# Enhanced Relative Comparison of Traditional Sorting Approaches towards Optimization of New Hybrid Two-in-One (OHTO) Novel Sorting Technique

**Rajeshwari B S[1,*], C.B. Yogeesha[2], M. Vaishnavi[3], Yashita P. Jain[3], B.V. Ramyashree[3] & Arpith Kumar[3]**

[1]Department of CSE, B.M.S College of Engineering, Bull Temple Road, Basavanagudi, Bengaluru 560 019, India
[2]Collaboration Technology Group, Cisco Systems India Private Limited, SEZ, Cessana Business Park, Marathahalli, Sarjapur, ORR, Kaadubesanahalli, Bengaluru 560 103, India
[3]Department of CSE, B.M.S College of Engineering, Bull Temple Road, Basavanagudi, Bengaluru 560 019, India
*E-mail: rajeshwari.cse@bmsce.ac.in,

**Abstract.** In the world of computer technology, sorting is an operation on a data set that involves ordering it in an increasing or decreasing fashion according to some linear relationship among the data items. With the rise in the generation of big data, the concept of big numbers has come into existence. When the number of records to be sorted is limited to thousands, traditional sorting approaches can be used; in such cases, complexities in their execution time can be ignored. However, in the case of big data, where processing times for billions or trillions of records are very long, time complexity is very significant. Therefore, an optimized sorting technique with efficient time complexity is very much required. Hence, in this paper an optimized sorting technique is proposed, named Optimized Hybrid Two-in-One Novel Sorting Technique (OHTO, a mixed approach of the Insertion Sort technique and the Bubble Sort technique. The proposed sorting technique uses the procedure of both Bubble Sort and Insertion Sort, resulting in fewer comparisons, fewer data movements, fewer data insertions, and less time complexity for any given input data set compared to existing sorting techniques.

**Keywords**: *algorithm design technique; bubble sort; hybrid sorting; insertion sort; optimization; sorting; time complexity*.

## 1 Introduction

Sorting plays an important role in organizing the elements of a data set in a particular order, most often in numerical or lexicographical order, to facilitate further analysis whose procedures may require sorted input. Over the years, sorting algorithms have emerged as an integral part of computing as it reduces the complexity of problems, thus providing a great way of preprocessing data. Despite the existence of a plethora of sorting techniques, newer algorithms keep

Received June 8th, 2022, 1st Revision September 13th, 2022, 2nd Revision October 23rd, 2022, 3rd Revision May 11th, 2023, Accepted for publication May 15th, 2023.

getting introduced, as work is going on continually to find better ones with more efficiency and optimization. The efficiency of any algorithm mainly depends on the number of data items being processed. One must appropriately choose the algorithms such that no high overhead is caused; at the same time, it is necessary to keep the time complexity minimal for the chosen data set. It has been observed that handling space complexity is not a challenge with memory becoming cheaper, whereas optimizing time complexity is significant.

There are numerous ways in which algorithms can be designed to achieve sorted order for given input data elements. The standard way of studying a technique and analyzing its efficiency is understanding the number of comparisons and time taken among the data items to decide their order. This must be followed by data exchanges, if necessary, to place the elements in the right position. This exchange is a costly operation, and the total number of exchanges is also important for evaluating the overall efficiency of the algorithm.

Hence, the above discussed factors must be taken into consideration in writing easy-to-use code and implement and debug it to attract the programmer population. The proposed sorting approach considers various parameters: data comparisons and exchanges, data movements, number of insertions, and time complexity to show that the proposed technique is efficient compared to existing sorting techniques. Such comparisons have been made for data sets of size 10, 100, 1,000 and 10,000.

The rest of this paper is organized as follows. Section 2 provides a glimpse of some related research work that has been done in the past. Section 3 briefly discusses the standard algorithms that form the basis of the idea presented in this paper. It further elaborates on the implementation of the proposed OHTO sorting technique with illustrative diagrams and the pseudocode. A comparison of the three algorithms, Insertion Sort, Bubble Sort and OHTO, is presented towards the end of Section 4 with the help of tables and charts. Section 5 is reserved for the conclusion.

## 2 Literature Review

Work around sorting algorithms has been going on since as early as the 1950s. A chronological inception of an abundance of standard sorting algorithms has taken place over the years, each of which is unique in its own ways. Based on the features, these algorithms are implemented in different suitable applications. Despite the existence of these algorithms, sorting larger arrays or databases optimally or quickly is still an open research problem.

Sorting algorithms are very much required in the software world for various types of analysis purposes. After referring to various papers the pros and cons of different sorting algorithms when performed individually can be known. By modifying the existing sorting algorithm or when two or more algorithms are combined to form a new algorithm, they perform better and are more efficiently than the individual existing algorithms.

