Machines and Algorithms
http://www.knovell.org/mna

Research Article

Min Max Merge: A Novel Comparison based Sorting Technique for Data-Intensive Processing

Abbas Mubarak[1, *]

[1]Department of Computer Science, Institute of Southern Punjab, Multan, 60000, Pakistan
[*]Corresponding Author: Abbas Mubarak. Email: abbas007sheikh@gmail.com

Received: 02 August 2024; Revised: 3 September 2024; Accepted: 30 September 2024; Published: 10 October 2024
AID: 003-03-000041

Abstract: Sorting is significant as it is a prerequisite for many applications and functions. It is one of the oldest topics in literature which is still as important as it was in the beginning. Researchers are still working to design more efficient sorting methods either by improving existing sorting methods by reducing time complexity, space complexity, comparisons and shift operations or formulating new ones as sorting is very essential for most used applications such as databases for extracting useful information efficiently. This paper presents a novel comparison-based sorting method named as min max merge sort which works by merging groups of numbers into one group. The Proposed sorting algorithm is not only better than some old classical comparison-based sorting algorithms but it also performs better than some latest presented sorting methods. The basic idea behind proposed sorting algorithm is to divide the input array into different groups and then perform their recursive merging on the basis of their maximum and minimum entries, which circumvents unnecessary data shifts and comparisons. The speed of proposed algorithm is comparatively faster than the traditional sorting methods, as it exploits localized minimum and maximum data entries instead of recursively scanning entire input array. This algorithm exploits linear auxiliary space, as it performs in-place operations for combining groups in same array. Space complexity of proposed sorting method is O(n) as it does not require extra memory space. Performance comparison of proposed method with other sorts shows the superiority of our proposed method.

Keywords: Sorting; 2mm Sort; Time Efficient; Sorting Complexity;

1. Introduction

There are several types of algorithms, such as sorting algorithms, searching algorithms, compression algorithms and path finding algorithms [1]. Among the various branches of algorithms sorting is an important domain which is the oldest and most studied module in computer science [2]. The rearrangement of input data in a specific manner is known as sorting [3], [4]. Since 1950s, computer scientists are working on various sorting algorithms to improve the complexity of old sorting methods in terms of space and time complexity [5]–[8]. A number of top computer scientists extensively studied this topic such as Turing award winner Tony Hoare, Von Neumann and Donald Knuth [9]. Sorting is integral part of approximately all computer and mobile applications. All software's uses different sorting methods however user always concern with fast sorting [10]. People always consider two things in all sorting functions, speed of sorting algorithm and simplicity of algorithm [1].

Sorting can be performed on numbers, strings or records containing both numbers and strings like IDs, names, or departments, etc. [11]. Each sorting algorithm has unique properties that add value to the specific function they are used to perform [12]. Every sort is not appropriate for every kind of data [11]. Every algorithm has its own best as well as worst case. Therefore, to find out best sorting method according to Big O is difficult [13]. A study is essential in order to develop or make new sorting algorithm, because not all algorithm works efficiently for the same problem [14]. As the world has become global village and information is increasing rapidly, therefore importance of efficient sorting has also increase [4]. To get efficient search results from big databases also requires data in specific order [15]. The way of sorting data is very important due to impact of execution time, how elements are swapped, compared and arranged is dependent of technique of particular sorting algorithm [1], [16].

Enormous number of sorting algorithms are in use to date. Out of which, bubble sort is used extensively as it is simple and spontaneous however it is not very efficient. Normally a sorting algorithm consists of comparison, swap, and assignment operations [17]. Every algorithm has different time complexity [10]. Sorting can be applied in three ways: vector sort, (list) table sort and address sort, depending on the data storage policy [18]. This study, use array data structure to store data for sorting where data can be in numeric or character form [19]. Sorting algorithms for serial computing (random access machines) allow only one operation to execute at a time. The sorting algorithms based on a comparison network model of computation, performed many operations simultaneously [3].

2. Literature Review

2.1. Overview of Sorting Methods

Sorting methods can be divided into two major groups: specialized sorts and general sorts. Each sorting algorithm possesses some particular properties i.e. each particular algorithm performs best when data is of specific type like small numbers, floating point numbers, big numbers and repeated numbers. Apart from the programming work, the availability of main memory, the size of disc or tape units, and the degree of list already ordering consideration in selecting a sorting method. Most of the sorting techniques are thus problem specific, that is, they perform effectively on some particular type of data or problem [11], [17]. From memory consumption standpoint, some sorting techniques require more space in memory than others; yet, these techniques allow faster sorting. Consequently, the choice of these methods relies on the location and purpose of the sorting of the inputs [1].

