A Mathematical based Divide and Conquer Approach to a New Sorting Algorithm with Min Max Index Value

Dr.S. Muthusundari[1*], L. Sherin Beevi[2], G. Shankar[3], U. Archana[4], Dr.G. Nirmala[5]

[1*]Associate Professor, Department of CSE, R.M.D. Engineering College, Kavaraipettai, sms.cse@rmd.ac.in, ORCID: https://orcid.org/0000-0003-1340-8008

[2]Assistant Professor, Department of CSE, R.M.D. Engineering College, Kavaraipettai, lsb.cse@emd.ac.in, ORCID: https://orcid.org/0000-0002-1180-7550

[3]Assistant Professor, Department of CSE, R.M.D. Engineering College, Kavaraipettai, gs.cse@emd.ac.in, ORCID: https://orcid.org/0000-0002-3660-1778

[4]Assistant Professor, Department of CSE, R.M.K. Engineering College, Kavaraipettai, aau.csd@rmkec.ac.in, ORCID: https://orcid.org/0000-0003-1169-1284

[5]Assistant Professor, Department of CSE, R.M.D. Engineering College, Kavaraipettai, gns.cse@emd.ac.in, ORCID: https://orcid.org/0000-0002-3684-8965

Received: November 16, 2022; Accepted: January 18, 2023; Published: February 28, 2023

Abstract

Information sorting is used in a large range of applications and is critical in determining current effectiveness, efficiency, and consumption. Sorting is not just essential in data science; it is also an important aspect of managing algorithms. It's a difficult challenge to choose the optimal sorting algorithm for a certain application. The trade-off between resource use, speed, and area is the basis for this. This study establishes a helpful link between sorting and categorization, resulting in a general method for speeding up sorting algorithms. In this paper, the performance evaluation of a novel sorting technique is implemented by incorporating a sub-array for obtaining the max index value from the given sub-array. Then calculate the mid value by divide and conquer tactics then proceed to binary search for identified the position of the element. This hybrid approaches yields O(nlogn) in the average cases. Sorting is becoming increasingly needed in real-time applications, such as real-time visualization, database management systems, sensor fusion, and finite element. As a result, on average, the suggested proposed sorting algorithm with modified Cycle sort outperforms the existing Sorting algorithm by 75% in time complexity. The goal of this proposed sorting algorithm is a 75%-time efficiency using the existing modified cycle sorting algorithm.

Keywords: Sub Array, Max Index, Mid Value, New Sort, Divide and Conquer, Binary Search.

1 Introduction

The majority of today's application domains need data be organized under certain capacity. Sorting is a method of putting data in order that is used in a multitude of sectors. Sorting is a technique for structuring and sifting large amounts of data (Budhani et.al,2021, Bahig et.al, 2019). It's also important in today's scientific software and professional data acquisition. Transaction processing, semantics, optimization

DOI: *10.58346/JISIS.2023.I1.009*

*Corresponding author: Associate Professor, Department of CSE, R.M.D. Engineering College, Kavaraipettai.

techniques, genomes, quantum chemical, weather prediction, and astrophysics are just a few examples. Computer science has been dealing with practical issues in sorting large amounts of data to several years. To achieve the best results, numerous operations with enormous databases demand quickly and efficiently sorting algorithms (Omar,2017). As a result, redundancy must be raised in order to build efficient sorting algorithms. The style of sorting algorithm employed determines the application's efficiency. To choose the optimum effective way, all angle measure about the user's hardware and software, as well as the ease of use and database availability, must be considered first (Wickremesinghe et.al, 2002). Among the most significant techniques in computer science is sorting.

The quest for novel techniques and better implementations is a particularly important topic in computer science research, given the vast amount of data that we may use as input. This on-going research into sorting algorithms has resulted in the development of novel sequential and parallel architecture strategies and algorithms (Mohammed, 2017, Alnihoud, ,2010). Because sorting is a crucial step in many algorithms, researching fast sorting implementations in new settings such as Graphics Processing Units (GPUs) has been crucial to improving algorithm performance (Muthusundari, 2013).

There are several workloads that are critical, and there are no productivity concessions that can be made. In such circumstances, new algorithms must be devised based on the needs.