The performance characteristics of different sorting techniques and types, their pros and cons, along with a comparative analysis have been proposed by Purvi Prajapati [1]. It was concluded that sorting is a specific problem as the efficiency of an algorithm depends on various parameters or factors. Considering time complexity as a factor, six sorting algorithms (Bubble Sort, Insertion Sort, Selection Sort, Counting Sort, Radix Sort and Bucket Sort) were compared by Sandeep Kaur Gill [2]. The range and the distribution of values were considered to see which algorithms would perform well in a given scenario. The algorithms were categorized as comparison and non-comparison sorting algorithms. It was concluded that the former must be preferred over the latter when the size of the dataset is small, and the latter must be preferred otherwise. Based on the pattern of the data fed into an input array – a random array, an ascending/descending order array, or an array containing the same elements – the best algorithm selection technique was implemented by Shubham Rustagi [3]. The algorithms considered were Bubble Sort, Insertion Sort, Quick Sort, Merge Sort, Selection Sort, Heap Sort.

The optimized selection sort algorithm (OSSA) developed by Sultanullah Jadoon [4] was proved to be more efficient and faster than Selection Sort and Insertion Sort. OSSA (O (n^2)) is a stable, in-placed, and easily understandable algorithm. The SA sorting algorithm (O(n^2)) for large amounts of sorted and/or unsorted data was proposed by Mohammad Shabaz [5]. This algorithm was proved to perform better than individual algorithms such as Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Heap Sort, Shell Sort, etc. in terms of memory required for sorting and execution time. The Selection Sort and Bubble Sort algorithms were improved by Jehad Alnihoud [6], under the name of Enhanced Selection Sort (ESS) and Enhanced Bubble Sort (EBS), respectively. ESS made the existing Selection Sort algorithm faster and more stable by performing fewer sort operations, although its time complexity was (O(n^2)). To sort n elements, EBS performed O(n log n) operations when compared to the Bubble Sort algorithm (O(n^2)). Input Sort was proposed by Anshu Mishra [7], typically for cases when elements of a stream from a file are to be sorted. This algorithm has a time complexity of O(n log n).

Considering efficiency as a major factor for sorting, a hybrid sorting algorithm – an enhanced version of Insertion Sort algorithm – was proposed by Tarundeep

Singh Sodhi [8]. It has varying complexity and is characterized by being efficient for huge lists, requiring a reduced number of comparisons. The Shell Sort algorithm was developed by Pooja K. Chhatwani [9] to improve the original Insertion Sort algorithm to efficiently allow the comparison and exchange of elements that are far apart. A bidirectional Insertion Sort was also introduced, as an optimization and enhancement of Insertion Sort. This worked efficiently for both big and small lists. An optimized Bubble Sort algorithm was proposed by Ramesh M Patelia [10], which reduced the number of pass iterations by half when compared to the original algorithm. The V-Re-Fr (VRF) Sorting algorithm, which is based on Bubble Sort algorithm, was proposed by Sunny Sharma [11] to improve functionality and reduce algorithm complexity. A left-right sort algorithm was proposed by Shashank Singh [12] and compared with three other algorithms – namely Insertion, Bubble and Selection Sort – based on CPU time, number of swaps, and number of comparisons. The proposed algorithm took fewer comparisons than all three original algorithms, whereas the number of swaps was found to be always the same as for Bubble Sort and Insertion Sort, for all length of input values.

Many ongoing research and studies take into account aspects of their own to come up with new ideas that could be applied in sorting arrays or databases [13]. This can be optimization of existing algorithms, ideas like inculcation of dynamic memory allocation, or the usage of unique data structures [14,15]. Hence, based on detailed survey on related work on sorting techniques, a new optimized sorting technique named Optimized Hybrid Two-in-One Novel Sorting Technique (OHTO) is proposed here. The proposed OHTO sorting technique is optimized in terms of data comparison, data movement, number of exchanges, number of insertions, and time complexity evaluation. The proposed approach is explained in the following section.

## 3 Proposed Optimized Hybrid Two-in-One Novel Sorting Technique (OHTO) Technique

### 3.1 Overview of Composed Algorithms

Insertion Sort is a simple sorting algorithm that is relatively efficient for small lists and mostly sorted lists and is often used as part of more sophisticated algorithms. It works by taking elements from a list one by one and inserting them in their correct position into a sorted list. In arrays, the new list and the remaining elements can share the array's space, but insertion is computationally expensive, requiring shifting over all the consecutive elements by one. This algorithm has a quadratic running time O(n^2) in the worst case. In average cases, it is also quadratic, which makes Insertion Sort impractical for sorting large arrays.

Bubble Sort is another simple sorting algorithm. The algorithm starts at the beginning of the data set, compares the first two elements, and swaps elements if the first is greater than the second. It continues doing this for each pair of adjacent elements to the end of the data set. Again, apply Bubble sort technique from first location. In each iteration, one element occupies its correct position and finally gets sorted array. This algorithm's average time and worst-case performance is O(n^2), so it is rarely used to sort large, unordered data sets. Bubble Sort can be used to sort a small number of data items, where its asymptotic inefficiency is not a large penalty.