The comparison results can be obtained in three ways: 1, Comparison of execution time 2, Comparison of total number of comparisons and 3, Comparison of total swapping frequency [19]. Among the numerous sorting techniques available, the one optimal for a given application depends on several variables like data size, data type, and element distribution in a data collection. The efficiency of the sorting algorithm is also dynamically influenced by numerous factors that may be aggregated as the number of comparisons (for comparison sorting), number of swaps (for in place sorting), memory utilization and recursion [20].

2.2. Categorization of Sorting Methods

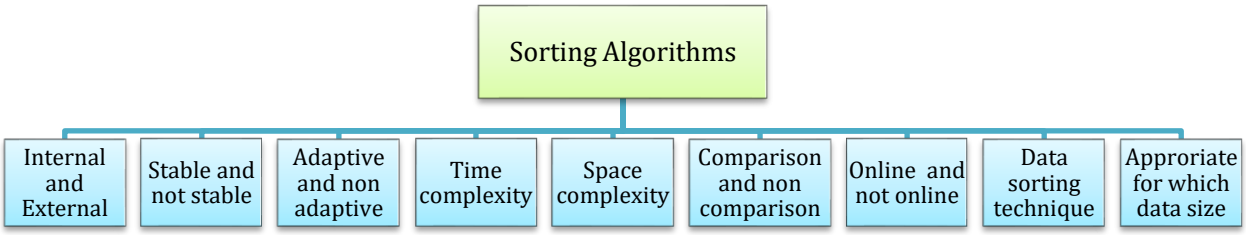

Figure 1: Categorization of Sorting Algorithms

Researchers have categorized the sorting methods in different categories i.e., depicted in figure 1 above [3], [12], [14], [18], [20]–[25].

2.2.1. Internal and External Sorting

Sorting method either do internal or external sorting. This is a key aspect for determining computational cost of sorting method. If a sorting function is performing sorting in Random Access Memory (RAM), which is also known as primary memory, then it is called internal sorting, whereas if the sorting function is using external memory, which is also known as secondary memory, then it is called externals sorting. In internal sorting, input numbers first load in main memory RAM and then processed inside RAM. Usually, internal sorting is much faster as it does not require data to be fetched from outside RAM. Some examples of internal sorting are 2mm sort, MMBPSS sort, Bubble sort, Min-Finder sort, Insertion sort and Selection sort [22]. While sorting algorithms which follows external sorting mechanism first brought data into primary memory from secondary storage and sorting is done. The process of fetching input data continuous during the sorting process as RAM sometimes could not store all data when the input size is too big. Heap sort, distribution sort and external merge sort are examples of external sorting.

2.2.2. Stable and Non-Stable Sorting

Some sorting methods are in the category of stable sorts while some are called not stable sorting algorithm. A sorting algorithm which retains the original input order after applying sorting process is known to be stable sorting algorithm. Stable sorting algorithm is needed where we have to retain sequence of equal values. For example, line of people waiting for some process according to their ages i.e. person with more age will be processed earlier but due to equal age stable sort will preserve the original sequence record i.e. first come first serve. Whereas those who are not stable sorting algorithms can change the place of occurrences of equal values in resultant list. Stability is usually not required when all the elements are different. Insertion sort, merge sort, bubble sort and counting sort are examples of stable sorting algorithms while heap sort, quick sort, selection sort and shell sort are examples of not stable sorting algorithms.

2.2.3. Adaptive and Non-Adaptive Sorting

An adaptive sorting algorithm is one that takes advantage of particular input sequence resulted decrease in time complexity. Insertion sort is an example of adaptive sort. It performs very fast when input data is nearly sorted. Quick sort is another example of adaptive sorting. Its time complexity reflects in different variation of input data such as random, sorted and reverse sorted.

2.2.4. Time and Space Complexity

According to time complexity sorting algorithms can majorly be divided into two groups i.e. O(nlogn) and O(n2). Insertion Sort, Bubble Sort, MinFinder and Selection Sort and 2mm sort comes under N^2 family while heap sort and quick sort belongs to time complexity O(nlogn). Space complexity determines the memory space taken by any algorithm during execution, and therefore this is an important aspect in time complexity. Space complexity of various algorithms can be different and the most efficient are those with less space.