Different tactics, like technique of brute-force, divide-and-conquer, randomised, and comparison models, have been used to sort the algorithms (Muthusundari, 2013). The complexity of a problem is determined by the approach used to solve it (Muthusundari, 2013). The comparison model, which has a lower bound complexity of equal to, can be used to attain the lower bound complexity Ω (n log n).

The summary of the research paper is follows as; the literature review in this research field is presented in section 2. Section 3 deals with the proposed algorithm and pseudo code, section 4 with the algorithm's execution and details of the complexity analysis, section 5 with the results and comparisons to other well-known algorithms, section 6 with conclusions and future work, and section 7 with the research work's important references.

2 Related Works

Because not all sorting algorithms perform well for the same task, research is required before developing or creating one as a result, recent research in the subject of sorting is required.

(Nicholas Trankle.et.al,2016) has newly designed a pseudo modelling technique known as shadow sort. Instead of processing all solutions in order to discover their corresponding Pareto set, Shadow Sort addresses all non-dominated mapping by rejecting influenced problems as quickly as possible. Shadow Sort was established in Matlab and outperforms Fastest Non-nominated Sort, Best Order Sort, and Merge Non-dominated Sort. Determined by the number of targets in the demographic, numerous Shadow Sort scenarios are suggested.

(Javier Moreno.et.al,2020) has suggested the advantages of the merge sort algorithm's properties, they compute the hegemony set, or the constraint solver that dominates each solution by comparing the MNDS to six well-known strategies that are regarded state-of-the-art in the field. The MNDS methodology exceeds the other strategies in regards of the amount of trials and the overall interval, according to the statistics.

(Alen LOVRENCIC,2020) has put forward an optimised circle sort algorithm that, depending on statistical analysis, showed to be up to 35% smoother than the predecessor. To increase overall effectiveness, the suggested approach proposes a novel, completely equitable interpretation of the centre element and eliminates superfluous procrastination. To sort an aggregate in context, the algorithm requires the divide and conquers strategy. The method has many of the same benefits as the merge sort algorithm, but it has the advantage of being in-place.

(NakibAman Turzo.et.al,2020) has recommended a modified cycle sort method, followed by a comparison of several sorting algorithms such as Cycle sort and Bubble sort. With it, they also applied a variety of additional sorting methods. A completely different notion that is still being is worked out. The CPU time consumed by each algorithm was measured using a programme written in Python. After indexing, the results were effectively connected. For essential correctness, the report was done several times, and it was determined that improved proposed cycle sort had the right approach between all cubic filter types where duplicate data was present, and that it was also the most efficient for massive datasets.

(Md. Shohel Rana.et.al,2019) has proposed a novel sorting technique using Min Finder is developed to eliminate the existing drawbacks and outperforms several systematic techniques in terms of functionality, simulation efficiency, and complexity analysis. As there have been a number of sorting techniques designed, yet these strategies have faster productivity in terms of time and space complexity, consistency, accuracy, plausibility, absoluteness, and productivity.

(Abdul Monem.et.al,2018) has implied a sorting method for parallel processing are presented in this study. The suggested methodology organises randomization and preserves those together in word document. It has three settings: number of cuts, multi-core, and time. A multitude of matrixes are used to partition the dataset. The whole algorithm provides convenient, effective, and ideal outcomes; as the count of processing increases, the prediction accuracy decreases.

(YashGugale,2018) has hinted to minimize the length of time needed to filter; the sorting strategy given in this study describes the natural series of sorted entries in an array of numeric values. The suggested algorithm has a complexity of O(nlogn), in which n is the total number of entries in the index table. On the basis of these criteria, the following table (Table.1) examines existing sorting techniques.

Table 1: Comparative Study of Existing Sort Methods

Sorting Algorithm	**Time Complexity**			**Memory Complexity**	**Stable**
	Best Case	**Normal Case**	**Terrible Case**		
Shadow Sort	O(nlogn)	O(n)	O (n2)	O (1)	Yes
MNDS with Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)	Yes
Circle Sort	O (n 2 lgN)	O(n2logn)	O (n2)	O(nlogn)	No
Modified Cycle Sort	o(n)	O(n)	O (n2)	O (1)	Yes
Min Finder Sort	O (n2)	O (n2)	O (n2)	O (1)	No
Parallel Processing Sort	Log n	O(n)	O(n)	O (1)	No
Super Sort	O(nlogn)	O(n)	O(n)	O (1)	Yes
Proposed New Sort	**O(n)**	**O(nlogn)**	**O (n2)**	**O (1)**	**Yes**

