A Threshold-Based Sorting Algorithm for Dense Wireless Communication Networks

Shahriar Shirvani Moghaddam [1] and Kiaksar Shirvani Moghaddam, [2]

[1]Shahid Rajaee Teacher Training University
[2]Affiliation not available

February 10, 2022

Abstract

This letter proposes a time-efficient algorithm applicable for time-varying wireless networks, which sorts a predefined number of elements in a large data set that are larger or smaller than the other or located between two parts. Based on the mean and standard deviation, a theoretical analysis for Gaussian, uniform, negative exponential, Rayleigh, and unknown distributions is presented, which finds exact and approximate thresholds. Then, the theoretical and numerical analyses show the superiority of the proposed algorithm to the well-known Merge, Quick, and K-S mean-based sorting algorithms in terms of the time complexity, the running time, and the similarity measure.

A Threshold-Based Sorting Algorithm for Dense Wireless Communication Networks

Shahriar Shirvani Moghaddam, *Senior Member, IEEE,* and Kiaksar Shirvani Moghaddam, *Student Member, IEEE*

***Abstract*—This letter proposes a time-efficient algorithm applicable for time-varying wireless networks, which sorts a predefined number of elements in a large data set that are larger or smaller than the other or located between two parts. Based on the mean and standard deviation, a theoretical analysis for Gaussian, uniform, negative exponential, Rayleigh, and unknown distributions is presented, which finds exact and approximate thresholds. Then, the theoretical and numerical analyses show the superiority of the proposed algorithm to the well-known Merge, Quick, and K-S mean-based sorting algorithms in terms of the time complexity, the running time, and the similarity measure.**

***Index Terms*—Sorting, Divide-and-conquer, Threshold, Merge, Quick, K-S mean-based.**

I. Introduction

COMMUNICATION and computer systems and networks deal with many cases which require rearrangement of data either in descending or ascending order [1]. This operation is called sorting. The purpose of an efficient sorting algorithm is to reduce the computational complexity and time taken to perform the comparison, swapping, and assignment operations [2]. Most sorting algorithms do not guarantee the sorting up to the end of the sorting process, and also, most of them cannot be controlled applying some limitations and modifications [2], [3]. A major category is based on the divide-and-conquer method that first divides the initial problem into several subproblems, the same in goals with a smaller scale [3]. In this category, Quick, Merge, and mean-based sorts have been widely used in data processing [4]–[6] and this letter.

Communication systems and networks need sorting quantities, such as distance, time, velocity, delay, channel gain, path-loss, signal to interference-plus-noise ratio (SINR), sum-rate, outage and error probabilities, and the number of free channels as positive (or negative in dB) numbers.

Essential sorting applications include fast search in searching, speed-up data retrieval and comparison, and do the clustering [6]–[8] and speed it up. Therefore, there is a need to propose new approaches to get the best sorting results in new applications [9]–[12]. In dense cellular homogeneous or heterogeneous networks and ultra-dense networks (UDNs), the data sets need to be sorted for selecting the best base-stations (BSs) or relays, femtocells, and access points (APs) based on SINR, sum-rate, outage probability, consumed power, number of free channels, and so on. Hence, it is necessary to find faster the best solution, when there exist some criteria in the optimization problems mentioned above.

In large networks based on massive multi-input multi-output (massive-MIMO) and UDNs, as well as heterogeneous networks consisting of small cells [13]–[16], dense sensor networks with a large number of sensors, and internet of things (IoT) applications, the number of nodes is huge. It is necessary to select a limited number of them (above or below a threshold level) and then sort them. On the other hand, the variations in many communication channels are high. If the sorting process is time-consuming, the information obtained in one moment is not helpful for the next moment. Hence, instead of sorting the whole data, a selected part above or below a threshold level would be sorted. Also, in some cases, due to a large number of data and a time-consuming sorting process, it is necessary to use the data sorted gradually. Moreover, in some applications, the nodes that do the processing have to consume little power, or their processing capabilities and memory size are limited, like edge processing [16] which offers low power consumption, low delay, high security, and so on for sensor nodes and passive radio frequency identifications (RFIDs). Furthermore, many applications do not need to sort all data, such as clustering, selecting the appropriate relays from several relays, and selecting K nearest neighbors (KNN) in a data set.