## 3.2 Optimized Hybrid Two-in-One Novel Sorting Technique (OHTO) Technique

In the proposed technique, the array or database is first split into two halves. Bubble Sort is applied on these halves. Insertion Sort is performed as the next step, wherein elements from the second half are the key elements that are inserted into the first half following the rules of Insertion Sort. What makes this approach more efficient is that unlike in normal Insertion Sort, two elements are inserted in every iteration or pass. This is useful as the halves are already sorted and hence the presented technique can take advantage of the fact that the pair of keys that are to be inserted will be sorted in a particular way, just as the first half of the array or database will be.

### 3.2.1 Illustrative Example Demonstrating OHTO Sorting Technique Towards Various Cases

Consider the following input chunk of data elements:

| 30 | 40 | 10 | 50 | 100 | 60 | 5 | 20 | 45 | 4 | 70 |
|---|---|---|---|---|---|---|---|---|---|---|

Divide the array into two halves:

| 30 | 40 | 10 | 50 | 100 | 60 | 5 | 20 | 45 | 4 | 70 |
|---|---|---|---|---|---|---|---|---|---|---|

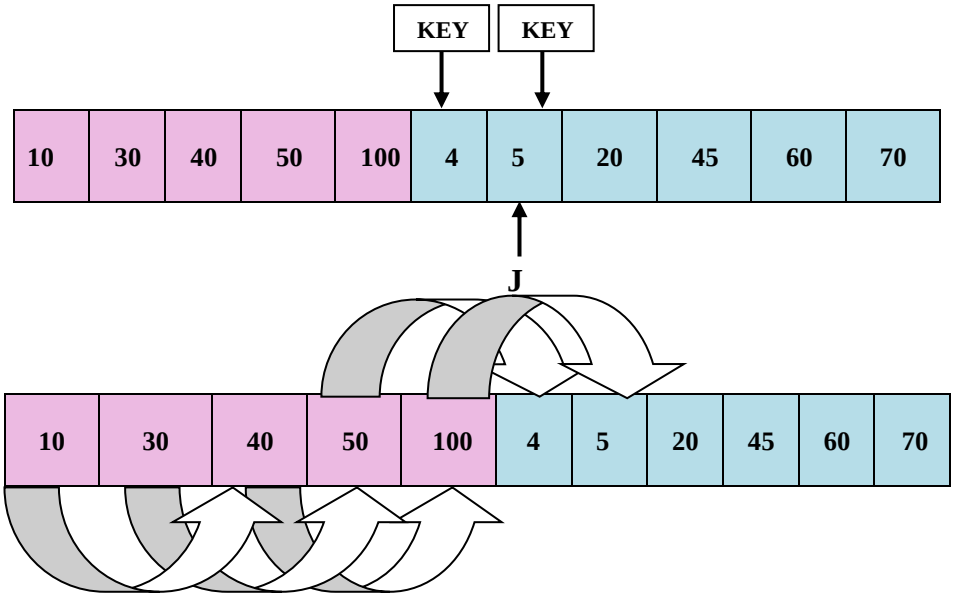
Apply Bubble Sort to the two halves:

| 10 | 30 | 40 | 50 | 100 | 4 | 5 | 20 | 45 | 60 | 70 |
|---|---|---|---|---|---|---|---|---|---|---|

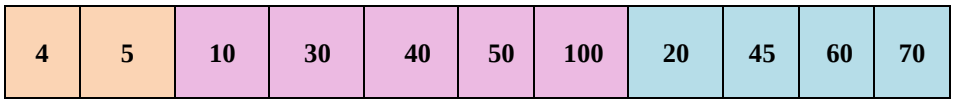
Apply Insertion Sort on the elements of the second halve to be inserted into the first halve.

**CASE 1: Inserting both keys before a specific element in the first half of the sorted halve**

Insert both keys before a specific element in the first half of the sorted halve by moving each element by two positions towards the right:

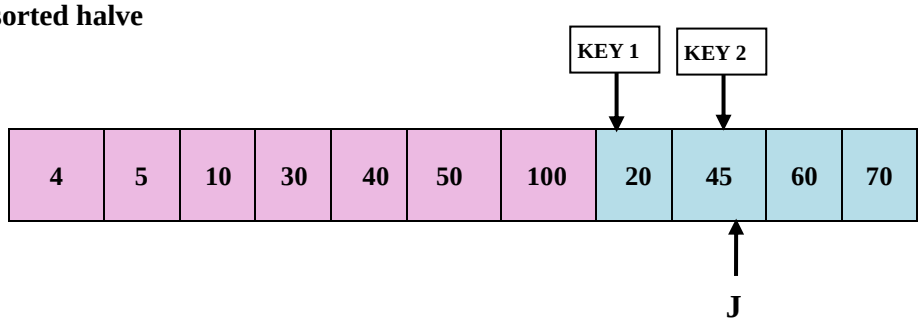



After insertion of ***KEY 1*** and ***KEY 2***:

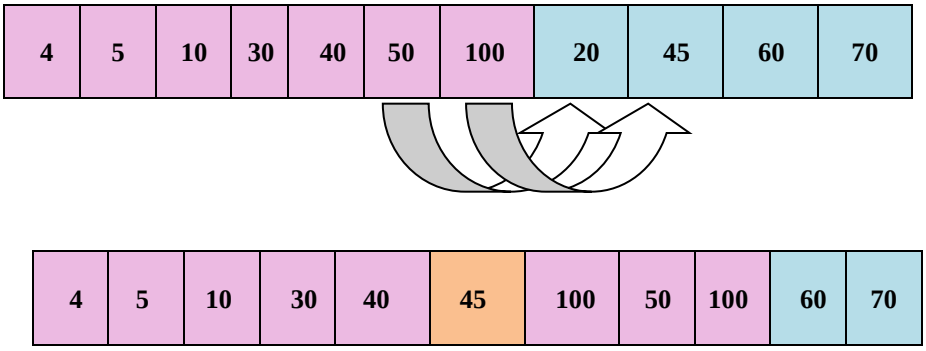



Increment J by 2 targeting towards insertion of the next two elements in the sorted second halve into first sorted halve.

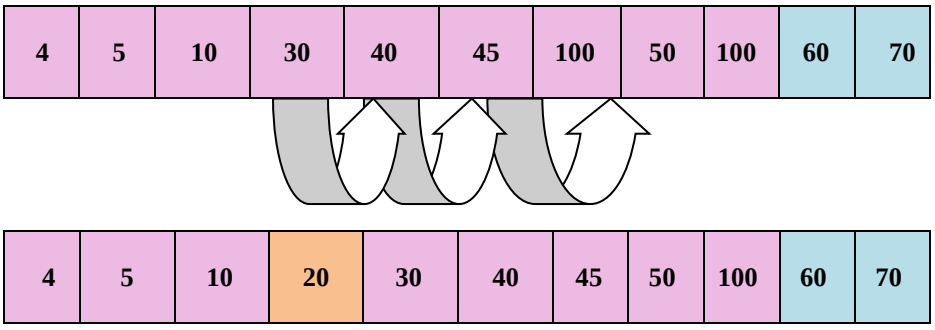
**CASE 2: Inserting both keys at different positions in the first half of the sorted halve**



Compare *KEY 2* value 45 with the elements in the first sorted halve and move data elements greater than *KEY 2* two positions towards the right. Insert *KEY 2* value 45 in the correct position in the first sorted halve:

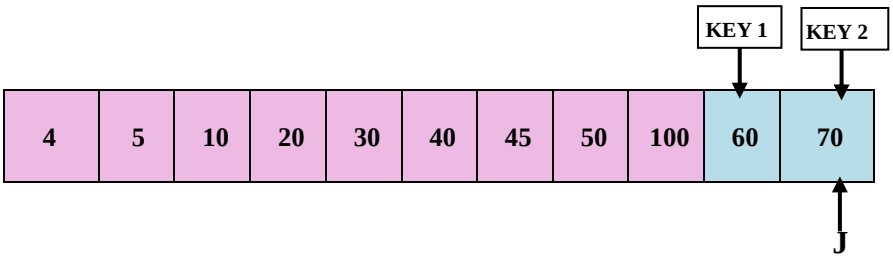



Compare *KEY 1* value 20 with the elements in the first sorted halve and move data elements greater than *KEY 1* one position towards the right. Insert *KEY 1* value 20 in the correct position in the first sorted halve:



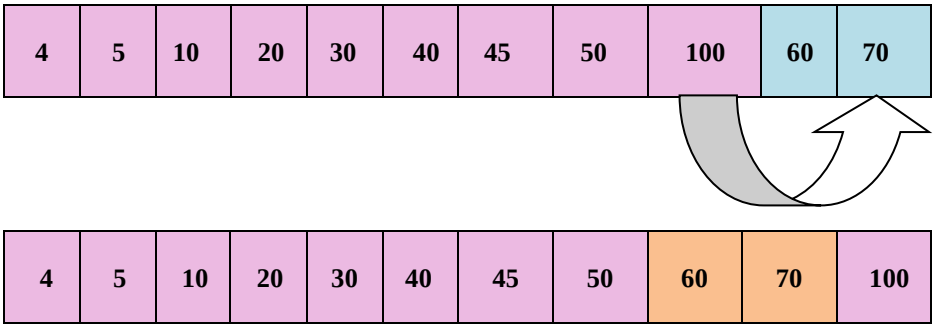

Increment J by 2 targeting towards insertion of the next two elements in the sorted second halve into the first sorted halve.

**CASE 3: Both keys in the second sorted halve to be inserted after a specific element into the first sorted halve**




Compare *KEY 2* value 70 with the elements in the first sorted part and move data elements greater than *KEY 2* two positions towards the right. Both keys 60 and

70 are greater than 50 but less than 100. Insert both keys after element 50 in the first sorted halve.




