Conventional Vs Enhanced Sorting Algorithm: A Review

Pooja Gupta
Assistant Professor, Uttaranchal University, Dehradun, Uttarakhand, India

Abstract: **Sorting problem is one of the most antique problems of computer science. From the beginning of computation, algorithms for sorting problem has been derived and analyzed by many researchers. The first sorting algorithm derived was bubble sort (1956). Many useful sorting algorithms are continually being invented like Merge sort, Timsort (2002), Library sort (2006). A vast number of sorting algorithms and their enhancements exists in the literature. One would really like to know that these enhancements of sorting algorithms are actually better than the conventional sorting algorithms. For this purpose authors have taken a case of classical merge sort and enhanced merge sort algorithm proposed by paira et al [12]. authors have tested the performance of both the algorithms using different random number distributions and found that there is no significant difference between the algorithms.**

Keywords: **Algorithm, Sorting, Efficiency, Time Complexity, Space Complexity**

I. INTRODUCTION

In computer science sorting problem has been researched at length. Several sorting algorithms and enhancement of classical sorting algorithm have been proposed, for example, merge sort was proposed in 1981 and its enhancement algorithm is proposed in 2016. Several enhancements of classical sorting algorithms have been presented in the literature [1, 2, and 3]. Out of all available sorting algorithms which one is most suitable for an application depends on many parameters like input size, type of data and distribution of data [4]. Other factors that influence the performance of sorting algorithm are number of comparison operations, number of swaps required, and memroy space [5]. In literature, there are many sorting algorithms to solve a particular problem and there is a drastic difference in their performance and efficiency. The difference in efficiency is much more important than differences due to changes in hardware and software of the system [6]. [7, 8] derived quick sort algorithm and many researchers considered QuickSort algorithm to be the fastest sorting algorithms [9, 10, 11]. The particular algorithm one chooses depends on the properties of data and the operations one may perform on the data [5].

The different cases with regard to Running Time that is popular in sorting algorithms are:

- O (n) is average case; average number of steps taken to solve an input of size n.
- O (n log n) is best case: this most efficient case, the minimum number of steps taken to solve an input of size n.
- O (n^2) is worst case: This is inefficient case; maximum number of steps taken to solve an input of size n.

Performance of a sorting algorithm is measured using big-Oh notaiton, i.e Worst time complexity of the algorithm. Some algorithms are quadratic means they fall in the category of O (n2) like insertion sort, bubble sort and selection sort. These algorithms are simple to implement, iterative and spontaneous. Other algorithms like quick sort, merge sort and heap sort falls into the category of O(n log n), they are normally complex to implement but these are faster than previous category [12].

Every sorting algorithm is problem centric [13]. Most of the algorithms are popular for sorting the unordered lists. The efficiency of the sorting algorithms is to optimize the importance of other sorting algorithms [5]. The optimality of these sorting algorithms is judged while calculating their time and space complexities [14].

