ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Representation of Different Sorting Algorithms Using Sorting Visualizer

Ashish Gautam[1], Deepak Thakur[2]

[1]Assistant Professor, Computer Science & Engineering, Creanovation Technologies Private Limited, NextGen Academy, Mohali, Punjab, India

[2]Assistant Professor, Computer Science & Engineering, Creanovation Technologies Private Limited, NextGen Academy, Mohali, Punjab, India

DOI: https://doi.org/10.51583/IJLTEMAS.2026.150500040

Received: 03 May 2026; Accepted: 08 May 2026; Published: 26 May 2026

ABSTRACT

The most effective way to sort elements (data) significantly impacts how quickly a computer can complete a task. The sorting algorithm is one of computer science's most important areas of study. It is the process of arranging unorganized elements in an organized manner. The main goal is to make records easier to search, sort, insert, and delete. Through the description of the existing five sorting algorithms: Bubble, Selection, Insertion, Merge, and Quick Sort, the Time and Space Complexity are determined. Time complexity varies as Best Case(O), Average Case(Θ), and Worst Case(Ω). Here in sorting, the Worst Case complexity is $O(n^2)$, and the Best Case complexity is $O(n \log n)$, where "n" represents the number of elements(data) in the array. This project consists of a UI-based web application that can visualize the sorting process using various colors and denote the status of the elements of the array.

INTRODUCTION

In today's era the data is growing in abundant amounts and searching through that huge data is time-consuming. Sorting and Searching is an important operations in computer programming. Among them, one of the most important subroutines in many algorithms is sorting the data. Therefore sorting techniques are used to sort the data in an ordered manner.

Different sorting algorithms differ in terms of efficiency, memory usage, time complexity, and other features. For decades work has been done on sorting algorithms to make it more efficient and less time-consuming. But every sorting algorithm is not suitable for different types of data sets. Some sorting methods might work excellently at some data sets but if we use the same sorting method for another data set it might work worse. So accordingly the algorithm must be selected to sort the data, keeping in mind that it is efficient and less time-consuming.

Sorting algorithms are basically devised to arrange the data items in a number of ways such as it can be sorted from higher to lower order or it can be sorted from lower to higher order as needed. The time complexity of many sorting algorithms is linear or quadratic. Many of them have quadratic time complexity as their worst-case scenario, whereas linear time complexity is their best-case scenario.

For each data set majorly two operations are performed within the elements: Comparison and Swapping. The efficiency of any sorting algorithm is compared in terms of time complexity which depends on the number of comparisons performed on the particular data set. Therefore the time complexities are referred to as best, worst, and average cases.

Sorting of data can be categorized into two ways: recursive approach and iterative approach.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

A. Recursive Approach

The inductive steps, which can be easily and quickly sketched out in terms of the Divide and Conquer approach, provide the foundation of the recursive approaches. This approach has been followed by many algorithms such as Quick Sort, Merge Sort, Heap Sort, etc. This approach is mainly used when the size of the code is small, here the time complexity of the algorithm doesn't matter.

