS			
		10	100	1000	10000
1	Bubble	600	600	610	610
2	Selection	420	420	420	430
3	Insertion	470	480	480	480
4	Quick	610	610	630	630
5	Merge	590	590	600	620
6	Radix	580	580	590	590
7	Heap	580	590	600	600
8	Counting	700	730	730	730

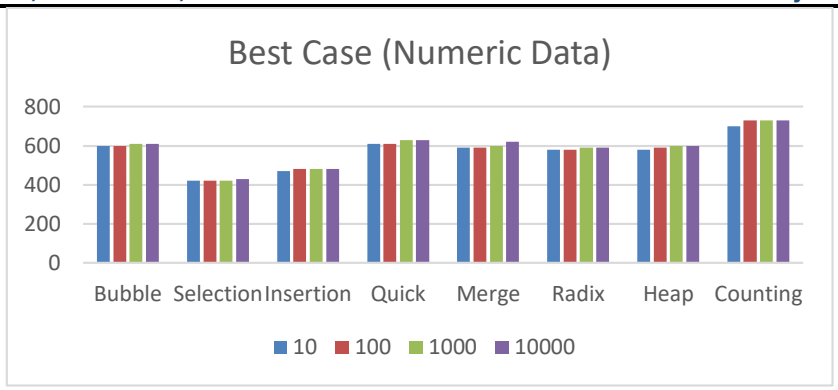

Figure 9: Graphical representation of Complexity time for Numeric Ordered Data

3.1.2 WORST CASE:

The unordered (unsorted) set of numeric data was taken by us for observing performance of different algorithms. In this Analysis, Quick Sort and Merge Sort perform better than other sorting algorithms. The Quick Sort performs much better than the Merge Sort in Heavy Load.

Table 2: Complexity time of various Sorting Algorithm for Numeric Unordered Data

S. NO.	SORTING ALGORITHMS	NUMBER OF ELEMENTS			
		10	100	1000	10000
1	Bubble	590	590	600	600
2	Selection	430	430	440	440
3	Insertion	440	440	450	450
4	Quick	340	360	360	360
5	Merge	400	400	410	410
6	Radix	670	670	670	730
7	Heap	470	470	480	480
8	Counting	700	700	710	720

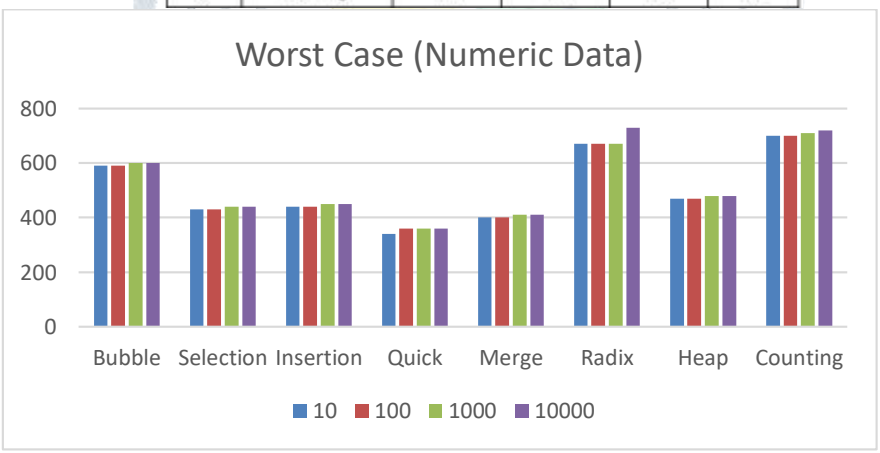

Figure 10: Graphical representation of Complexity time for Numeric Unordered Data

3.2 FOR STRING DATA:

For String data, we take 10 elements in Low Load, 50 elements in Intermediate Load and 100 elements in High Load and the performance of different algorithms was recorded. The results were recorded for two cases in which the one case was Best case and the other one was Worst Case.

3.2.1 BEST CASE:

We take an ordered (sorted) data set of strings in each load for observing performance of different algorithms. Here, the sorted data means the string data in alphabetical order. On the basis of a given data set, the performance was recorded and shown in graphs. The Selection Sort performs much better than the rest, when data given was sorted.

Table 3: Complexity time of various Sorting Algorithm for String Ordered Data

S. NO.	SORTING ALGORITHMS	NUMBER OF ELEMENTS		
		10	50	100
1	Bubble	560	560	580
2	Selection	470	470	470
3	Insertion	520	520	520
4	Quick	520	520	530
5	Merge	530	530	530
6	Radix	560	580	580
7	Heap	590	590	590
8	Counting	530	540	540