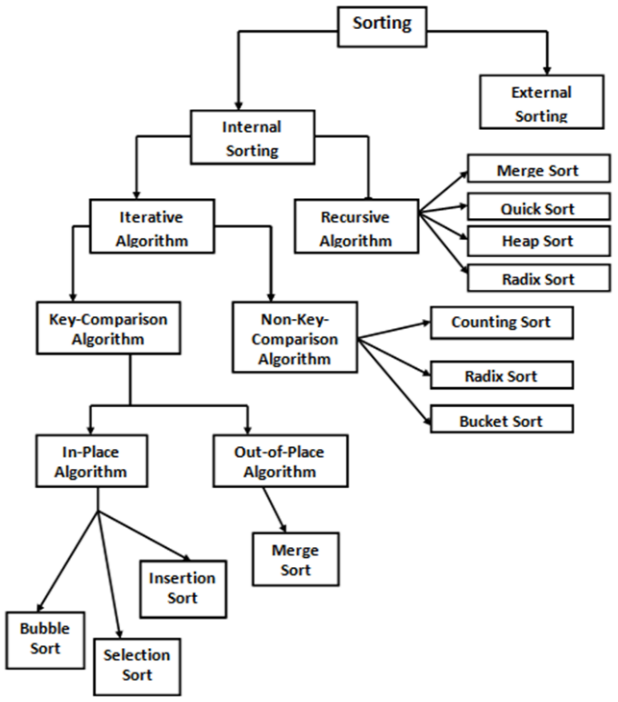

Figure 1.1: Classification of Sorting

The objective of this paper is to provide a more comprehensive and systematic review of the literature pertaining to Sorting algorithms available in computer science. According to Paira et al, the newly proposed Merge sort algorithm is faster than the conventional Merge Sort algorithm having a time complexity of O(n $\log_2$ n) [12]. The main objective is to scan the performance of this proposed Merge-sort algorithm against the conventional Merge sort algorithm proposed by Qin et al. In order to realize this objective a comparative analysis was conducted to check the performance of algorithms using different sets of input.

II. LITERATURE SURVEY

There are various types of sorting algorithms available. We can categorize sorting algorithms using different parameters

- Internal and External
- Iterative and Recursive
- Stable and unstable
- Linear and quadratic
- Comparison and non-key comparisons
- In Place and Out of Place

Every sorting algorithm has its own advantage and disadvantage [15]. Selection sort, Bubble sort, insertion sort, quick sort, merges sort, etc are key-comparison sorting algorithms, whereas counting sort, bucket sort and radix sort is non-key comparison sorting algorithms.

Iterative approach algorithms are simple to implement. Recursive algorithms are complex algorithms, but more efficient than the iterative algorithms [12].

Name	Time complexity			Space Complexity	Stable	Principle	Key-comparison Based	In-place/Out-Of-Place
	Best	Average	Worst					
Bubble Sort	Ω(n)	$\Theta(n^2)$	$O(n^2)$	O(1)	Yes	Iterative	Yes	In Place
Insertion Sort	Ω(n)	$\Theta(n^2)$	$O(n^2)$	O(1)	Yes	Iterative	Yes	In Place
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	O(1)	No	Iterative	Yes	In Place
BCIS	Ω(n)	$\Theta(n^{1.5})$	$O(n^2)$	---	No	Divide and Conquer	Yes	In place
Cocktail Sort	Ω(n)	$\Theta(n^2)$	$O(n^2)$	O(1)	Yes	Exchanging	No	In place
Counting Sort	Ω (n+k)	Θ (n+k)	O(n+k)	O(n+k)	Yes	Integer sorting	No	Out-of-Place
Radix Sort	Ω (wn)	Θ (wn)	O(wn)	O(n+r) auxiliary	Yes	uniform distribution of elements	No	---
Bucket/Bin Sort	Ω (n+k)	Θ (n+k)	$O(n^2)$	O(n)	Yes	uniform distribution of elements	No	Out-of-Place
Merge Sort	Ω (n log n)	Θ (n log n)	O(n log n)	O(n)	yes	Recursive Divide and Conquer	yes	Out-of-Place
Quick Sort	Ω (n log n)	Θ (n log n)	O(n2)	O(logn)	No	Recursive Partitioning	Yes	Out-of-Place
Heap Sort	Ω (n log n	Θ (n log n)	O(n log n)	O(1)	No	Recursive	Yes	In Place
Tim Sort	Ω(n)	Θ (n log n)	O(n log n)	O(n)	yes	Insertion & Merging	Yes	Out-of-Place
Library Sort	Ω(n)	Θ (n log n)	$O(n^2)$	O(n)	yes	Insertion	No	

Table 1.1

*n = number of elements to be sorted
*k = number of possible values in the range.
*w = word size
*r = radix

According to Sareen et al, every sorting algorithm has its advantage and disadvantage, but the quick sort algorithm is most efficient algorithm among all existing sorting algorithms. They developed a program to show the running time comparison of different sorting algorithms. They measured the running time in microseconds; they used different input size of each algorithm and then calculated the average running time for each algorithm [15].

Jehad et al proposed two enhanced versions of selection sort and bubble sort, namely: Enhanced Selection Sort (ESS) and Enhanced Bubble Sort (EBS). Although their ESS has same running time complexity (O (n^2)) but they emphasized that ESS is faster than selection sort.they proved it by storing input data in secondary memory.