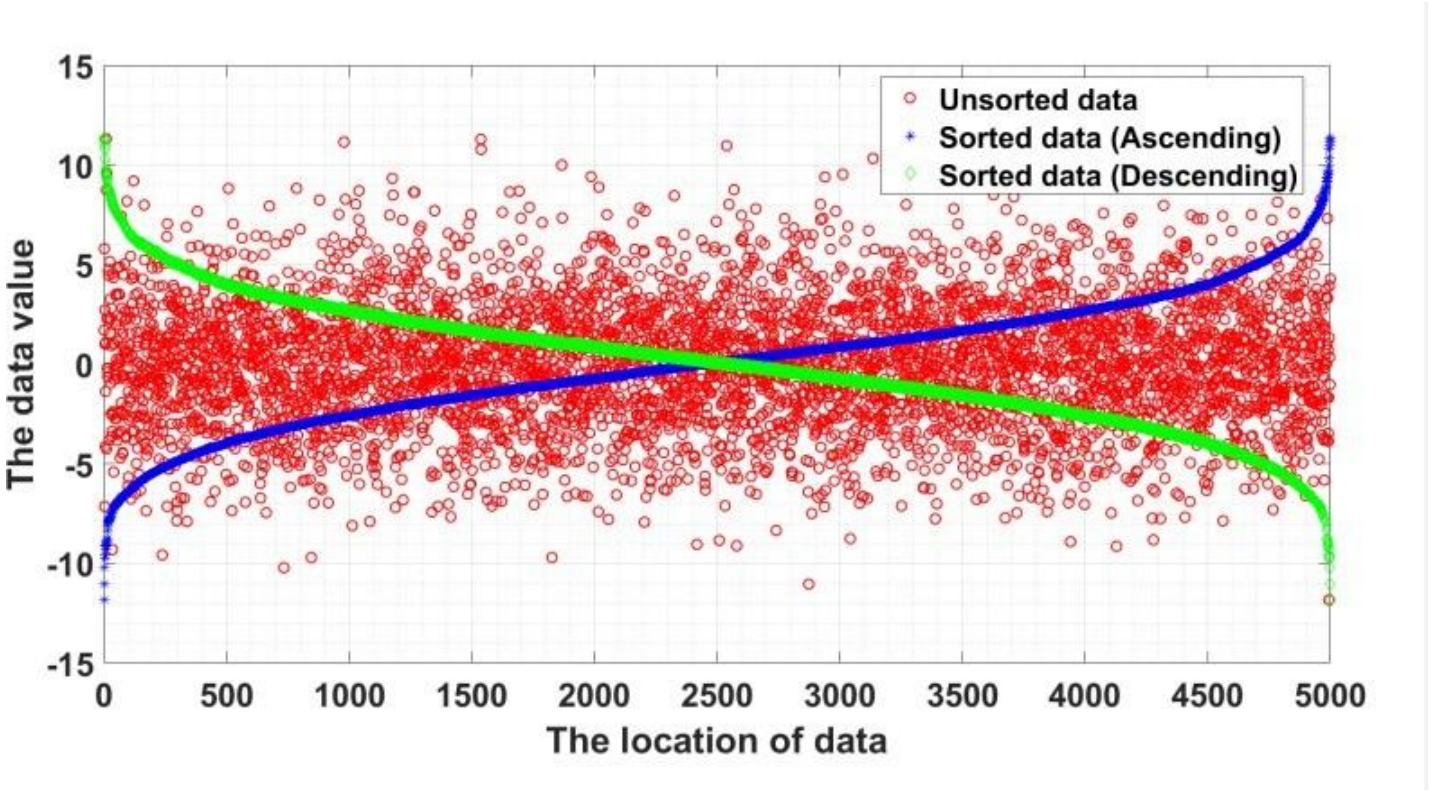


Figure 1. An example of sorted and unsorted non-integer data for 500 random numbers.

Merge Sort: Merge Sort is based on the Divide and Conquer concept. This algorithm divides the array into two equal parts, which are then combined in a sorted way.

Quick Sort: Quick Sort considers the pivot element and divides or splits the array based on that pivot element. By shifting all the components less than the pivot to its left and the greater ones to its right.

Heap Sort: The fundamental concept behind this approach is a binary tree. It generates either a minimum or maximum heap. The foundation of heap sorting is the idea of taking elements out of the heap list and adding them to the sorted list.

B. Iterative Approach

In the worst situation, the iterative technique scans the complete set of elements by making continuous and repeated passes over the set of items that are being sorted. Essentially, the goal of this approach is to identify successive approximations or iterations in order to arrive at a solution.

Bubble Sort: The most elementary sorting method is Bubble Sort, which repeatedly swaps nearby elements if they are in the wrong order. Large data sets should not be used with this approach due to its high average and worst-case time complexity.

Insertion Sort: The simple sorting algorithm known as insertion sort functions similarly to how you would arrange playing cards in your hands. This algorithm's basic idea incorporates two arrays: an unsorted array and a sorted array. The first element of the array is in the sorted one, while the remaining items are in the unsorted

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

one. Until the array is sorted, the first element existing in the unsorted array is compared to the elements of the sorted array.

Selection Sort: The working of selection sort is very simple. The selection sort method sorts an array by repeatedly selecting the first element from the unsorted portion that is the least in value while still taking into account ascending order.