2.2.5. Comparison and Non-Comparison Sorting

In comparison-based sorting algorithm, sorting is done by comparing numbers with each other to find minimum or maximum number for its proper position. Insertion, Quick and Bubble sort are some examples of comparison-based sorting. While in non-comparison-based sorting numbers comparison is not done and numbers are sorted using some other technique. Bucket and Radix sort are examples of non-comparison-based sorting.

2.2.6. Online and Offline Sorting

A sorting method that can work at the time of input given and full array of numbers is not required to input before processing, this process is known as online sorting. Whereas in offline sorting the whole input must be entered before algorithms processing. Insertion sort is an example of online sorting.

2.3. Characteristics of Good Sorting Algorithm

Some important characteristics of algorithm are [14], [26].

- **Finiteness:** It must be stopped after execution of restricted numbers of steps.
- **Definiteness:** All steps of an algorithm must be unambiguous and clearly defined.
- **Input:** The algorithm must have input values from a definite set.
- **Output:** An algorithm must produce some output form the given input.
- **Effectiveness:** The algorithm should execute each step exactly using a certain amount of time.
- **Correctness:** The algorithm must produce the correct output values for every finite set of inputs.

2.4. Sorting Algorithms Applications

Many algorithms, in addition to their primary function of sorting, also make use of a variety of methods to sort lists as a preparatory step in order to cut down on the amount of time it takes for them to carry out their tasks [20]. Sorting is also significant for searching, merging and normalization [10], [12], [14], [17], [18]. Successful sorting is essential to improve the utilization of another algorithm [23]. Sorting is also used in Central Processing Unit (CPU) scheduling in Operating system, recommendation system based on search time, Television channels sorted based on view time [27]. Sorting is inevitable in query retrieval and different types of join such as sort merge join [15]. Database query processing needs algorithms for duplicate removal, grouping, and aggregation, whereas in-stream aggregation is most efficient by far but requires sorted input [28].

3. Motivation

The author's in [1] presented a novel sorting algorithm which sorts the input list without comparisons, however it performs some manipulations with array indices for the purpose of sorting. The proposed sorting method in [1] uses two arrays for input sorting and two arrays for calculating repeated values. The drawback of above-mentioned sorting method is the extra usage of memory space [1]. The author's in [19] proposed new variant of selection sort algorithm called MMBPSS and it sorts the input list by finding out minimum and maximum numbers from first and second half of input list separately. It than compares the obtained numbers of both halves to find out minimum and maximum number of complete lists. The drawback of this algorithm is that it finds 4 numbers in each iteration and uses only 02 numbers. The authors in [2] brought a new variant of Timsort algorithm named as adaptive shivers sort. Its computational cost and number of comparisons are less than Timsort algorithms. The author's in [29] presented a novel comparison free sorting method with input K-bit binary bus. It operates on one-hot weight representation. For example, element 5 in binary is 101 whose one-hot representation is 100000. This binary to one-hot representation can be done using conventional one-hot decoder [29]. The authors in [30] introduced a new sorting method min-max sorting algorithm which works by finding minimum and maximum number form the input list and adjust them at first and last index of the list [6].

4. Review of 2mm Sorting Method

2mm sorting method is a comparison-based sorting method. In one cycle it changes 02 numbers as briefed in Pseudo Code below.

2mm Pseudo Code [22]
"Length ← size of (array)

```
midindex←Length/2
minindex←0
maxindex←Length -1
min ← a [minindex]
max ← a [minindex]
minloc ←maxloc←0


For i= minindex+1 to midindex
    If a[i] <min
        min←a[i]
        minloc←i
    End If
    If a[i]>max
        max←a[i]
        maxloc←i
    End If
  exchange a[minindex] with a[minloc]
  exchange a[midindex] with a[maxloc]
End For loop
 min ← a [midindex+1]
 max ← a [midindex+1]
For i= midindex+2 to maxindex
    If a[i] <min
        min←a[i]
        minloc←i
    End If
    If a[i]>max
        max←a[i]
        maxloc←i
    End If
  exchange a[midindex+1] with a[minloc]
  exchange a[maxindex] with a[maxloc]
End For loop
  minindex←0
  maxindex←Length -1
  min ← a [minindex]
  max ← a [maxindex]
```

From beginning to mid index and mid + 1 to end index, 2mm determines minimum and maximum numbers. Thus, it determines minimum and maximum number of lists by comparing minimum and

maximum numbers to ascertain minimum and maximum numbers of full array [22]. Next iterations save the previously discovered numbers and follow the same continuous process until all numbers are arranged.