Their second proposed algorithm Enhanced Bubble sort running time is O(*nlgn*) which is obviously faster than traditional Bubble sort time (O (n^2)).

Performance of both enhanced algorithms was evaluated by applying them on a real-world case study of sorting a database of (12500) records and experimental results proved that the EBS is more efficienent than shell sort and enhanced shell sort algorithms [5].

Sodhi et al [16], improvised insertion sorting algorithm and proposed more efficient and enhanced insertion sort. This enhanced insertion sort's worst case running time complexity is less than O $(n^{1.585})$. It also reduces the number of comparisons by half. The authors compared proposed algorithm with some bubble sort, selection sort etc. its complexity varies from O (n) to $O(n^{1.585})$.

Khairullah et al [17] worked upon bubble sort, selection sort and insertion sort and proposed enhanced sorting algorithms. They proved the efficiency of their proposed algorithms over classical algorithms. To reduce the number of comparisons, they suggested that by skipping or minimizing operations (comparisons or swap operations) performance of typical inefficient algorithms can be enhanced. Selection sort is enhanced by the reduction of swap operations in [14].

Cocktail sort, also known as shaker sort of dual selection sort, is a bidirectional version of selection sort. Cocktails sort algorithm finds both the lowest and largest elements in the array in each iteration [19-21]. The inner loop inside the nested loop of insertion sort can be simplified by using a sentinel value.

Merge sort algorithm uses divide-and-conquer principle and recursively sorts the array. It was introduced by Qin et al [19]. They showed that merge sort has a time complexity of O *(n*logn) and its time complexity is lesser as compared to an iterative sorting algorithm like Bubble, Insertion and Selection which have a quadratic time complexity. Merge sort works well with large input size where as above iterative algorithm limits their use of small input.

Khalid et al [22] suggested that quick sort perform fastest for large input, whereas selection sort is the slowest. They performed an assessment and analyzed the performance of Selection sort, Insertion sort, merge sort, quick sort and bubble sort. For smaller input size all all above sorting algorithm behaved similarly.

For large input Merge Sort performs better than other sorting algorithms [22]. The only problem with Merge sort is this it requires an additional auxiliary array for storing the elements and that is why is is called out of place sorting algorithm as it needs extra space for sorting. Its space complexity is O(n) [18]. If we compare the space complexity of Merge sort with quick sort then Quick Sort is more efficient than the Merge Sort as it is an in place sorting algorithm. But, Quick sort has its own disadvantage, its worst case time complexity is $O(n^2)$ due to the unbalanced partition of the array [19]. Heap sort performs better than the Quick Sort as it has a time complexity of $O(n \log_2 n)$. But it is extremely unstable [20].

Sorting algorithms can be categorized as iterative sorting algorithms (bubble sort, selection sort, and insertion sort) and recursive sorting algorithms (Merge sort and Quick sort). Implementation of iterative algorithms is simpler than recursive ones. According to Paira et al [18], the iterative sorting algorithms, in the worst case, examine almost the entire list of arrays resulting into time complexity of $O(n^2)$. Insertion sort is the most efficient iterative sorting algorithm for small input. Its efficiency increased if the input is partially sorted decreases with the input size.

Apart form Enhanced Insertion sort there is one more modified version of Insertion sort i.e. Doubly Inserted sort. This doubly insertion sort works in two-way. It scans two data elements simultaneously. It scans the first and the last element of the array and compares them and then apply conventional insertion sort on both. This doubly Inserted sort is better than all iterative sorting algorithms as its worst case running time is O(n).

In fact Doubly Inserted Sort performs better than Merge sort and Quick Sort as its worst case time complexity and space complexity is O(n) and O(1) respectively [18].

[4] Proposed a better insertion sort algorithm, they named it Bidirectional Conditional Insertion Sort (BCIS). BCIS reduce the shifting operation of classical Insertion sorts. BCIS assume that there are two already sorted lists on the left and the right side of the array, whereas the unsorted list located between these two sorted lists. For ascendling sorting all the small elements should be located in the left sub-list and all the large elements should be located in the right sub-list.