As the main goal of this investigation, instead of sorting whole big size data, and then removing some of them that do not satisfy the threshold level, we first obtain the threshold level by calculating the mean value and standard deviation of the data, extracting the required amount of data, and finally just sort this part of data. Briefly, we offer:

- Mathematical formulas based on the mean and standard deviation for known and unknown distributions;
- A threshold-based sorting algorithm for refined data above/below a threshold level or between two thresholds;
- Software realization of the proposed algorithm in $C\#$;
- Extracting the values of the correction factor for different percentages of data with unknown distribution;
- A time complexity analysis;
- A numerical analysis to compare the conventional and threshold-based versions of the Merge, Quick, and K-S mean-based algorithms.

In the following, Section II derives thresholds for known and unknown distributions, theoretically. The proposed threshold-based sort is discussed in Section III. Section IV demonstrates the time complexity analysis. Numerical analyses are presented in Section V. Finally, Section VI concludes this letter.

S. Shirvani Moghaddam is with the Faculty of Electrical Engineering, Shahid Rajaee Teacher Training University (SRTTU), 1678815811, Tehran, Iran (e-mail: sh_shirvani@sru.ac.ir).

K. Shirvani Moghaddam is a B.Sc. student at the School of Computer Engineering, Iran University of Science and Technology (IUST), 1684613114, Narmak, Tehran, Iran (e-mail: kiaksar_shirvani@comp.iust.ac.ir).

II. Threshold Level for Different Distributions

Assuming that the probability distribution of the data is known, the number of data above a threshold level, n_1, can be determined using equation (1) in terms of the threshold level, Th_X, the total number of data, n_t, and the probability density function (pdf), $f_X(x)$.

$$n_1 = n_t \int_{Th_X}^{\infty} f_X(x)\, dx \tag{1}$$

For four specific distributions widely used in wireless communications, such as Gaussian, uniform, negative exponential, and Rayleigh, the exact values of the threshold in terms of n_1, n_t, mean value, $\bar{X}$, and standard deviation, σ_X are as follows.

In the Gaussian distribution with the following pdf [17]

$$f_{XG}(x) = \frac{1}{\sqrt{2\pi}\sigma_{XG}} e^{-(\frac{x-\bar{X}_G}{\sqrt{2}\sigma_{XG}})^2} \tag{2}$$

Using equation (1) and inserting equation (2) in it, we have

$$n_1 = n_t Q(\frac{Th_G - \bar{X}_G}{\sigma_{XG}}) \tag{3}$$

$$Th_G = \bar{X}_G + Q^{-1}(\frac{n_1}{n_t})\sigma_{XG} \tag{4}$$

where $Q(u)$ is given as

$$Q(u) = \int_u^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx \tag{5}$$

and $Q^{-1}(.)$ is the inverse function of Q-function.

In the uniform distribution, with pdf as (6) [17], we have

$$f_{XU}(x) = \begin{cases} \frac{1}{2\sqrt{3}\sigma_{XU}}, & \bar{X}_U - \sqrt{3}\sigma_{XU} \le x \le \bar{X}_U + \sqrt{3}\sigma_{XU} \\ 0, & Otherwise \end{cases} \tag{6}$$

$$n_1 = n_t \frac{\bar{X}_U + \sqrt{3}\sigma_{XU} - Th_U}{2\sqrt{3}\sigma_{XU}} \tag{7}$$

$$Th_U = \bar{X}_U + \sqrt{3}(1 - \frac{2n_1}{n_t})\sigma_{XU} \tag{8}$$

For negative exponential distribution as (9) [17], it is given

$$f_{XNE}(x) = \begin{cases} \frac{e^{-\frac{x}{\bar{X}_{NE}}}}{\bar{X}_{NE}}, & x \ge 0\ ,\ \sigma_{XNE} = \sqrt{3}\bar{X}_{NE} \\ 0, & Otherwise \end{cases} \tag{9}$$

$$n_1 = n_t e^{-\frac{Th_{NE}}{\bar{X}_{NE}}} \tag{10}$$

$$Th_{NE} = \bar{X}_{NE} + \frac{\ln(\frac{n_t}{n_1}) - 1}{\sqrt{3}}\sigma_{XNE} \tag{11}$$