Initially some variables are initializing. Then a for loop will execute from first half of array and another for loop will execute for another half of array. Each for loop will find minimum and maximum numbers of respective sub arrays and adjusts them at first and last index of that sub part of array. The above code will make the input list executable for next mail while loop processing. The main while loop required the input list in a manner where minimum and maximum numbers needs to be adjusted at each sub part of array for execution of one of the four cases inside main while loop. After the above code execution there can be 04 possibilities of array therefore for each possibility while loop have 04 cases and respective case only will be executed after if decision. One case is where both maximum and minimum numbers are already at their correct positions i.e. minimum number at first index and maximum number at last position. Second case is both numbers are not at their positions i.e. minimum number at mid+1 index and maximum number is at mid index. Third case is where both maximum and minimum numbers of whole array are at first sub array and last case is where both numbers are at second sub arrays [22].

```
While (minindex < midindex-1)
{
    If (min<=a[midindex+1])
    {
      If( max>=A[midindex]
        {
          Case1
        }
      Else
        {

          Case 2

        }
    }
    Else
    {
      If(max>=a[midindex])
        {
          Case 3
        }
      Else
        {
          Case 4
        }
    }
}
```

5. Proposed Algorithm - Min Max Merge Algorithm

The proposed sorting algorithm belongs to comparison-based family and it uses preprocessing presented in [31]. The basic operation in comparison-based sorting method is comparison of two input elements [32]. In its initial phase after applying preprocessing, it first sorts the input array into groups of two consecutive elements by comparing them. It than merge the input list elements in multiplication of 2 i.e. first sort group of two elements, then combine two groups and sort 4 elements, following up combine two groups of 4 element and sort 8 elements and same process continues until all elements are not sorted. In case of remaining last elements which are left without any group it adjusts them into previous groups. For example, in the input list of 10 elements when min max merge will make group of 4 elements than last 2 element will

be adjust with second group i.e. index 4 to 7 where index starts from 0. As min max merge is not use additional array for merging process therefore it combines group of elements in the same array by bubbling the elements. Min max merge 2mm sort min max formula and merge sort merging technique for sorting process.

At Mendeley Data the source code of Min Max Merge sorting algorithm is available (https://dx.doi.org/10.17632/sjxbcn97n6.1)