One sorted part in the classical insertion sort is now distributed into two sorted parts in BCIS. These two sorted lists made BCIS economical in terms of memory usage. There is one more benefit of BCIS; it can insert more than one element in their correct positions in one loop iteration as compared to classical insertion sort.

III. METHODOLODY AND IMPLEMENTATION

In order to measure the actual time improvement achieved by the proposed algorithm, profiling of the enhanced merge sort algorithm and existing merge sort algorithm will be done. A C++ Program will be used to measure the performance of the both the algorithms. Profiling experiments will be carried out on random data sets.

Characteristics of Merge Sort Algorithm:

- Divide and Conquer
- Recursive
- Stable
- Out of place
- Space Complexity: O(n)
- Time Complexity: O (n log n)

Steps for classical Merge sort:

- Input the unsorted array
- Divide array into two halves.
- Sort each half recursively.
- Perform merging of two halves.
- Output the sorted array

Pros:

- Merge Sort is better than Quick Sort in terms of time complexity.
- It is used for both internal and external sorting

Cons:

- An extra memory requirement as it is recursive.
- High space complexity.

Proposed Merge sort algorithm

The authors applied Max-Min algorithm principle with classical Merge sort to modify and propose new Merge sort algorithm. According to them its time complexity is same as of the classical merge sort, but its performance is better for all input sizes.

Steps for Proposed Merge sort:

- Input the unsorted array
- Pair-wise sorting of every element of the array
- Divide the given array into two halves taking into consideration odd-even positions of the original array
- Repeat the above steps until the complete array is divided into sub-lists containing two elements each
- Perform merging of two halves
- Output the sorted array

Pros:

- Executes faster than classical Merge sort algorithm for all input sizes.

Cons:

- It requires a stack space of O (n).

Implementation:

In order to test the speed of the different sorting algorithm authors made a C++ program which runs each algorithm several times for randomly-generated arrays. The test was run in an Intel(R) Pentium ® Dual CPU E2180 @2.00GHz, and the program was compiled using the Online C++ compiler for generating larger size of Input numbers (upto 40000). Cpp.sh is a simple front-end for a GCC compiler [20]. The system uses GCC 4.9.2.

Ten different array sizes were used: 4000, 8000, 12000, 16000, 20000, 24000, 28000, 32000, 36000 and 40000. Random numbers between 0 and 999.

Three cases with different random number distributions were used:

Case 1: Completely random.

Case 2: All sorted.

Case 3: All reversed

Each case is explained with the help of a Bar chart and a table of numbers. This table presents Input size, CPU time elapsed, the number of comparison and assignment operations performed by the both sorting algorithms *on average* during the sorting of one input array. In case of integers, the number of comparisons and assignment operations and their relative ratio is not particularly significant as comparison is fast. But, when a comparison of the elements is a complex process then it may be important to compare different sorting algorithms with respect to the number of total operations during sorting.

Each table has the following format:

	CPU Time		**No. of Comparisons**		**No. of Assignments**	
Input Size	MergeSort	Proposed Algo.	MergeSort	Proposed Algo.	MergeSort	Proposed Algo.
(value)	(value)	(value)	(value)	(value)	(value)	(value)

Table 3.1: Sample table format

Comparative execution analysis

Each bar in the charts represents the time spent (in milliseconds) by the sorting algorithm, in average, to sort the array once. Lower bars are better.

Case 1:

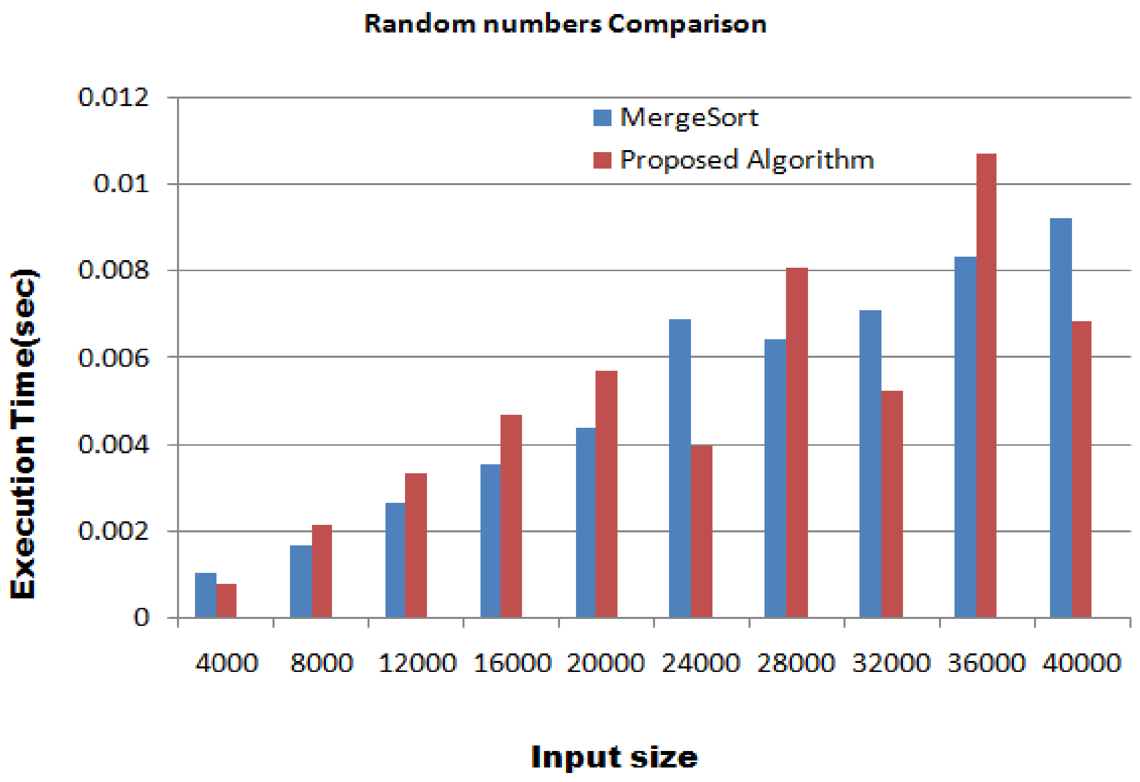

Figure 3.1: Comparative execution times of both Sorting algorithms for sorting random numbers

	CPU Time		**No. of Comparisons**		**No. of Assignments**	
Input Size	MergeSort	Proposed Algo.	MergeSort	Proposed Algo.	MergeSort	Proposed Algo.
4000	0.001034	0.000785	63127	42803	179250	95808
8000	0.001676	0.002127	93576	138509	207616	388380
12000	0.002646	0.00333	147733	218706	327232	609289
16000	0.003515	0.00466	203234	301148	447232	836444
20000	0.004383	0.005704	260984	388732	574464	1069387
24000	0.006875	0.003941	319407	474031	702464	1308048
28000	0.006435	0.008075	378776	564095	830464	1547520
32000	0.007101	0.005232	438603	8126301	958464	21876040
36000	0.008321	0.010709	499913	747212	1092928	2038310
40000	0.009229	0.006852	561947	836304	1228928	2287834

Table 3.2: Comparative Analysis in terms of CPU Time, Comparison and Assignment operations for random numbers

Case 2:

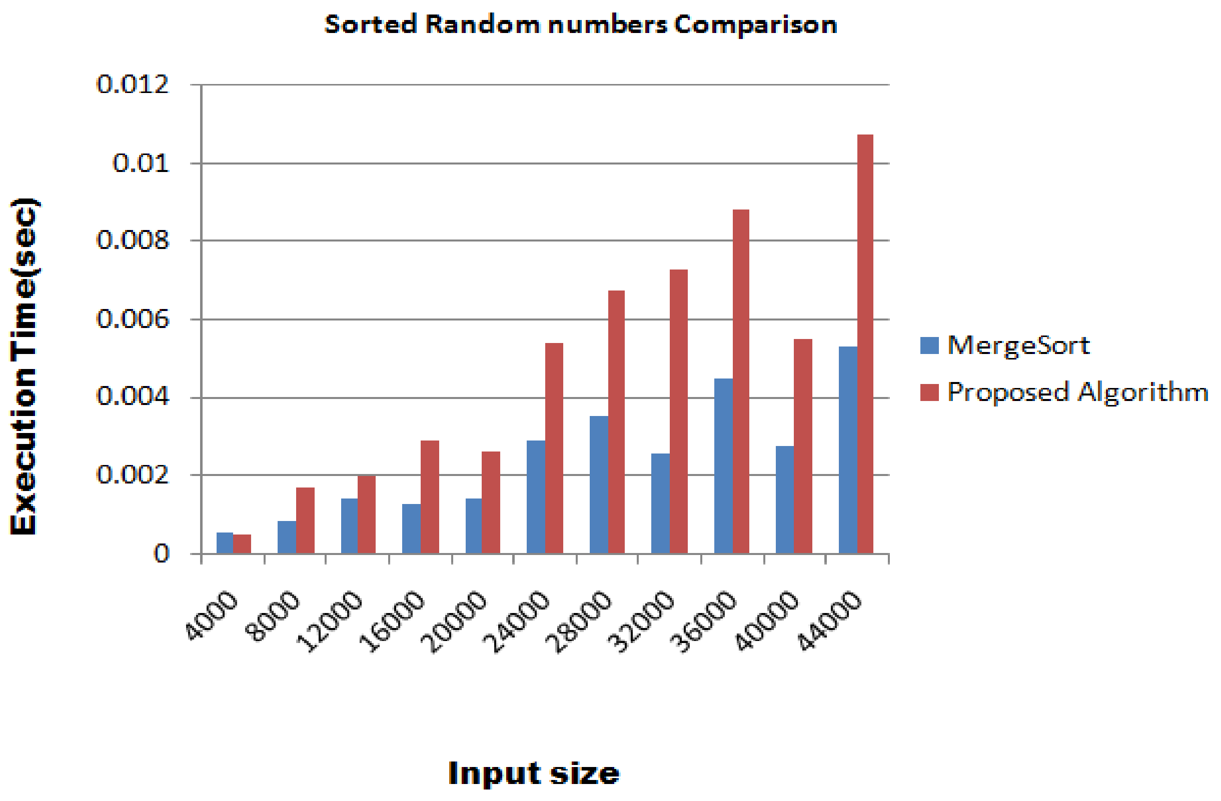

Figure 3.2: Comparative execution times of both Sorting algorithms for sorting sorted numbers

	CPU Time		No. of Comparisons		No. of Assignments	
Input Size	MergeSort	Proposed Algo.	MergeSort	Proposed Algo.	MergeSort	Proposed Algo.
4000	0.000125	0.000498	26242	67632	95808	167440
8000	0.000542	0.001684	57668	147264	207616	362880
12000	0.000798	0.001984	89603	230924	327232	570160
16000	0.001237	0.002893	124852	318519	447232	781760
20000	0.00138	0.002618	157808	406432	574464	1000912
24000	0.002894	0.005413	192810	497836	702464	1224320
28000	0.003501	0.006752	228373	589713	830464	1448208
32000	0.002577	0.007261	267004	685028	958464	1675520
36000	0.004466	0.008812	300830	777706	1092928	1906672
40000	0.002734	0.005503	336684	872847	1228928	2141824

Table 3.3: Comparative Analysis in terms of CPU Time, Comparison and Assignment operations for sorted numbers

Case 3:

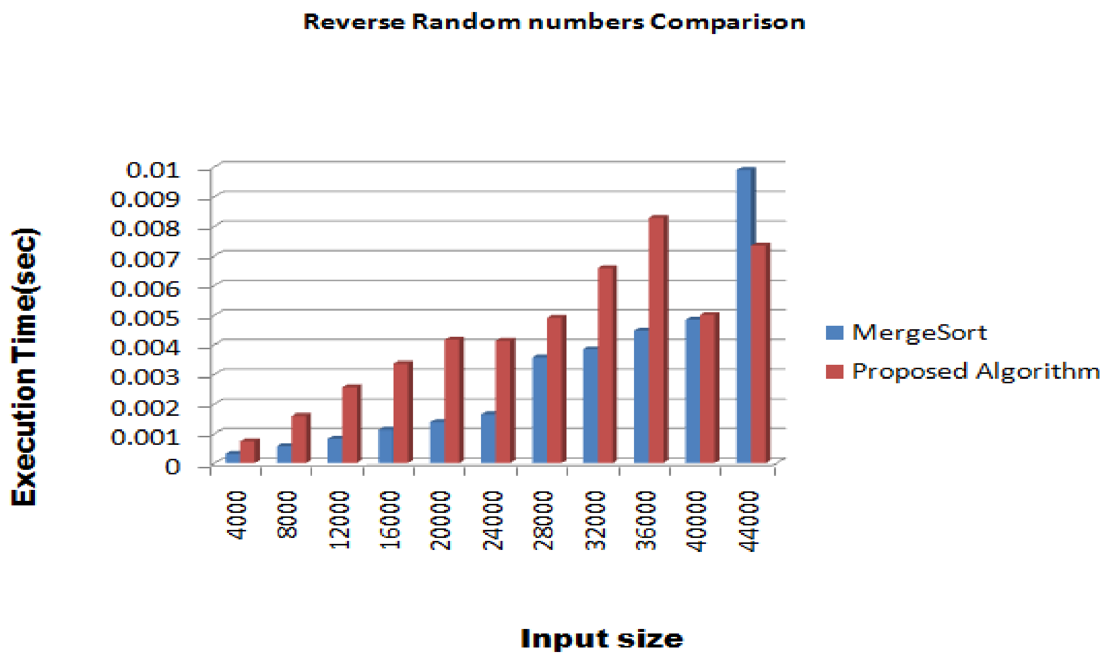

Figure 3.3: Comparative execution times of both Sorting algorithms for sorting reversed sorted numbers

	CPU Time		No. of Comparisons		No. of Assignments	
Input Size	MergeSort	Proposed Algo.	Merge Sort	Proposed Algo.	MergeSort	Proposed Algo.
4000	0.000318	0.000739	26242	67632	95808	167440
8000	0.000563	0.001591	57668	147264	207616	362880
12000	0.000821	0.002562	89603	230924	327232	570160
16000	0.001126	0.003361	124852	318519	447232	781760
20000	0.001382	0.004162	157808	406432	574464	1000912
24000	0.001646	0.004129	192810	497836	702464	1224320
28000	0.003568	0.004909	228373	589713	830464	1448208
32000	0.003845	0.006581	267004	685028	958464	1675520
36000	0.004478	0.008277	300830	777706	1092928	1906672
40000	0.004840	0.005006	336684	872847	1228928	2141824

Table 3.4: Comparative Analysis in terms of CPU Time, Comparison and Assignment operations for reversed numbers

IV. CONCLUSION

The efficiency of the sorting algorithms is to optimize the importance of other sorting algorithms [13]. The optimality of these sorting algorithms is judged while calculating their time and space complexities [21].

The main objective of this paper was to compare the conventional Merge Sort Algorithm with enhanced Merge sort algorithm.

According to Paira et al, it is observed that when n=4000, both Merge Sort and the proposed algorithm takes the same time to execute. But as 'n' is increasing execution time of Merge Sort > Execution time of the proposed algorithm [21].

We have performed the comparison of both the algorithms for different input sizes (4000, 8000, 12000, 16000, 20000…………………..40000).

We have also used three different random number distributions:

1. Completely random.
2. All sorted.
3. All reversed:

But after executing both of the algorithms on the Turbo C++ compiler as well as on GCC compiler online, we found out that there is no significant difference between the algorithms, however, as the input size increases (when n>4000) we found that conventional merge sort algorithm's execution time is slightly better than the proposed algorithm in all the cases and Conventional merge sort algorithm uses the lesser number of comparison and assignment operations as compared to proposed algorithm.

REFERENCES

[1]. Min, Wang. "Analysis on bubble sort algorithm optimization." In Information Technology and Applications (IFITA), 2010 International Forum on, vol. 1, pp. 208-211. IEEE, 2010.