3 Proposed Methodology

The goal of this study is to implement an effective and efficient sorting algorithm with an average case complexity of O (nlogn). This can be accomplished by locating the maximum value in the specified array, using divide and conquer to compute the midpoint, and then using the Binary search technique. The following are the main components of the suggested sorting algorithm:

- Finding max value
- Apply divide and conquer
- Search the element by Binary search

The Block diagram of the proposed sorting algorithm is depicted in the fig.1.

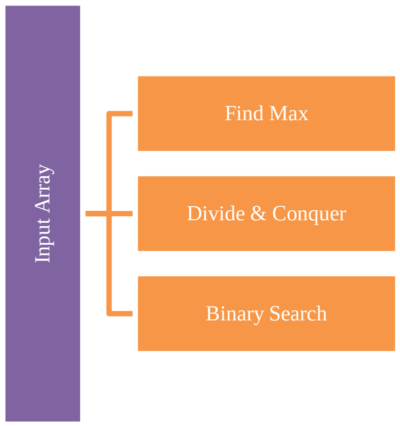


Figure 1: Block Diagram of Proposed Sorting Algorithm

The items can be sorted in ascending or descending order using the proposed sorting algorithm. Throughout each execution to the sorting algorithm, it runs through the three steps below. When the threshold standards are fulfilled, the innermost iterative call returns. Finally have a completely sorted array at the end of the latest execution. The procedure is as follows:

3.1. Foresighted Selection

The array will point to the first position in the given input list which is stored in the index 0 as assigned the initial largest member is called maxi element then assigning first element of the array to x. Creating a sub-array by excluding the first element of the original array and subtracting x value from each element of the original array. Finding the maximum of the sub array starting with the index value “ind” is altered. This recently eliminated element is now the ‘current highest,' and it is compared to another entry in the unsorted list. This decision continues until the unordered list reaches its end. Finally, a ‘forward sorted' sub-list with members in increasing order is generated, as well as an unordered list is less than its earlier size.

3.2. Hind Sighted Selection

The last element is chosen as the ‘present highest' element in this stage. It inserts the component again from unsorted list to an unfilled ‘backward sorted' array, that further holds all the filtered elements from the discrete segments pass. Then, in reverse order, it checks the ‘present biggest' with the second member in the unordered list. This decision continues until the unordered list reaches its end. Finally, a 'reverse sorted' sub-list is formed, with members in increasing order, as well as an unordered list that is smaller than before.

3.3. Amalgamation

The unsorted list is subdivided out from midway to make specific unsorted left and right sub-lists while generating the ‘intermediate sorted 1' collection. The mid value is determined by applying divide and conquer manner. This method is called iteratively on both the conservative and liberal unsorted sub-lists. The two individual sorted sub-lists are combined using the agglomerative approach to form the 'intermediary sorted 1' list. At the end of this stage, two sub lists are done. The two possibilities are the unsorted list (which is shorter than present length) and the 'intermediary sorted 1' list.

3.4. Boundary Conditions

The algorithm accepts if the unsorted sub-list has worse than one element. Because both foresighted and hind sighted selection, it removes elements from the given array, then continues until forward selection may result becoming null. As a result, after forward selection, the array's width must be at least one; otherwise, the backward selection technique will be worthless.

3.5. Proposed New Sort Algorithm

```
Algorithm New-Sort Algorithm
//Input : List L(Contains n elements)
//Output : List L(Sorted Order)
num=int(input()) #Input for total number of elements in an array
arr=list(map(int,input().split()))  #Input for elements of the array
x=arr[0]#Assigning first element of the array to x.
sub_arr=[ele-x for ele in arr[1:]] # creating a subarray by excluding the first element of the original
array and subtracting x value from each element of the original array.
for ind in range(num-1):
maxi=max(sub_arr[ind:]) # Finding the maximum of the subarray starting with the index value
"ind".
index=sub_arr. Index(maxi)#getting the index value of the maxi element of the subarray.
sub_arr[ind],sub_arr[index]=sub_arr[index],sub_arr[ind]#swapping the "ind" index element and
the maxi element in subarray.
arr[ind+1],arr[index+1]=arr[index+1],arr[ind+1] #swapping the maxi element and ind+1 indec
element in original array.
left=1;right=num-1;index=0
while left<=right: # This is nothing but binary search operation. This is done to find the correct index
to place the x element.
   mid=(left+right)//2
   if left==right:
index=mid-1 if x>arr[mid] else mid+1  #Using ternary operator i.e if x>mid-1 then index becomes
mid-1 otherwise index becomes mid+1.
   if x>arr[mid]:
      right=mid-1
elif x<arr[mid]:
      left=mid+1
delarr[0]#deleting the element at index 0.
arr.insert(index,x) # inserting the deleted element that is already stored as x in the index found using
the binary search operation.
print(*arr)
```