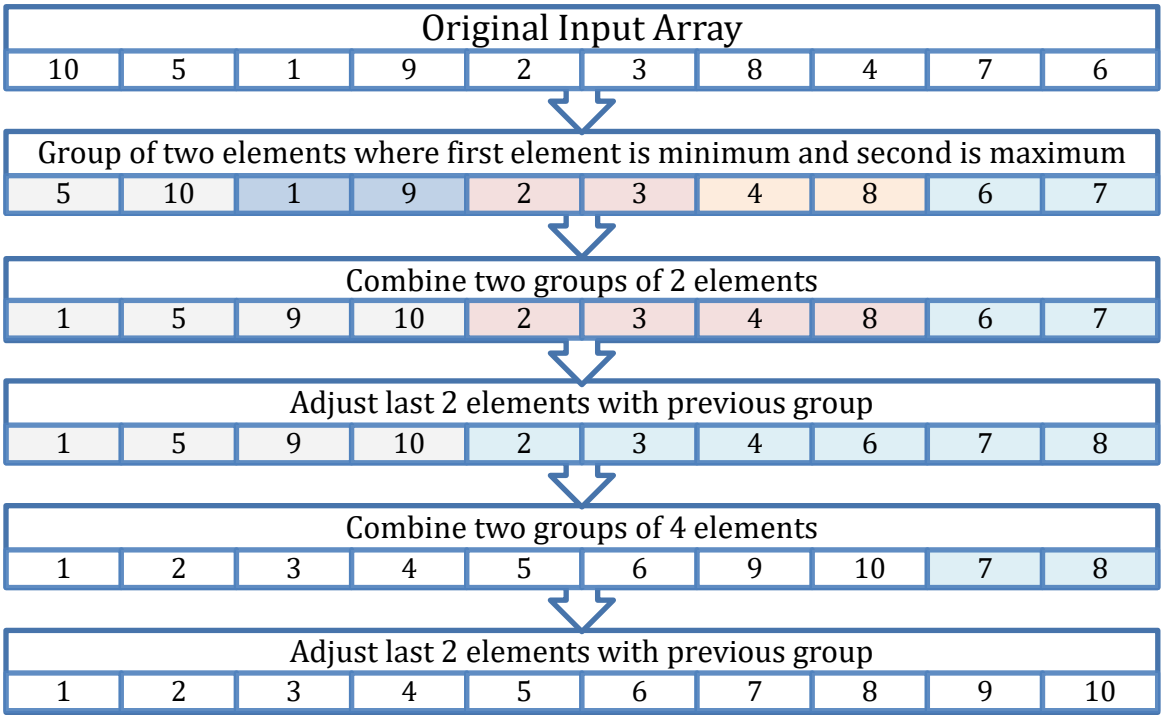

Figure 1: Dry-Run of Proposed Sorting Mechanism

5.1. Mathematical Analysis

Time complexity for applying data preprocessing as adapted by the study [31], is calculated below in equation 1 .

$$T(n) = O\left(\frac{n}{2}\right) + O\left(\frac{n}{2} - 3\right) + O(\frac{n}{2} - 3) + O(\frac{n}{4} + 1) + O(\frac{n}{2}) + O\left(\frac{n}{2} - 3\right) + O\left(\frac{n}{2} - 3\right)$$

$$T(n) = C + (\frac{2n + 2n - 12 + 2n - 12 + n + 4 + 2n + 2n - 12 + 2n - 12}{4})$$

$$T(n) = C + (\frac{13n - 44}{4})$$

$$T(n) = \Omega(n) \quad (1)$$

Time complexity for proposed method has been calculated below in equation 2. To sort input list in the manner of minimum and maximum $O\left(\frac{n}{2}\right)$ will execute. Then loop will be executed to adjust group of 4 numbers and will execute $\left(\frac{n}{4}\right)$ times and it will execute $\left(\frac{n}{4}\right)$ times for whole array. Time complexity will be,

$$T(n) = O\left(\frac{n}{2}\right) + \left(\frac{n}{4}\right)\left(\frac{n}{4}\right) + \left(\frac{n}{8}\right)\left(\frac{n}{8}\right) + \left(\frac{n}{16}\right)\left(\frac{n}{16}\right) + \cdots \left(\frac{n}{n}\right).1$$

$$T(n) = O\left(\frac{n}{2}\right) + \left(\frac{n^2}{16}\right) + \left(\frac{n^2}{64}\right) + \left(\frac{n^2}{256}\right) + \cdots 1$$

$$T(n) = O\left(\frac{128n + 16n^2 + 4n^2 + n^2}{256}\right) + \cdots 1$$

$$T(n) = O\left(\frac{128n + 21n^2}{256}\right) + \cdots 1$$

$$T(n) = O(n^2) \quad (2)$$

6. Results and Discussion

In order to accomplish the task of reducing the amount of computational complexity and time required to carry out swapping, comparison, and assignment operations, an efficient sorting algorithm is being developed [4]. The experiment results of insertion, bubble, selection, MMBPSS, MinFinder and 2mm were taken for comparison from our preceding study [22]. These results are compared with our proposed sorting algorithm results.

Table 1: Computational Time analysis of different sorting algorithms in seconds on random input data

Sorts | input	Bubble	Selection	Insertion	MMBPSS	MinFinder	2mm	Min max merge
100	0.000727	0.0057	0.0047	0.0051	0.0052	0.000333	0.0010053
1000	0.001999	0.0009994	0.0009994	0.0010068	0.002001	0.000999	0.0249867
100000	27.5589	11.3939	7.00767	10.8788	16.6568	4.16842	2.13268
200000	114.27	46.2444	29.046	27.5309	66.729	18.0029	8.54076
300000	257.078	104.649	64.0444	62.0226	148.685	41.0046	19.1262
400000	458.822	185.428	113.453	109.771	268.054	73.2097	33.8753
500000	851.909	250.735	173.653	181.682	393.721	111.081	89.8946