[2]. Coding Unit Programming Tutorials. Cocktail sort algorithm or shaker sort algorithm. http://www.codingunit.com/cocktail-sort-algorithm-or-shaker-sort-algorithm, (accessed on January 1, 2017).

[3]. Debasis Samanta. Classic Data Structures. PHI Learning Pvt. Ltd., 2nd edition, 2009.

[4]. Mohammed, Adnan Saher, Şahin Emrah Amrahov, and Fatih V. Çelebi. "Bidirectional Conditional Insertion Sort algorithm; An efficient progress on the classical insertion sort." Future Generation Computer Systems 71 (2017): 102-112

[5]. Alnihoud, Jehad, and Rami Mansi. "An Enhancement of Major Sorting Algorithms." Int. Arab J. Inf. Technol. 7, no. 1 (2010): 55-62.

[6]. Thomas H cormen "Introduction to Algorithms", Third Edition, ISBN 978-0-262-03384-8, Chapter 1, Page 5-7.

[7]. Hoare, C. A. R. (1961). Partition: Algorithm 63, Quicksort: Algorithm 64, and Find: Algorithm 65. Comm. ACM. 4(7), 321-322

[8]. Hoare R.(1962). Quicksort. the Computer Journal, 4(1), 10-15

[9]. Sedgewick R. (1977). QuickSort with Equal Keys. Siam J Comput.,6: 240-287

[10]. Van Emden M. H. (1970). Algorithms 402: Increasing the efficiency of Quicksort. Communications of the ACM, 563-567

[11]. Knuth D.E. (2005). The Art of Computer Programming. Vol. 3: Sorting and Searching, Addison-Wesley, Reading, Mass

[12]. Paira, Smita, Sourabh Chandra, and SK Safikul Alam. "Enhanced Merge Sort-a new approach to the merging process." Procedia Computer Science 93 (2016): 982-987.

[13]. Paira, Smita, Sourabh Chandra, Sk Safikul Alam, and Partha Sarthi Dey. "A Review Report on Divide and Conquer Sorting Algorithm." In National Conference on Electrical, Electronics, and Computer Engineering, ISBN, pp. 978-93.

[14]. Horowitz, Ellis, and Alessandro Zorat. "Divide-and-conquer for parallel processing." IEEE Transactions on Computers 32, no. 6 (1983): 582-585.

[15]. Sareen, Pankaj. "Comparison of sorting algorithms (on the basis of average case)." International Journal of Advanced Research in Computer Science and Software Engineering 3, no. 3 (2013): 522-532.

[16]. Sodhi, Tarundeep Singh, Surmeet Kaur, and Snehdeep Kaur. "Enhanced insertion sort algorithm." International journal of Computer applications 64, no. 21 (2013).

[17]. Khairullah, Md. "Enhancing worst sorting algorithms." (2013).

[18]. Paira, Smita, Anisha Agarwal, Sk Safikul Alam, and Sourabh Chandra. "Doubly Inserted Sort: A Partially Insertion Based Dual Scanned Sorting Algorithm." In Emerging Research in Computing, Information, Communication and Applications, pp. 11-19. Springer, New Delhi, 2015.

[19]. Song Qin Merge Sort Agorithm CS.fit.edu Available: cs.fit.edu/~pkc/ classes/writing/hw13/song.pdf.

[20]. Yadav, Rohit, Kratika Varshney, and Nitin Kr Verma. "Analysis of Recursive and Non-recursive Merge Sort Algorithm." International Journal of Advanced Research in Computer Science and Software Engineering 3, no. 11 (2013).

[21]. Paira, S., S. Chandra, S. S. Alam, and S. S. Patra. "Max min sorting algorithm—a new approach of sorting." Int. J. Technol. Explor. Learn.(IJTEL) 3, no. 2 (2014): 405-408.

[22]. Al-Kharabsheh, Khalid Suleiman, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani, and Nabeel Imhammed Zanoon. "Review on sorting algorithms a comparative study." International Journal of Computer Science and Security (IJCSS) 7, no. 3 (2013): 120-126.