For Rayleigh distribution, as (12) [17], it is concluded

$$f_{XR}(x) = \begin{cases} \frac{\pi x}{2\bar{X}_R^2} e^{-\pi(\frac{x}{2\bar{X}_R})^2}, & x \ge 0\ ,\ \sigma_{XR} = \sqrt{\frac{4-\pi}{\pi}}\bar{X}_R \\ 0, & Otherwise \end{cases} \tag{12}$$

$$n_1 = n_t e^{-\frac{\pi Th_R^2}{4\bar{X}_R^2}} \tag{13}$$

$$Th_R = \bar{X}_R + (\sqrt{\frac{2}{4-\pi}\ln\left(\frac{n_t}{n_1}\right)} - \sqrt{\frac{\pi}{4-\pi}})\sigma_{XR} \tag{14}$$

Resulting from the threshold values mentioned above for four popular distributions, equations (4), (8), (11), and (14), generally, the threshold value for all distributions is as

$$Th_X = \bar{X} + m_e \sigma_X \tag{15}$$

m_e is a constant value that depends on the pdf, the total number of data, n_t, and the number of data above the threshold level, n_1. Because the threshold value is directly related to both the mean and variance values, decrease (increase) the threshold level, increases (decreases) n_1.

Sometimes high SNR solutions decrease the feasible area for optimization problems, and we are forced to find sub-optimum solutions because of time complexity and other considerations and constraints. For example, in ultra-reliable low-latency communications, vehicle-to-vehicle communications, massive machine-type communications, handover process, and offloading, the optimization problems limit the solutions between two thresholds [16], [17]. To estimate the number of data between two thresholds, n_3, we obtain equations (16), (17), (18), and (19), for Gaussian, uniform, negative exponential, and Rayleigh distributions, respectively.

$$n_3 = n_t \frac{Th_{U2} - Th_{U1}}{2\sqrt{3}\sigma_{XU}} \tag{16}$$

$$n_3 = n_t[Q\left(\frac{Th_{G1} - \bar{X}_G}{\sigma_{XG}}\right) - Q(\frac{Th_{G2} - \bar{X}_G}{\sigma_{XG}})] \tag{17}$$

$$n_3 = n_t[e^{-\frac{Th_{NE1}}{\bar{X}_{NE}}} - e^{-\frac{ThN_{E2}}{\bar{X}_{NE}}}] \tag{18}$$

$$n_3 = n_t[e^{-\frac{\pi Th_{R1}^2}{4\bar{X}_R^2}} - e^{-\frac{\pi Th_{R2}^2}{4\bar{X}_R^2}}] \tag{19}$$

In practice, the statistical distribution of data is not available or may be different from the popular ones. Therefore, the threshold level should be determined approximately based on the mean and standard deviation. The Chebyshev inequalities are always valid for each random variable X as [17]

$$\begin{cases} P\left(\left|X - \bar{X}\right| < m\sigma_X\right) \ge 1 - \frac{1}{m^2} \\ P\left(\left|X - \bar{X}\right| > m\sigma_X\right) \le \frac{1}{m^2} \end{cases} \tag{20}$$

In other words,

$$\begin{cases} P\left(\bar{X} - m\sigma_X < X < \bar{X}\right) + \\ P(\bar{X} < X < \bar{X} + m\sigma_X) \ge 1 - \frac{1}{m^2} \\ P\left(X > \bar{X} + m\sigma_X\right) + P(X < \bar{X} - m\sigma_X) \le \frac{1}{m^2} \end{cases} \tag{21}$$

If the first and second terms on the left side of these inequalities are approximately equal, we have:

$$\begin{cases} P\left(\bar{X} - m\sigma_X < X < \bar{X}\right) = \frac{1}{2} - \frac{1}{2m^2} + \varepsilon \\ P\left(\bar{X} < X < \bar{X} + m\sigma_X\right) = \frac{1}{2} - \frac{1}{2m^2} + \varepsilon \\ P\left(X > \bar{X} + m\sigma_X\right) = \frac{1}{2m^2} - \varepsilon \\ P\left(X < \bar{X} - m\sigma_X\right) = \frac{1}{2m^2} - \varepsilon \end{cases} \tag{22}$$

ε is a non-negative value in the range of $\left[0, \frac{1}{2m^2}\right]$, which depends on the statistical distribution of data.