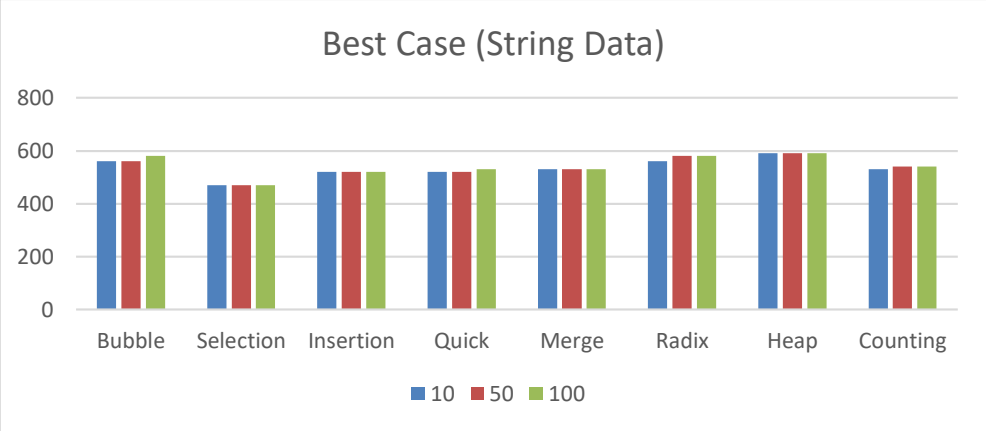

Figure 9: Graphical representation of Complexity time for String Ordered Data

3.2.2 WORST CASE:

We take an unordered (unsorted) data set of strings for observing performance of different algorithms. The observed result was recorded in a table and shown in the graph. The analysis of algorithms in worst case concludes that the quick sort performs better than the rest of the algorithms in each load.

Table 4: Complexity time of various Sorting Algorithm for String Unordered Data

S. NO.	SORTING ALGORITHMS	NUMBER OF ELEMENTS		
		10	50	100
1	Bubble	840	840	850
2	Selection	770	780	780
3	Insertion	720	720	730
4	Quick	650	670	670
5	Merge	740	740	750
6	Radix	770	780	780
7	Heap	700	720	720
8	Counting	750	750	750

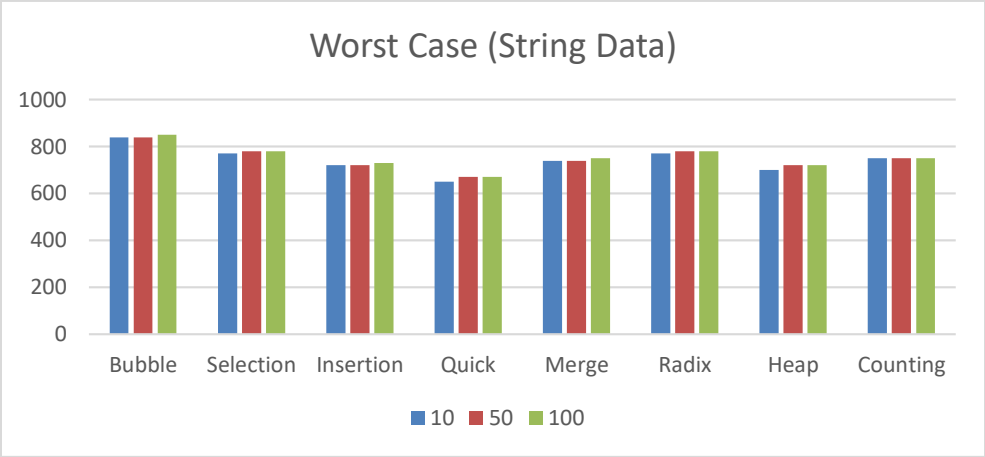

Figure 10: Graphical representation of Complexity time for String Unordered Data

IV CONCLUSION:

In this research paper we compared various sorting algorithms in respect to their execution time. The efficiency of every algorithm varies with the number of input or load and we have compared the execution of the algorithm so that we can see which algorithm is best to use. The sorting algorithms are evaluated in C++. From our experimental results we identified that in a Best case or in a sorted data set of both String and Numeric data type, the Selection Sort behaves optimal. But for the Worst case or for unsorted data sets of both String and Numeric Data, the Quick and Merge sort behaves better than others, especially the Quick Sort. So, the final conclusion of our analysis was, it is optimal to use Selection Sort in sorted data and it is optimal to use Quick Sort in unsorted data.

V REFERENCES

[1] ZHANG YIWEN, TAN JI, "ANALYSIS AND IMPROVEMENT ON SIMPLE SELECTION SORT ALGORITHM ", SILICON VALLE, NO. 18, PP. 77- 94, 2009.
[2] YAN WEIMIN, WU WEIMIN "DATA STRUCTURES "IN, BEIJING: TSINGHUA UNIVERSITY PRESS., PP. 263-278, 2000.
[3] Kowalk W.P. (2011) Insertion Sort. In: Vöcking B. et al. (eds) Algorithms Unplugged. Springer, Berlin, Heidelberg.
[4] Min Wang, "Analysis on bubble sort algorithm optimization ", 2010 International Forum on Information Technology and Applications, July 2010.
[5] U. Ameta, M. Patel and A. K. Sharma, "Scaled Agile Framework Implementation in Organizations', its Shortcomings and an AI Based Solution to Track Team's Performance," *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*, Bangalore, India, 2022, pp. 1-7, doi: 10.1109/GCAT55367.2022.9971968
[6] Patel M (2018) Data Structure and Algorithm With C. Educreation Publishing
[7] Weik M.H. (2000) merge sort. In: Computer Science and Communications Dictionary. Springer, Boston, MA
[8] Jehad, M. Rami "An Enhancement of Major Sorting Algorithms," The International Arab Journal of Information Technology, Vol. 7, No. 1, January 2010