Table 1. Comparison between Six Sort Algorithm

Algorithm	**Best**	**Average**	**Worst**	**Stable**	**Extra space**
BUBBLESORT	$O(n)$	$O(n^2)$	$O(n^2)$	yes	$O(1)$
SELECTIONSORT	$O(n^2)$	$O(n^2)$	$O(n^2)$	no	$O(1)$
INSERTIONSORT	$O(n)$	$O(n^2)$	$O(n^2)$	yes	$O(1)$
QUICKSORT	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	no	$O(\log n)$
MERGESORT	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	yes	$O(n)$
HEAPSORT	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	no	$O(1)$

LITERATURE REVIEW

Super sort sorting algorithm by Yash Gugale from the Department of Computer Engineering at pune. The sorting technique presented in this paper makes use of the natural sequence of sorted elements in an array of random numbers so as to reduce the number of steps needed to sort. The complexity of the proposed algorithm is $O(n\log n)$ where n represents the number of elements in the input array.

RBS: A new comparative and better solution of sorting algorithm for array by Md. Hasan Imam Bijoy at Dhaka, Bangladesh. In this paper, the algorithm is designed and developed that can be executed on random data, ordering data, and some specific data.

When compared to bubble sort, RBS compares the left and right values, increases the left index, and then decreases the right index after one step. This method cuts the bubble sorting algorithm's execution time in half.

MinFinder: A new approach in sorting algorithm by Md. Shohel Rana in China. The proposed MinFinder sorting algorithm mainly finds the element whose value is the smallest from the list or array. This is done by shifting elements from one position to the right from the first position to the position of the smallest element found. It keeps elements with equal keys in the same relative order in the output as they appeared in the input.

SRCS: A new proposed counting sort algorithm based on the square root method by Hridoy Roy at Jashore, Bangladesh. The proposed algorithm is the extended version of the existing counting sort algorithm. In the proposed algorithm a square root method is used to count the number of occurrences. Then the cumulative sum method is used to build some blocks. After that, the general counting sort is used to rearrange each block. Fuse Sort Algorithm: A proposal of divide & conquer based sorting approach with O(n loglog n) Time and Linear Space Complexity by Yashwant Singh Patel at Bhubaneshwar, India. This proposed algorithm is based on a divide-and-conquer approach.

First, it divides the n number of elements into n number of parts. Each part is having √n size. It keeps on continuing and divides the problem into smaller problems until each subpart minimum size is 2. After dividing the problem into √n halves having size √n, it starts to sort each sub-part with the help of a recursion method. At last, each sub-part is combined to form the solution to the original problem. An efficient sorting algorithm for binary data by Bhavani Yerram at Warangal, India. In this paper an iterative recursive algorithm with a divide-

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

and-conquer approach is proposed for the efficient sorting of data. To sort the data into ascending order bin sort uses binary numbers stored in an array and bit positions of 0 and 1 in relative order. In bin sort, we consider the bit position and sort the numbers into two separate lists based on 0 and 1. The first Most Significant Bit MSB of the numbers is considered and numbers are separated into two lists i.

A sorting algorithm based on ordered block insertions by Héctor Ferrada in Chile. The core idea of the algorithm is simple: we train a CDF model F over a small sample of keys A and then use the model to predict the position of each key in the sorted output.

OneByOne(OBO): a fast sorting algorithm by Ashjan Alotaibi at Riyadh, SA. The OBO sorting algorithm has two functions. In the beginning, the first function checks the array's first element to determine whether it is less than or equal to the second element. The algorithm then proceeds to the next iteration and starts from the second index. This is the idea behind the name OneByOne.

METHODOLOGY

Classical Bubble Sort: This is the simplest sorting algorithm that works by comparing and swapping the adjacent elements if not present in sorted order.

Algorithm 1: Bubble Sort

1. procedure Bubble Sort(arr, size)
2. for i=0 to n-i-1
3. for j=0 to n-i-2
4. if arrtj)>arrtj+1)
5. Svap arrtj) and arrtj+1)
6. flag = 1;
7. end if
8. end for
9. if(flag=0)
10. break
11. end for
12. end procedure

Stepwise sorting of Classical Bubble Sort:

Array:

5	**3**	**1**	**9**	**8**	**2**	**4**	7

Pass 1:

5	**3**	**1**	**9**	**8**	**2**	**4**	7
3	5	1	9	8	2	4	7
3	1	5	9	8	2	4	7
3	1	5	9	8	2	4	7
3	1	5	8	9	2	4	7
3	1	5	8	2	9	4	7
3	1	5	8	2	4	9	7
3	1	5	8	2	4	7	9