Table 2: Computational Time Analysis on Reverse Data to analyze Worst Case

Input| Sorts	Insertion	Bubble	Selection	MinFinder	MMBPSS	2mm	Min max merge
1000	0.0019997	0.002001	0.0019988	0.002	0.0010068	0.0009986	0.0001
10000	0.137915	0.176891	0.118927	0.235857	0.0699651	0.0579659	0.0007
100000	14.0323	17.806	12.7451	23.1758	6.55928	4.90797	0.0019977
200000	57.1027	72.666	48.548	92.0527	26.2981	19.7298	0.003998
300000	129.583	164.516	109.16	210.879	59.578	44.2776	0.006995
400000	225.351	291.462	195.492	366.052	106.495	78.3613	0.007995
500000	371.548	538.847	289.507	691.669	183.389	126.828	0.0099765

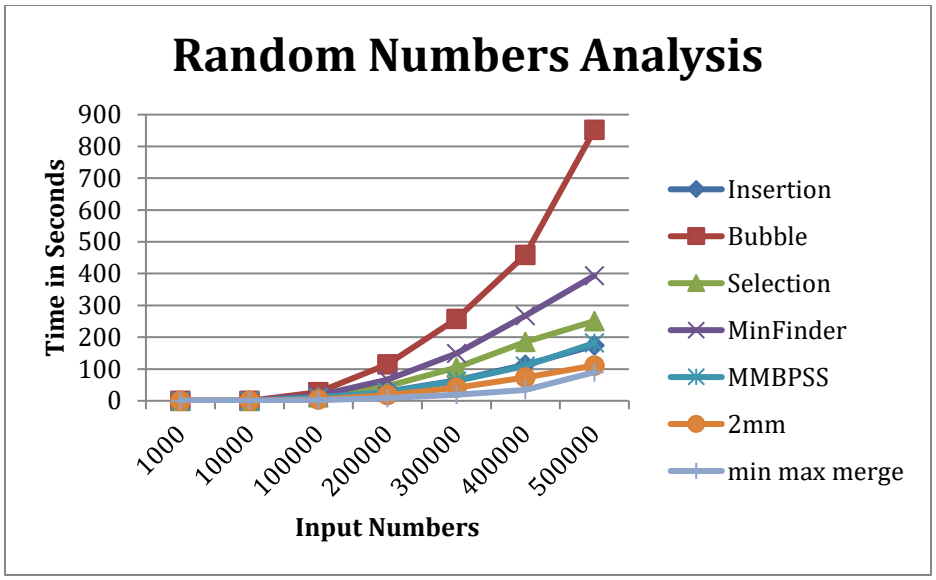

Figure 2: Computational Time visualization of different sorting algorithms in seconds on random input data

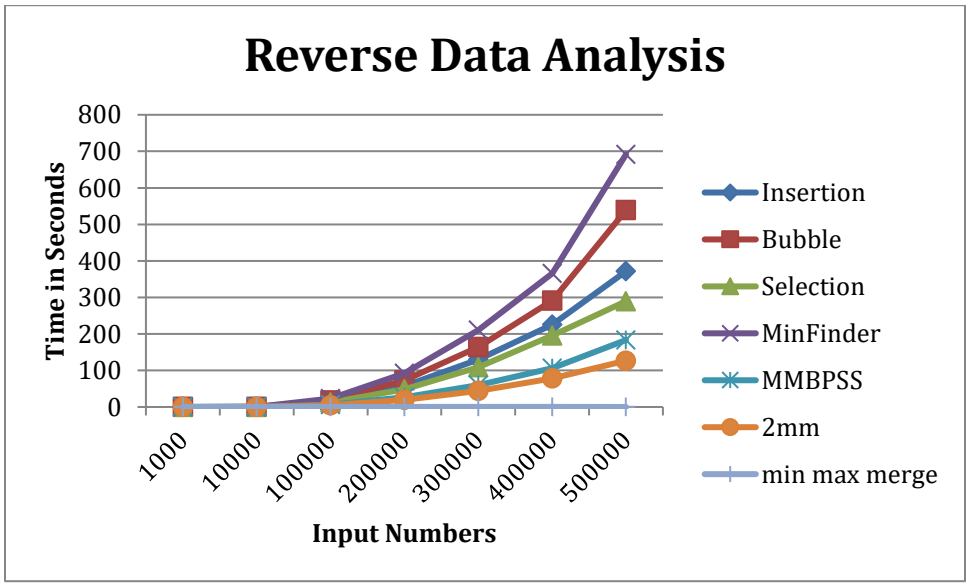

Figure 3: Computational Time Visualization on Reverse Data to analyze Worst Case

Based on the analysis of results, mentioned in Table 1, 2 and Figure 3, 4 it could be said that, the proposed Min Max Merge Sort shows superior performance with random data inputs and achieves better computational time than all other algorithms especially when working with large datasets such as 500k elements which finished execution in 89.89 seconds while Bubble Sort required 851.91 seconds. The performance of Min Max Merge Sort remains stable with worst-case input data because it demonstrates limited runtime slowness while outperforming all other techniques (for example, achieving 0.0099 seconds for 500k elements as opposed to 538.84 seconds for Bubble Sort).