3.5.1. Discussion of a Proposed New Sort Algorithm

The main purpose of a proposed methodology is to study and analyze the performance of the proposed sorting algorithm with existing sorting algorithms like Improved cyclic sort and Modified cycle sort. And to demonstrate performance with existing modified cycle sorting with 75% efficiency.

The proposed algorithm accepts n total inputs in a random ordered array. Then it selects the first element as x and removes it from the list L, and it subtracts each element in an array with the x value to find a new array. The algorithm starts by applying the first step of finding the maximum value in the new array. The maximum value index is found. The divide and conquer strategy is used in the new array with the index of the maximum value during the second phase. Consequently, the list is divided into two toe lists, L1 and L2. A binary search technique is used in the third phase to sort the array into ascending or descending order.

4 Performance Analysis of the New Sorting Algorithm

4.1. Best Case

The new sorting algorithm selects the sequence of sorted elements by traversing from the given input list in forward and backward direction. If the items are already in sorted manner, it will choose them in continuous order in the first iteration. As a result, the temporal recurrence relation in this scenario will be considered as T(m) = O(m), where m denotes the number of elements.

4.2. Terrible Case

Sorting all the elements will take 2n steps if the algorithm adds only one element at a time to the sorted list for each forward and backward direction. Due to the mean and merge strategy, even though components like 4 and 3 occur various times in the given array list, they appear once when separated into two sub lists. As a result, T(n) = O is the worst-case temporal complexity (n2).

4.3. Normal Case

The given input list is a normal sequence of numbers, then the temporal recurrence relation is T(n) = O (nlogn).

5 Results

Sorting algorithms are generated by software programmers to sort any unordered lists in a given array, independent of their original order or list length. If the given lists are sorted, then Searching becomes easier. The proposed sorting algorithm is written in the Python programming language. The tests were performed on a Windows operating system with a 2.4 GHz Intel Core i7 processor and 8 GB hard disk of 1076 MHz DDR3 memory. The experimental test was carried out using empirical data (integer numbers) produced at random using standard Python library methods. Table 2 presents the results in order of increasing complexity.

Table 2: Performance of the Algorithm

Input Size (n)	**Execution Time (milliseconds)**		
	Best Case	**Normal Case**	**Terrible Case**
100	0.05	0.33	0.34
1000	0.68	5.45	55.24
10000	10	85.01	242.76

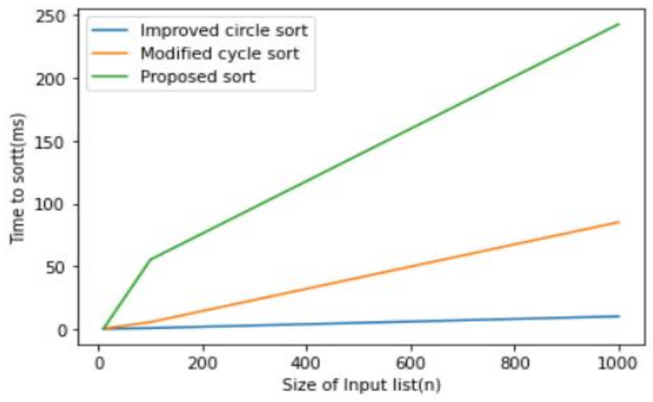
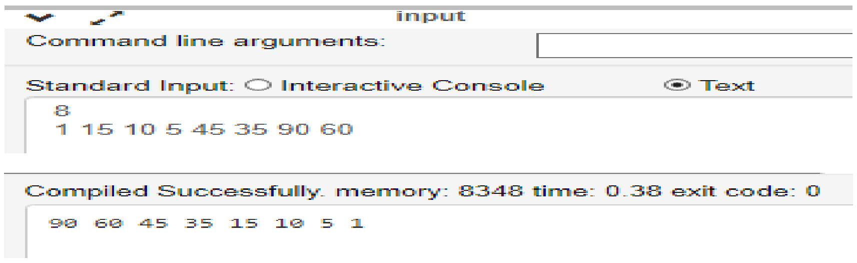