Final sorted array:

| 4 | 5 | 10 | 20 | 30 | 40 | 45 | 50 | 60 | 70 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|

### 3.2.2 OHTO Algorithm

The steps of the proposed OHTO algorithm design are as follows:

***OHTO (A[ ] , n)***
***Input:*** *Unsorted Array: A[1..n], Number of Elements: n*
***Output:*** *Sorted Array A[1..n] with*
- *Reduced number of Comparisons*
- *Reduced number of Data Exchanges*
- *Reduced number of Data Movements*
- *Reduced number of Data Insertions*

```
1.  Divide an input array A into two halves: (A[1...n/2]), (A[n/2+1...n])
2.  Bubble_Sort (A[1...n/2],n/2)
3.  Bubble_Sort (A[n/2+1...n],n/2)
4.  for j ← (n/2+1) to n        // Apply Insertion Sort technique on second halve elements to
                                   be inserted into the first halve
5.      A[j]  ← key1
6.      A[j+1]  ←key2
7.      i ← j-1
8.      if(key1 < A[i] && key2 < A[i]))
9.          while ( i<=1 && (key1 < A[i] && key2 < A[i]))  // Case 1: Insert both key
                                                              elements before a specific element in the sorted array.
10.             A[i]→A[i+2]   // Move the i^th element two positions to the right.
11.             i←i-1
12.             if(key1 < A[i] && key2 < A[i])
13.                 A[i]←key1
14.                 A[i+1] ← key2
15.                 j←j+2     // Jump j by two positions to the right.
```

```
16.                     endif
17.                 break
18.                 end while
19.             endif
20.             else
21.                 if(key2 < A[i])   // Case 2: Insert both key elements at a specific element in the
                                          sorted array.
22.                     while(i<=1 && (key2 < A[i]))
23.                         A[i+2]←key2   // Move the i^th element two positions to the right.
24.                         i←i-1
25.                     end while
26.                     A[i+1]←key2
27.                     while(i<=1 && (key1 < A[i]))
28.                         A[i+1]←key1
29.                         i←i-1
30.                     end while
31.                     A[i+1]←key1
32.                     j←j+2
33.                 endif
34.                 else
35.                     while(i<=1 && (key2 < A[i]) )  // Case 3: Insert both key elements after
                                                          a specific element the sorted array.
36.                         A[i+2]←A[i]              // Move the i^th element two positions to the right.
37.                         i←i-1
38.                         if(key2 < A[i]) && key1>A[i-1])
39.                             A[i+1]←key2
40.                             A[i+2]←key1
41.                             j←j+2
42.                             break
43.                         end if
44.                 end while
45.  end for
46.  end
```

## 4 OHTO Sorting Technique: Results and Discussion

The proposed sorting approach considers various parameters: data comparisons, number of exchanges, number of insertions, and time complexity to show proposed technique's efficiency compared to existing sorting techniques. Such comparisons have been made for data sets of size 10, 100, 1,000, and >10,000, where the order of the dataset was random.

The below tables and charts show the comparison of the three algorithms based on four parameters.

1. *Number of Comparisons*

The number of comparisons carried out by the proposed OHTO sorting technique is shown in Table 1.

**Table 1** Comparison on number of comparisons.

| Sorting Technique / No. of Data Elements | Bubble Sort | Insertion Sort | OHTO Sort |
|---|---|---|---|
| 10 | 45 | 25 | 24 |
| 100 | 4950 | 2572 | 2500 |
| 1,000 | 4,99,500 | 2,50,131 | 1,25,250 |
| > 10,000 | 49,995,000 | 25,150,175 | 25,000,000 |

Figure 1 depicts the comparison of the three sorting techniques on the basis of the total number of data element comparisons carried out throughout the sorting process. It can be seen that the Bubble Sort algorithm always makes the highest number of comparisons. In most cases, the proposed algorithm makes nearly the same number of comparisons as Insertion Sort, whereas this number is almost doubled in the case of Bubble Sort. Thus, the proposed technique is over 50% more efficient than Bubble Sort in this aspect and slightly better than Insertion Sort as well.