7. Conclusion

Sorting is significant branch of algorithms. Researchers are continuously working on developing efficient

sorting methods and improving available functions due to phenomenal increase of data. This research presents new sorting method with preprocessing technique. It uses merge technique for sorting like merge sort. The huge difference is that where merge sort uses extra array for combining two groups of pieces, min max merge combines the two groups into one group in same memory space and this was the biggest challenge in this study. The proposed algorithm can be used for various types of data as it also uses preprocessing before applying original algorithm which helps in making data suitable for algorithm. Extensive analysis proves that proposed sorting method with preprocessing technique is significantly better than other comparison-based sorting methods. The authors aim to develop algorithm parallelized version in future.

Ethical Approval: The purpose of this research is to develop a computational model, rendering it unnecessary to involve human or animal subjects.

Funding Statement: This research has not received funding from any external source.

Conflicts of Interest: Author of this study declare no conflicts of interest.

Data Availability: In addition to the raw data, program code and supplementary materials, we have provided detailed documentation outlining the methodology, data collection procedures, and analysis techniques employed in this study.

References

[1] F. Idrizi, A. Rustemi, and F. Dalipi, “A new modified sorting algorithm: a comparison with state of the art,” in *2017 6th Mediterranean Conference on Embedded Computing (MECO)*, 2017, pp. 1–6.

[2] V. Jugé, “Adaptive Shivers sort: an alternative sorting algorithm,” arXiv Prepr. arXiv1809.08411, 2018.

[3] O. O. Moses, “Improving the performance of bubble sort using a modified diminishing increment sorting,” Sci. Res. Essay, vol. 4, no. 8, pp. 740–744, 2009.

[4] S. S. Moghaddam and K. S. Moghaddam, “On the performance of mean-based sort for large data sets,” IEEE Access, vol. 9, pp. 37418–37430, 2021.

[5] M. Shabaz and A. Kumar, “SA sorting: a novel sorting technique for large-scale data,” J. Comput. Networks Commun., vol. 2019, no. 1, p. 3027578, 2019.

[6] W. H. Ford, Data Structures with C++ Using STL, 2/e. Pearson Education India, 2002.

[7] H. Rohil and M. sha, “Run Time Bubble Sort – An Enhancement of Bubble Sort,” Int. J. Comput. Trends Technol., vol. 14, pp. 36–38, 2014, doi: 10.14445/22312803/IJCTT-V14P109.

[8] R. Shah, R. Gadia, and A. Joshi, “A Novel Approach to Sorting Algorithm,” in Research Advances in Network Technologies, CRC Press, 2023, pp. 179–190.

[9] P. Olukanmi, P. Popoola, and M. Olusanya, “Centroid Sort: a clustering-based technique for accelerating sorting algorithms,” in 2020 2nd International Multidisciplinary Information Technology and Engineering Conference (IMITEC), 2020, pp. 1–5.

[10] H. R. Singh and M. Sarmah, “Comparing rapid sort with some existing sorting algorithms,” in Proceedings of Fourth International Conference on Soft Computing for Problem Solving: SocProS 2014, Volume 1, 2015, pp. 609–618.

[11] M. H. I. Bijoy, M. R. Hasan, and M. Rabbani, “RBS: a new comparative and better solution of sorting algorithm for array,” in 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2020, pp. 1–5.

[12] P. Kumar, A. Gangal, S. Kumari, and S. Tiwari, “Recombinant sort: N-dimensional cartesian spaced algorithm designed from synergetic combination of hashing, bucket, counting and radix sort,” arXiv Prepr. arXiv2107.01391, 2021.

[13] M. Marcellino, D. W. Pratama, S. S. Suntiarko, and K. Margi, “Comparative of advanced sorting algorithms (quick sort, heap sort, merge sort, intro sort, radix sort) based on time and memory usage,” in 2021 1st international conference on computer science and artificial intelligence (ICCSAI), 2021, vol. 1, pp. 154–160.