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Pass 2:

3	**1**	**5**	**8**	**2**	**4**	**7**	**9**
1	3	5	8	2	4	7	
1	3	5	8	2	4	7	
1	3	5	8	2	4	7	
1	3	5	2	8	4	7	
1	3	5	2	4	8	7	
1	3	5	2	4	7	8	

Pass 3:

1	**3**	**5**	**2**	**4**	**7**	**8**	**9**
1	3	5	2	4	7		
1	3	5	2	4	7		
1	3	2	5	4	7		
1	3	2	4	5	7		
1	3	2	4	5	7		

Pass 4:

1	**3**	**2**	**4**	**5**	**7**	**8**	**9**
1	3	2	4	5			
1	2	3	4	5			
1	2	3	4	5			
1	2	3	4	5			

Pass 5:

1	**2**	**3**	**4**	**5**	**7**	**8**	**9**
1	2	3	4	5			
1	2	3	4	5			
1	2	3	4	5			

Mostly bubble sort requires (n-1) passes to sort the array. Here in the above example, the array contains the 8 elements so it must require at most 7 passes but the given array is sorted in the 5th pass only. This is possible only when we use the flag bit in the bubble sort algorithm which tells if an array is sorted or not and if it requires further passes.

So if $(flag = 1)$ the array is not still sorted and requires the next pass and if $(flag = 0)$ means the array has been sorted in the current pass and no passes are required further. In this way, the number of passes is reduced in Bubble Sort.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Algorithm 2: Merge Function. Sort Arr[l,. ...r] in ascending order.

1. procedure HERGE_FUNCTI0N(A, l, n, r)
2. a = n - l + 1
3. b = r - n
4. let L and R array of lengths a and b respectively
5. copy Arrtl…n) into L and Arrtn+1….r) into R
6. i = j = 0
7. k = l
8. vhile i < a AND j < b do
9. if Lti) <= Rtj) then
10. Atk) = Lti)
11. i = i + 1
12. else
13. Arrtk) = Rtj)
14. j = j + 1
15. end if
16. k = k + 1
17. end vhile
18. if i < a then
19. append Lti….a) to Arr
20. if j < b then
21. append Rtj….b) to Arr
22. delete arrays L and R
23. end procedure

Proposed Bubble Sort:

The proposed algorithm is specifically designed for a large amount of data. We will take the data randomly. It will also use OpenMP. It consists of the following steps:

1. Initially, we get the array elements using the rand() function.
2. Now using the OpenMP function get_num_thread() we will get the number of threads your device has and store it in a variable.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

3. Divide the array into equal sub-arrays.
4. The number of subarrays will be equal to the number of cores in the device.
5. So for that, the array is divided by the number of threads(cores).
6. Now since we have got the equally divided sub-arrays, each sub-arrays will be sorted using bubble sort parallelly.
7. Here it will be parallelized under section construction, so until all the sections have not completed the task, the completed one will wait for the remaining to get completed.
8. Once the sub-arrays are sorted among themselves, we will merge them by creating another array of size equal to the number of elements in the original array.
9. The merging will be performed using the Merge Algorithm on the sorted subarrays.
10. Finally, we get the sorted array.

Algorithm 3: Proposed Bubble sort

Input: Array of items of size n

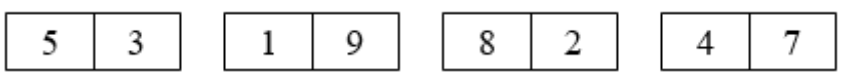
Output: Sorted array

1. arr = rand() % x
2. t = get_nun_threads()
3. subarray_size = sizeof(arrt)) / t
4. n = sizeof(arr) / sizeof(arrt0));
5. arr_s1t) = 0 to subarray_size - 1
6. arr_s2t) = subarray_size + 1 to subarray_size + subarray_size
7. #pragna onp parallel
8. section construct
9. Algorithn 1: Bubble sort //section part executes according to threads.
10. Algorithn 2: Herge Function
11. Arr is the sorted array
12. end

Process of sorting by proposed Bubble sort:

Array:

Here we assume the number of threads as 4, the above array, contains 8 elements, and further we will get 4 equally divided subarrays.