Figure 2: Performance of the Algorithm



```
Command line arguments:

Standard Input: ○ Interactive Console        ◉ Text
  8
  1 15 10 5 45 35 90 60

Compiled Successfully. memory: 8348 time: 0.38 exit code: 0
  90 60 45 35 15 10 5 1
```

Figure 3: Results of the Proposed Sorting Algorithm

For the effectiveness and precision, we casted the programme 10 times for each algorithm with same input and we took the mean value of time. And our time measurement unit is milli second. This section depicts a comparison study utilising a variety of measures such as time complexity is presented in table 3.

Table 3: Comparative Performance Analysis of Proposed Sorting Algorithm with Existing

Input Size (n)	**Execution Time (milliseconds)**		
	Improved Circle Sort	**Modified Cycle Sort**	**Proposed Sort**
100	.39	1.505	0.33
1000	3.89	213.1	5.45
10000	55.84	351.90	85.01

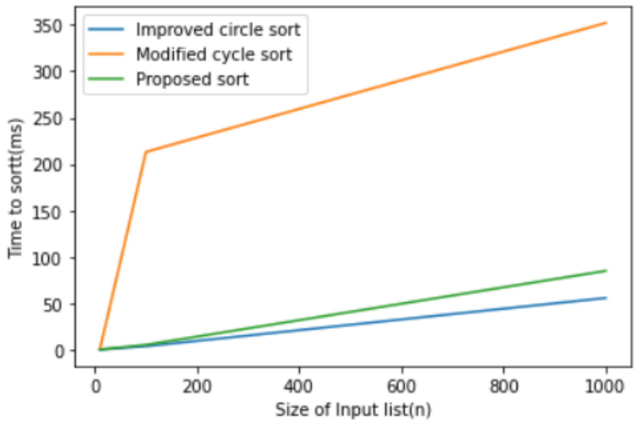


Figure 4: Comparative Performance Analysis of Proposed Sorting Algorithm with Existing

6 Conclusion and Future Works

With a comparison of existing sorting algorithms, such as Improved Circle sort and Modified Cycle sort, the proposed sorting algorithm has effectively demonstrated its performance efficiency in time complexity. In a smaller data set of random number sequences, the suggested sorting algorithm is 88 %accurate, whereas in a bigger data set of random number sequences, it is 76% accurate. As a result, on average, the suggested sorting algorithm outperforms the existing Sorting algorithm by 75%.

This paper discusses a comparison between the new suggested proposed algorithm and Improved circle sort and Modified cycle sort. Its categories the performance of these algorithms for the same given set of elements (100, 1000, 10000). For small input the performance of the comparative techniques is all nearest, but modified sort takes lesser performance. For the large input Improved cyclic sort is the fastest and the Modified cycle sort is the lowest performance. The proposed sort is having lesser performance with the improved cyclic sort and having much higher performance than Modified cycle sort. In this initial step of this research, the proposed is proving 75% efficiency than the modified cyclic sort. Hence the objective is proved. Further step for future work; in the future we will improve our algorithms to optimize software's in searching method and retrieve data and too much efficient than Improved cyclic sort.

The algorithm is defined in the Python programming language for ease of implementation. Because Python lacks pointers, each recursive call on the sub-list requires the creation of new left and right sub-lists. We can't alter the same list again since, unlike an array, this will reorder the integers because Python lists are mutable. Simply rearrange the pointers so that they point to the second bigger element in the ordered list to combine the sections. We can also provide the linked lists after link realignment instead of building new sub-lists for recursion. As a result, future work will focus on eliminating some intermediary steps and implementing the method in C or C++ using pointers, with the results compared.

References

[1] Alnihoud, J., & Mansi, R. (2010). An enhancement of major sorting algorithms. *International Arab Journal of Information Technology, 7*(1), 55– 62.