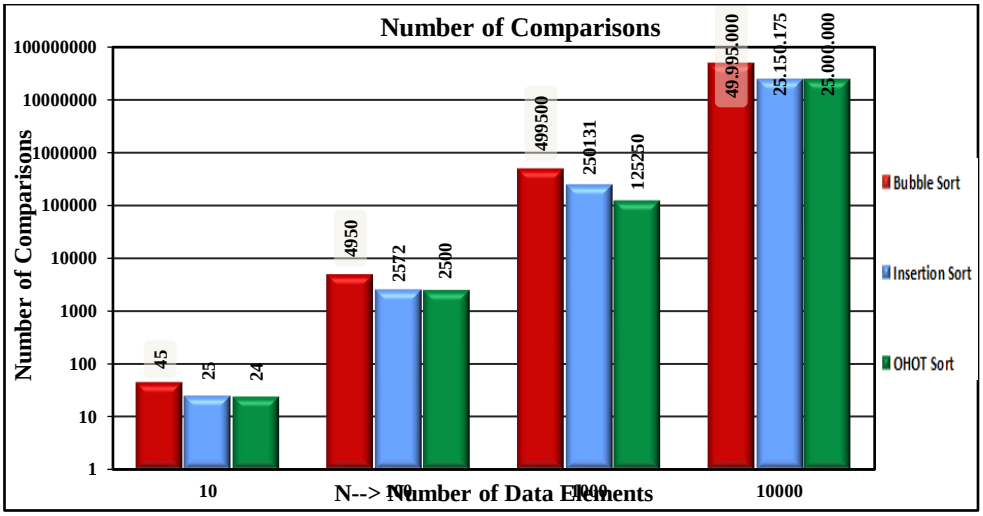



**Figure 1** Number of comparisons of data elements.

2. *Number of Movements (considering 1 swap ⇔ 2 movements)*

The number of data movements carried out by the proposed OHTO sorting technique is shown in Table 2.

**Table 2** Comparison on number of data movements.

| Sorting Technique / No. of Data Elements | Bubble Sort | Insertion Sort | OHTO Sort |
|---|---|---|---|
| 10 | 90 | 25 | 24 |
| 100 | 9900 | 2572 | 2500 |
| 1,000 | 9,99,000 | 2,50,131 | 1,25,250 |
| > 10,000 | 99,990,000 | 25,150,175 | 25,000,000 |

Figure 2 shows the number of data movements that occur when the different algorithms are implemented. Data movement is a rather costly operation, and the aim should always be to keep it minimal. In this regard, Bubble Sort stands at the top, as the algorithm is designed in such a way that one swap – which amounts to two data movements – occurs at every step. Hence, the number of movements is very high. This is not the case with Insertion Sort. The number of movements is relatively much smaller, the same as for the proposed technique. The average difference in this number is nearly 74% for Bubble Sort and the proposed sort techniques. The similarity between Insertion Sort and the OHTO sorting technique in this respect is because of the same underlying principle/idea of insertion of elements. Still, the proposed technique proved to achieve a smaller number of movements in some cases, as shown in the chart graph.

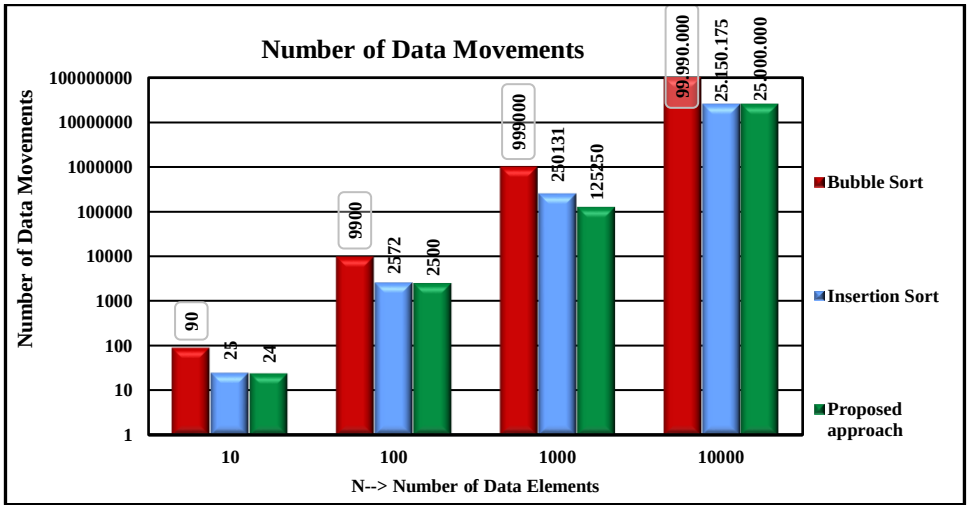



**Figure 2** Number of movements of data elements.

3. *Number of Data Insertions*

The number of insertions carried out by the proposed OHTO sorting technique is shown in Table 3.

**Table 3** Comparison on number of data insertions.

| Sorting Technique / No. of Data Elements | Bubble Sort | Insertion Sort | OHTO Sort |
|---|---|---|---|
| 10 | NA | 9 | 4 |
| 100 | NA | 99 | 50 |
| 1,000 | NA | 999 | 500 |
| > 10,000 | NA | 9999 | 5000 |

Figure 3 demonstrates the relationship between the number of insertions and the size of the input data set for Insertion Sort and the proposed OHTO sort technique. With an increase in input size, the number of insertions increases linearly in both cases. However, the proposed technique implements sorting by performing approximately 49% less insertions when compared to Insertion Sort. This is also a parameter that contributes to efficiency, so the proposed algorithm successfully reduces the number considerably. This is owing to the fact that the two halves of the input data set are bubble sorted in the beginning. This affects the later stages of the algorithm significantly.