[14] M. S. Rana, M. A. Hossin, S. M. H. Mahmud, H. Jahan, A. K. M. Z. Satter, and T. Bhuiyan, “MinFinder: A new approach in sorting algorithm,” Procedia Comput. Sci., vol. 154, pp. 130–136, 2019.

[15] A. Prasad, M. Rezaalipour, M. Dehyadegari, and M. Nazm Bojnordi, “Memristive Data Ranking,” 2021, pp. 440–452, doi: 10.1109/HPCA51647.2021.00045.

[16] A. B. G. Santos, M. F. Ballera, M. V Abante, N. P. Balba, C. B. Rebong, and B. G. Dadiz, “Asymptotic analysis of the running time performed by various sorting algorithms,” in 2021 International Conference on Intelligent Technologies (CONIT), 2021, pp. 1–6.

[17] M. Khairullah, “Enhancing worst sorting algorithms,” Int. J. Adv. Sci. Technol., vol. 56, pp. 13–26, 2013.

[18] W. Min, “Analysis on bubble sort algorithm optimization,” in 2010 International forum on information technology and applications, 2010, vol. 1, pp. 208–211.

[19] K. Thabit and A. A. BAWAZIR, “Novel approach of selection sort algorithm with parallel computing and dynamic programing concepts,” J. King Abdulaziz Univ. Comput. Inf. Technol. Sci., vol. 2, pp. 27–44, 2013.

[20] A. S. Mohammed, \cSahin Emrah Amrahov, and F. V Çelebi, “Bidirectional Conditional Insertion Sort algorithm; An efficient progress on the classical insertion sort,” Futur. Gener. Comput. Syst., vol. 71, pp. 102–112, 2017.

[21] N. Faujdar and S. P. Ghrera, “Analysis and testing of sorting algorithms on a standard dataset,” in 2015 Fifth International Conference on Communication Systems and Network Technologies, 2015, pp. 962–967.

[22] A. Mubarak, S. Iqbal, T. Naeem, and S. Hussain, “2 mm: A new technique for sorting data,” Theor. Comput. Sci., vol. 910, pp. 68–90, 2022.

[23] S. M. Cheema, N. Sarwar, and F. Yousaf, “Contrastive analysis of bubble \& merge sort proposing hybrid approach,” in 2016 Sixth International Conference on Innovative Computing Technology (INTECH), 2016, pp. 371–375.

[24] A. Shatnawi, Y. AlZahouri, M. A. Shehab, Y. Jararweh, and M. Al-Ayyoub, “Toward a new approach for sorting extremely large data files in the big data era,” Cluster Comput., vol. 22, pp. 819–828, 2019.

[25] P. Prajapati, N. Bhatt, and N. Bhatt, “Performance comparison of different sorting algorithms,” vol. VI, no. Vi, pp. 39–41, 2017.

[26] D. E. Knuth, The art of computer programming, vol. 3. Pearson Education, 1997.

[27] S. K. Gupta, D. P. Singh, and J. Choudhary, “New GPU Sorting Algorithm Using Sorted Matrix,” Procedia Comput. Sci., vol. 218, pp. 1682–1691, 2023.

[28] T. Do, G. Graefe, and J. Naughton, “Efficient sorting, duplicate removal, grouping, and aggregation,” ACM Trans. Database Syst., vol. 47, no. 4, pp. 1–35, 2023.

[29] S. Abdel-Hafeez and A. Gordon-Ross, “An Efficient O ($ N $) Comparison-Free Sorting Algorithm,” IEEE Trans. Very Large Scale Integr. Syst., vol. 25, no. 6, pp. 1930–1942, 2017.

[30] A. Agarwal et al., “A new approach to sorting: min-max sorting algorithm,” SORT, vol. 2, no. 2, p. n2, 2013.

[31] A. Mubarak, S. Iqbal, Q. Rasool, N. Asghar, N. Faujdar, and A. Rauf, “Preprocessing: A method for reducing time complexity,” J. Comput. \& Biomed. Informatics, vol. 4, no. 01, pp. 104–117, 2022.

[32] K. Iwama and J. Teruyama, “Improved average complexity for comparison-based sorting,” Theor. Comput. Sci., vol. 807, pp. 201–219, 2020.