[2] Bahig, H.M. (2019). Complexity analysis and performance of double hashing sort algorithm. *Journal of the Egyptian Mathematical Society*, *27*(1), 1-12.

[3] Budhani, S.K., Tewari, N., Joshi, M., & Kala, K. (2021). Quicker Sort Algorithm: Upgrading time complexity of Quick Sort to Linear Logarithmic. *In IEEE 2nd International Conference on Computation, Automation and Knowledge Management (ICCAKM)*, 342-345.

[4] Rahma, A.M.S., Abid, A.P.D.M.A., & Ali, K. (2018). A new Sort Algorithm for Multi Core parallel Computers. *Iraqi Journal of Information Technology. V*, *9*(1), 65-81.

[5] Lovrencic, A. (2020). An Improved Circle Sort Algorithm. *Acta Electrotechnica et Informatica, 20*(3), 17–23.

[6] Rana, M.S., Hossin, M.A., Mahmud, S.H., Jahan, H., Satter, A.Z., & Bhuiyan, T. (2019). Min Finder: A new approach in sorting algorithm. *Procedia Computer Science*, *154*, 130-136.

[7] Mohammed, A.S., Amrahov, Ş.E., & Çelebi, F.V. (2017). Bidirectional Conditional Insertion Sort algorithm; An efficient progress on the classical insertion sort. *Future Generation Computer Systems*, *71*, 102-112.

[8] Moreno, J., Rodriguez, D., Nebro, A.J., & Lozano, J.A. (2020). Merge nondominated sorting algorithm for many-objective optimization. *IEEE Transactions on Cybernetics*, *51*(12), 6154-6164.

[9] Muthusundari, S., & Suresh, R.M. (2014). An enhanced D-Shuffle Sorting algorithm for secured encryption message to represent in tree. *In IEEE International Conference on Advanced Communications, Control and Computing Technologies*, 1583-1588.
[10] Muthusundari, S., & Suresh, R.M. (2013). An Enhanced Bubble Sorting algorithm with Divide and Conquer Approach Leads to the Construction of Balanced Search Tree. *CiiT International Journal of Artificial Intelligent systems and Machine Learning*, *5*(4), 189-192.
[11] Omar, Y.M., Osama, H., & Badr, A. (2017). Double hashing sort algorithm. *Computing in Science & Engineering*, *19*(2), 63-69.
[12] Bekker, J., & Trankle, N. (2021). Shadow Sort–A Hybrid Non-Dominated Sorting Algorithm, 1-8.
[13] Almathami, H.K., Alrafiee, M.A., Vlahu-Gjorgievska, E., & Win, K.T. (2019). An Analytical Approach to Using and Implementing Beacons: Opportunities and Challenges. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 10(1), 58-74.
[14] Turzo, N.A., Sarker, P., Kumar, B., Ghose, J., & Chakraborty, A. (2020). Defining a Modified Cycle Sort Algorithm and Parallel Critique with other Sorting Algorithms. *GRD Journals-Global Research and Development Journal for Engineering, 5*(5), 1-8.
[15] Wickremesinghe, R., Arge, L., Chase, J.S., & Vitter, J.S. (2002). Efficient sorting using registers and caches. *Journal of Experimental Algorithmics (JEA)*, *7*, 9.
[16] Gugale, Y. (2018). Super sort sorting algorithm. *In IEEE 3rd International Conference for Convergence in Technology (I2CT)*, 1-5.

Authors Biography

Dr.S. Muthusundari, M.Sc., M.Phil., M.E, Ph. D is Associate Professor in the Department of Computer Science and Engineering, since June 2007. She obtained her M. Sc (CS) from Madura College, M. Phil (CS) from Mother Teresa University and M.E(CSE) from Sathyabama University. She received her Ph. D (Data Structures) at Sathyabama University, Chennai. She has been in the teaching profession for the past 28 years and has handled both UG and PG programmes. Her areas of interest include Computer Networks, Data Structures, Computer Architectures, Operating Systems, Software Engineering, Digital Signal Processing and Object-Oriented Analysis and Design. She has published 50 papers in various International Journals and Conferences. She has attended many workshops & FDPs sponsored by AICTE related to his area of interest. She has produced 12 M. Phil students. She has written text book on “Ad Hoc and Sensor Networks”. She has organized a six-days AICTE-ISTE sponsored Induction/Refresher programme in the month of May 2018. She received Rs 20,000/- from TNSCST for the two days national seminar on Machine Learning with Hands on Workshop Using Weka Tool and R Programming on June 2019. She has guided projects for many undergraduate and post graduate students. She is a life time member of IAENG &amp; ACM. She received Rs 15,000/- from TNSCST & NCSCT New Delhi for Organizing one day Popularization of Science Program on “E-Waste Management” in the year 2020.