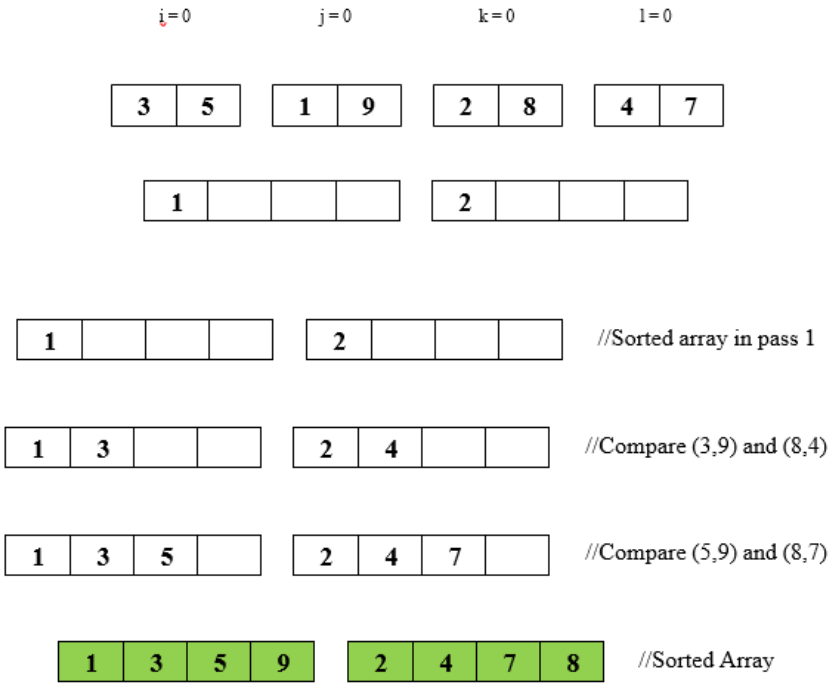


ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

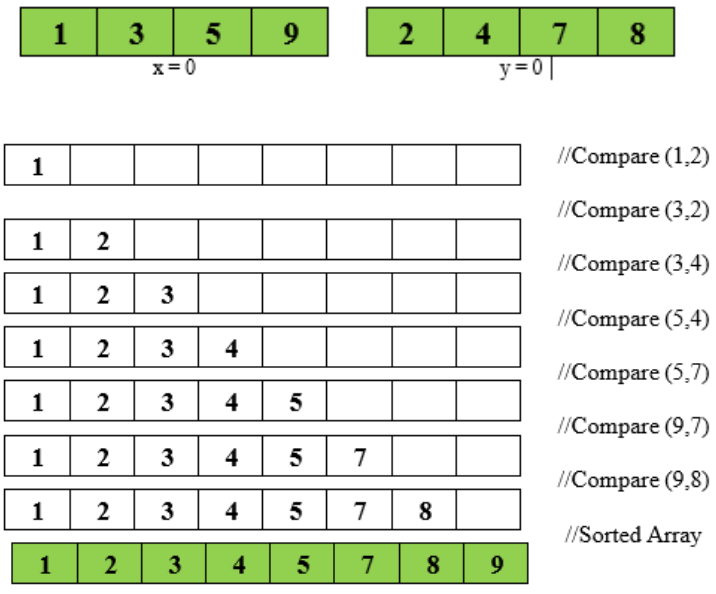
Now we will perform bubble sort on individual subarrays parallelly. In the first subarray we will compare (5, 3) in the second subarray we will compare(1, 9) similarly in the third and fourth we will compare (8, 2) and (4, 7) respectively. Here in this array, we will get subarrays sorted in a single pass only. Our Sorted subarrays are as below:

Now we will merge these subarrays into a single sorted array.

For that, we will perform a merge on two subarrays parallelly. In the first two subarrays whose indices are i=0 and j=0.



Here now we get 2 sorted arrays and further merge them in a similar way as above



Hence the above array is the final sorted array.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

EXPERIMENTAL RESULTS

The experimental test is performed under consideration of 4 cores. It is performed on data generated randomly using the c++ library. The results are taken on a 2.30 GHz Intel Core i3 processor with 4 GB 2133MHz SODIMM memory machine.