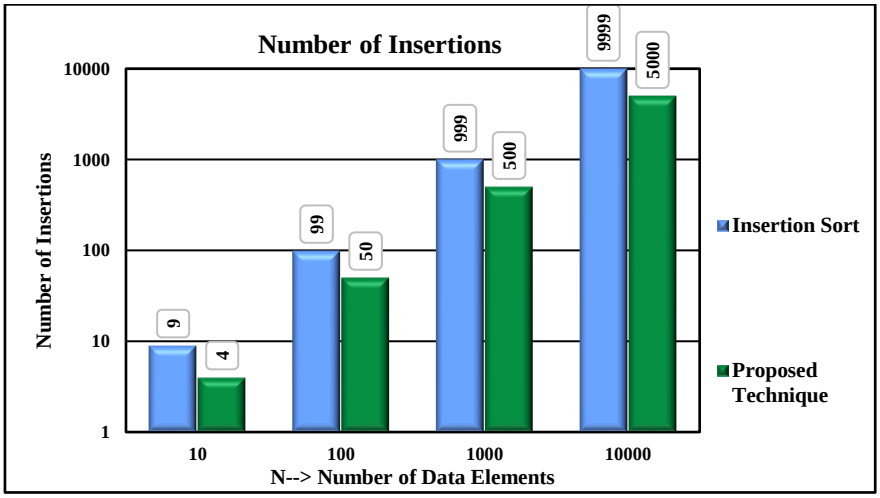



**Figure 3** Number of insertions of data elements.

4. *Execution Time (in Milliseconds*)

The time complexity comparison amongst the three sorting algorithms is shown in Table 4, while the actual time taken by the proposed OHTO sorting technique to sort n data elements is shown in Table 5.

We perform Bubble Sort once on each half of the dataset, after which Insertion Sort is performed by choosing the pivot element from the second sorted data element. Therefore, the time complexity = 2*(Time-Taken for Bubble Sort on one half of the set) + Insertion Sort using the sorted dataset.

1. Best case: $2^* \Omega(n/2) + \Omega(n) = \Omega(n)$
2. Average case: $2^* \theta((n/2)^2) + \theta(n) = \theta(n)$
3. Worst case: $2^*O((n/2)^2) + O(n) = O(n)$

The number of comparisons made, in contrast to the procedures of Bubble Sort and Insertion Sort, is smaller for the OHTO algorithm. Insertion Sort is usually used for smaller data sets, whereas OHTO is applicable for large and small data sets. Hence, in such cases the OHTO algorithm would be preferable.

**Table 4** Comparison of time complexity.

| Sorting Technique / Time Complexity | Bubble Sort | Insertion Sort | OHTO Sort |
|---|---|---|---|
| Best case (Ω) | n | n^2 | n |
| Average case (θ) | n^2 | n^2 | n |
| Worst Case (O) | n^2 | n^2 | n |

**Table 5** Comparison of time (milliseconds).

| Sorting Technique / No. of Data Elements | Bubble Sort | Insertion Sort | OHTO Sort |
|---|---|---|---|
| 100 | 4662 | 2663 | 1666 |
| 1,000 | 343,789 | 366,322 | 215,824 |
| > 10,000 | 30,825 | 60,656 | 9,033 |

Figure 4 illustrates the effectiveness of the proposed algorithm in obtaining quicker results by comparing the time of execution of the three algorithms. For data sets of all sizes, it can be seen that the proposed OHTO sorting technique executes faster, thus yielding results in less time when compared to the other two techniques. Another key observation that can be made is that with an increase in input size, i.e., as we proceed towards the right side in the chart above, it can be noted that execution time increases for Insertion Sort but remains the shortest for proposed OHTO sorting technique. In total, the new proposed OHTO sorting technique achieves sorting faster, thus, serving its main purpose.