Mrs.L. Sherin Beevi is an Assistant Professor in the Department of Computer Science and Engineering at R.M.D Engineering College, Chennai. She obtained her Bachelor of Technology degree in Information Technology in 2007 from St. Xavier's Catholic College of Engineering, Nagercoil, and her Master of Technology degree in Computer Science and Engineering in 2009 from Francis Xavier College of Engineering, Tirunelveli, Anna University, Chennai. She is currently pursuing her PhD in Blockchain with Quantum Computing at Anna University, Chennai. She has been in the teaching profession for the past 13 years in all over the India, during which time she has taught both undergraduate and postgraduate programs. Her areas of interest include computer networks, blockchain,

quantum computing, data science, and deep learning. She has 10 publications in various fields and has presented her research at various conferences and seminars.
Orcid: 0000-0002-1180-7550

Mr.G. Shankar, M.E, (Ph. D)., is working as Assistant Professor in Department of Computer Science and Engineering at R.M.D. Engineering College, Chennai. He is dynamic, enthusiastic, strong passion towards teaching, dedicated and hard working towards his work. He obtained B.E., degree from Anna University, M.E., from Annamalai University, and pursuing Ph. D in Annamalai University in the area of Image and video Processing with Deep learning and machine learning. He worked as a lecturer in the department of Computer Engineering in P.T. Lee CNPT College from 2008 to 2010. He is worked as assistant professor in the department of computer science and engineering in various engineering colleges from Tamil Nadu, Andhra Pradesh and Telangana in the period of 2012- till date. His areas of interests include Deep Learning, Computer Vision, Image Processing, Machine learning. He has published many papers in International, national Conferences and Indexed Journals. He has attended many workshops & FDPs sponsored by AICTE related to her area of interest. He received Discipline star award from NPTEL in the year of 2021. Orcid: 0000-0002-3660-1778

Ms.U. Archana, M.E, is working as Assistant Professor in Department of Computer Science and Design at R.M.K. Engineering College, Chennai. She is passion towards teaching, dedicated and hard working towards h work. She completed B.E. from R.M.K College of Engineering and Technology and obtained her degree from Anna University Chennai. She completed M.E., from R.M.D Engineering College and obtained her degree from Anna University Chennai. Her areas of include Opinion mining, Recommendation System. He has published many papers in International, national Conferences. She has attended many workshops & FDPs sponsored by AICTE related to her area of interest.

Dr.G. Nirmala, MCA, M.E, Ph.D., is a university rank holder in UG and PG, enthusiastic, strong passion towards teaching, dedicated and hardworking, Assistant Professor in Department of Computer Science and Engineering. she obtained BSc., degree from Alagappa Government Arts College under Madurai Kamaraj University, MCA from Alagappa University, M.E CSE from Anna University and Ph. D - faculty of Information and communication Engineering from Anna University in the field of Deep learning. She worked as a lecturer in the department of Computer Science and Engineering in Alagappa Chettiar college of Engineering and Technology, Karaikudi from 2003 to 2006. She has been in the teaching and learning profession, counselling, mentoring students for the past 20 years and has handled both for UG and PG programs. Her areas of interest include Deep Learning, Computer Vision, Image Processing, system software, Computer Architecture, Operating Systems, Object Oriented Programming and Middle ware Technologies. She has published 10 papers in International, national Conferences and Indexed Journals. She has attended many workshops & FDPs sponsored by AICTE related to her area of interest. She has completed mission 10X high impact teaching skills by Wipro and campus connect certified by Infosys. She received best paper award in 2014.She has completed 9 nptel courses. She has completed 5 modules in NITTR. She is a life time member of ISTE. Orcid-ID: 0000-0002-3684-8965