We tested the proposed bubble sort algorithm and compared it with the existing bubble sort, selection sort, and insertion sort algorithms because they belong to the same family, $O(n^2)$, and use basic comparing and swapping operations. Here proposed algorithm complexity remains the same as $O(n^2)$ only the difference comes in its execution time where it's quite efficient than classical bubble sort as the number of comparisons is reduced in a new one.

Table 2. Execution time of different sorting algorithms

Sorting Algorithm	**Execution Time to sort (seconds) for input size n**							
	1000	**2000**	**3000**	**4000**	**5000**	**6000**	**7000**	**10000**
Bubble Sort	0.00157	0.00953	0.02162	0.04056	0.05872	0.08080	0.10669	0.22893
Selection Sort	0.00172	0.00541	0.01284	0.02168	0.03399	0.03734	0.05687	0.11489
Insertion Sort	0.00098	0.00403	0.00859	0.01624	0.02424	0.03188	0.04564	0.07169
Proposed Bubble Sort	0.0165	0.00568	0.01105	0.02896	0.03589	0.03542	0.05068	0.10652

Figure 2 represents execution time in seconds to sort the array along (Y-axis) for classical bubble sort, selection sort, insertion sort, proposed bubble sort algorithm with various array sizes along (X-axis). We started with 1000 elements and finished with 10000 elements.

This figure shows that the proposed bubble sort is better than classical bubble sort but is less efficient with selection from 3000 elements to 5000. And then the proposed bubble sort is better than selection sort but overall insertion is still more efficient then the proposed algorithm.

CONCLUSION AND FUTURE WORK

Managing massive amounts of data has always been a problem in our digital era. Although there are many sorting algorithms, each one has advantages and disadvantages of its own. We focused on comparison-based algorithms. The proposed sorting algorithm is carried out while keeping in mind that it takes minimum execution time and the highest possible speed for a given situation.

The novel sorting algorithm that is introduced in this paper is evaluated against other sorting methods. Using an experimental database and the same setup-a computer with the same configuration, an IDE (Code Blocks), a method that is used consistently, etc. We have shown that the proposed bubble sort algorithm is more efficient than the classical bubble sort algorithm.

Our study proved the effectiveness of sorting clusters as compared to sorting the entire data set, and our suggested method is also successfully tested. Our proposed algorithm generally struggles to function at its best on other other sorts of datasets because it requires constant comparing, swapping, etc. Even so, based on our extensive experimental datasets, we are certain that it will surpass the Big-$O(n^2)$ time complexity sorting algorithm.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