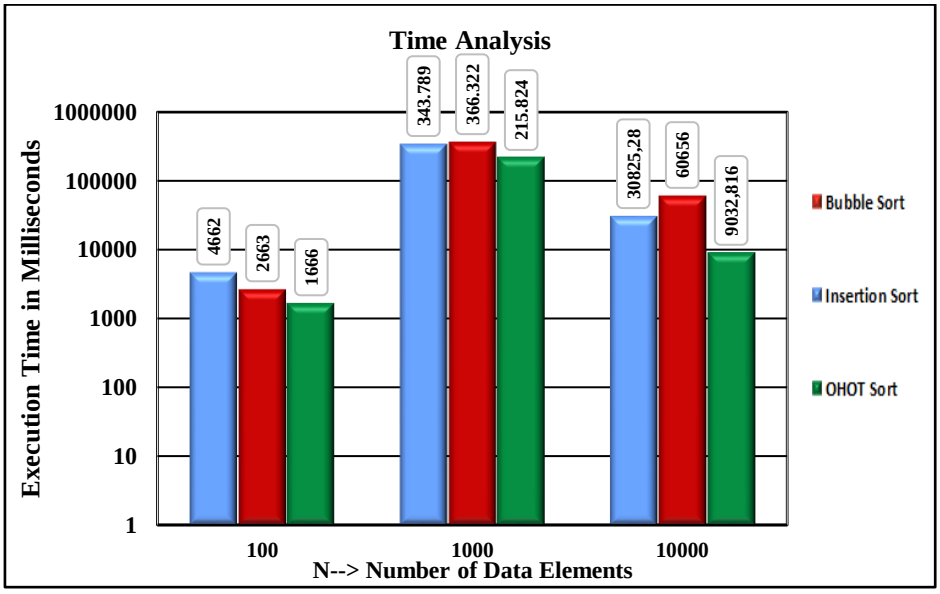



**Figure 4** Execution time of algorithms.

## 5 Conclusion

This paper presented a novel sort algorithm that is a hybrid of the Bubble Sort and Insertion Sort algorithms to sort a given data set. The proposed OHTO sorting technique uses the procedure of the Bubble Sort technique followed by the insertion of the elements of the second sorted half of the data set into the first half by following the Insertion Sort protocol. If the given array is completely unsorted, the OHTO algorithm may not perform much better than other sorting algorithms with more data movements. Hence, this is a disadvantage when the data set is completely unsorted and after performing Bubble Sort on each half of the dataset, the consecutive elements that are supposed to be in the final sorted array are not present in the same half of the dataset. However, the proposed OHTO sorting technique has the advantages of requiring fewer comparisons, fewer data insertions, and less time to sort any given large data set.

## References

[1] Prajapati, P., Bhatt, N. & Bhatt, N., *Performance Comparison of Different Sorting Algorithms*, International Journal of Latest Technology in Engineering, Management & Applied Science, **6**(6), pp. 39-41, ISSN 2278-2540, 2017.

[2] Gill S.K., Singh, V.P., Sharma, P. & Kumar, D., *A Comparative Study of Various Sorting Algorithms*, International Journal of Advanced Studies of Scientific Research, **4**(1), pp. 367-372, 2019.

[3] Rustagi, S. & Yadav, T., *A Comparative Study of Various Sorting Algorithms Optimized Sorting Process*, International Journal of Technology & Management, **4**(1), pp. 52-56, ISSN: 2454-8421, 2018.

[4] Jadoon, S., Solehria, S.F. & Qayum, M., *Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm: A Comparative Study*, International Journal of Electrical and Computer Sciences, **11**(2), pp. 18-23, 2011.

[5] Shabaz, M. & Kumar, A., *SA Sorting: A Novel Sorting Technique for Large-Scale Data*, Research Article ID 3027578, pp. 1-7, 2019. DOI: https://doi.org/10.115 5/2019/3027578.

[6] Alnihoud, J. & Mansi, R., *An Enhancement of Major Sorting Algorithms*, The International Arab Journal of Information Technology, **7**(1), pp. 55-62, 2010.

[7] Mishra, A. & Goyal, G., *An Optimized Input Sorting Algorithm*, Global Journal of Computer Science and Technology: Network, Web & Security, **16**(1), ISSN: 0975-4172, 2016.

[8] Sodhi, T.S., Kaur, S. & Kaur, S., *Enhanced Insertion Sort Algorithm*, International Journal of Computer Applications, **64**(21), pp. 35-39, 2013.

[9] Chhatwani, P.K., *Insertion Sort with its Enhancement*, International Journal of Computer Science and Mobile Computing, **3**(3), pp. 801-806, 2014.

[10] Patelia, R.M., Vyas, S.D. & Vyas, P.S., *An Analysis and Design of Optimized Bubble Sort Algorithm*, International Journal of Research in Information Technology, **3**(1), pp. 65-68, 2015.

[11] Sharma, S., Kumar, V., Singh, P. & Singh, A., *A Novel Algorithm for Optimized Sorting*, International Journal of Scientific Research in Science, Engineering and Technology, **2**(2), pp. 1072-1076, ISSN: 2395-1990, 2016.

[12] Singh, S., Singh, R.S. & Mandoria, H.L., *Left-Right Sort: A Novel Sorting Algorithm and Comparison with Insertion Sort, Bubble Sort and Selection Sort*, International Journal of Engineering & Scientific Research, **3**(10), pp. 47-59, ISSN: 2347-6532, 2015.

[13] Yogeesha, C.B., Pujeri, R.V. & Veena, R.S., *Randomized Algorithms: On the Improvement of Searching Techniques Using Probabilistic Linear Linked Skip Lists*, International Conference on Advances in Computing - ICAdC http://link.springer.com/chapter/ 10.1007/978-81-322-0740-519, pp. 147-153.

[14] Rajeshwari B S, Singh, N., Ibaduddin, M.S., Singhal, I. & Sree Vidya B S., *A Novel Efficient Selection Sort Technique for the Larger Dataset*, GIS Science Journal, **8**(8), pp. 257-266, 2021.

[15] Pujeri, R.V. & Yogeesha, CB., *A Relative Study of Genetic Algorithms and Hopfield Neural Network to Optimize Shortest Path Routing Algorithms for Finding Drilling Holes on the Printed Circuit Board-PCB*, Journal of Adv Research in Dynamical and Control Systems, **15**(Special Issue), pp. 801-808, Dec 2017.