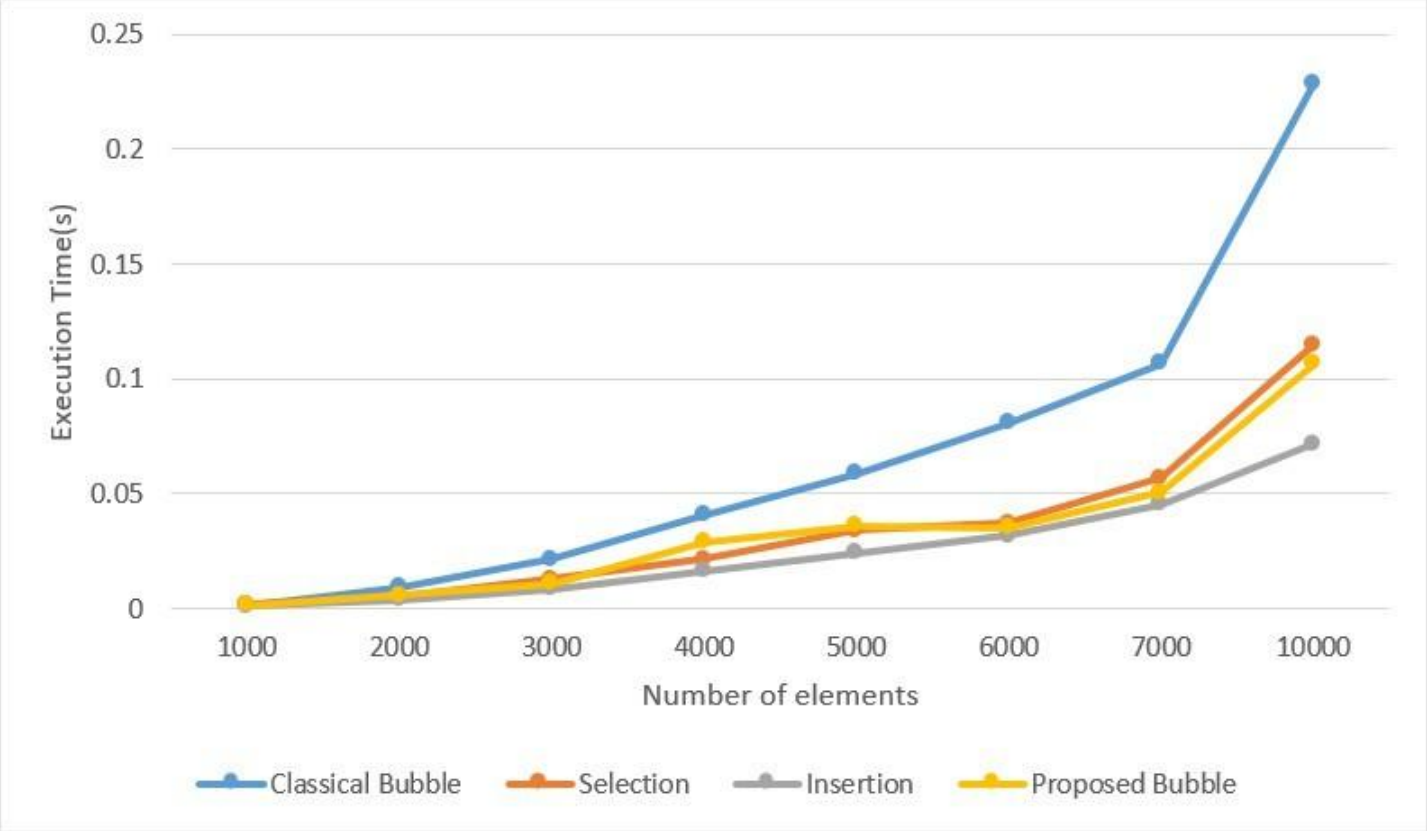


Figure 2. Comparison of execution time between sorting algorithms

The present study was limited to four core devices which yield roughly more speed and less execution time. Future work will focus on identifying ways to reduce the time complexity, simplify and optimize our suggested algorithm so that it may be used in a variety of practical and real-world situations.

REFERENCES

1. Gugale, Y. (2018, April). Super sort sorting algorithm. In 2018 3rd International Conference for Convergence in Technology (I2CT) (pp. 1-5). IEEE.
2. Bijoy, M. H. I., Hasan, M. R., & Rabbani, M. (2020, July). RBS: a new comparative and better solution of sorting algorithm for array. In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-5). IEEE.
3. Rana, M. S., Hossin, M. A., Mahmud, S. H., Jahan, H., Satter, A. Z., & Bhuiyan, T. (2019). MinFinder: A new approach in sorting algorithm. Procedia Computer Science, 154, 130-136.
4. Roy, H., Shafiuzzaman, M., & Samsuddoha, M. (2019, December). SRCS: A New Proposed Counting Sort Algorithm based on Square Root Method. In 2019 22nd International Conference on Computer and Information Technology (ICCIT) (pp. 1-6). IEEE.
5. Patel, Y. S., Singh, N. K., & Vashishtha, L. K. (2014, September). Fuse sort algorithm a proposal of divide & conquer based sorting approach with $O(n \log\log n)$ time and linear space complexity. In 2014 International Conference on Data Mining and Intelligent Computing (ICDMIC) (pp. 1-6). IEEE.
6. Yerram, B., & Bhonagiri, J. K. (2020, July). An efficient sorting algorithm for binary data. In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-4). IEEE.
7. Ferrada, H. (2022). A sorting algorithm based on ordered block insertions. Journal of Computational Science, 64, 101866.
8. Alotaibi, A., Almutairi, A., & Kurdi, H. (2020). OneByOne (OBO): A fast sorting algorithm. Procedia Computer Science, 175, 270-277.
9. Zutshi, A., & Goswami, D. (2021). Systematic review and exploration of new avenues for sorting algorithms. International Journal of Information Management Data Insights, 1(2), 100042.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

10. Garg, A., Patel, V., & Mishra, D. (2022, May). Mid-Point Sorting Algorithm: A New Way to Sort. In 2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES) (pp. 65-71). IEEE.