If the total number of elements in the data set is n_t, the num-

ber of data greater than the threshold level of $\bar{X} + m\sigma_X, n_1$, is

$$n_1 = n_t P\left(X > \bar{X} + m\sigma_X\right) = (\frac{1}{2m^2} - \varepsilon)n_t \tag{23}$$

and n_2, the number of data smaller than the threshold level of $\bar{X} - m\sigma_X$, is

$$n_2 = n_t P\left(X < \bar{X} - m\sigma_X\right) = (\frac{1}{2m^2} - \varepsilon)n_t \tag{24}$$

and the data between two thresholds, n_3, is given as

$$n_3 = n_t P\left(\bar{X} - m\sigma_X < X < \bar{X} + m\sigma_X\right) = (1 - \frac{1}{m^2} + 2\varepsilon)n_t \tag{25}$$

In the formulas mentioned above, m is given as

$$m = \sqrt{\frac{n_t}{2n_i + 2\varepsilon n_t}} = \sqrt{\frac{n_t}{n_t - n_3 + 2\varepsilon n_t}},\ i = 1, 2 \tag{26}$$

By removing ϵ and considering a correction factor, c_f,

$$m = c_f \sqrt{\frac{n_t}{2n_i}} = c_f \sqrt{\frac{n_t}{n_t - n_3}},\ i = 1, 2 \tag{27}$$

It is guaranteed that m obtained from (27) will provide the least number of data above or below a threshold level or the number of data between two threshold levels. For large m, ε tends to zero, and $P\left(\left|X - \bar{X}\right| < \infty\right) = 1$, resulting in n_1 and n_2 being zero and n_3 being the total number of data.

III. The Proposed Threshold-Based Algorithm

The main goal is to sort the K number of a big size, $n_t (\gg K)$ unsorted data such that it can get K from the highest or lowest values sorted, and also sorted values between two thresholds in a shorter time. By selecting the proper m and finding the appropriate threshold level, we make two independent subarrays, one approximately in the length of the desired data, and then do the sorting process. Also, we obtain the required number of larger or smaller data sorted from the total data without having any sort on the whole data. To reach the desired number of data, do the following steps:

- Calculate the mean value, $\bar{X}$, and standard deviation, σ_X;
- Calculate exact m using (4), (8), (11), and (14), for known, or approximate m that satisfies (27) for unknown distributions;
- Sort the refined data above the threshold level, $\bar{X} + m\sigma_X$ or below the threshold level, $\bar{X} - m\sigma_X$, or between two threshold levels, $[\bar{X} - m\sigma_X\ , \bar{X} + m\sigma_X]$.

The proposed threshold-based sorting algorithm is such that the data are sorted step by step, and can be stopped whenever it reaches the required number of sorted data. Also, after sorting data larger (or smaller) than the threshold level, data smaller (or larger) than the threshold level can be sorted, consequently. Algorithm 1 shows the pseudo-code of the threshold-based sorting algorithm. The main functions are Threshold, Sort, and Partition.

The proposed sorting algorithm can be considered as a general framework for Quick, Heap, and K-S mean-based algorithms. When the threshold value is selected randomly in the data list, it is the same as Quick, while selecting the mean value as the threshold level introduces a mean-based sorting algorithm. Also, when the threshold value is chosen as the maximum value of the data list, it is like the Heap algorithm. In these special cases, data variance does not affect the threshold value.

Algorithm 1 Pseudo-code of the proposed threshold-based sorting algorithm.

Variables: arr: Array of data; **arr.Length:** Length of the array; n_1**:** Number of desired data above the threshold level; **mean:** The mean value of the array; **const:** Exact constant value; **std:** Standard deviation; **threshold:** Calculated threshold level; i**:** Temporary variable to be used in the For loop; j**:** Temporary variable for swapping two array elements.

Function `Sort(`arr, n_1`)`**:**

```
threshold ⟵ mean + const × std
j ⟵ 0
for i ← 0 to arr.Length − 1 do
    if arr[i] > threshold then
        swap arr[i] and arr[j]
        j ⟵ j + 1
    end
end
Sort arr from 0 to n1 using any sorting algorithm
return void
```

End Function

IV. Time Complexity Analysis

In the following analyses, Conv. stands for sorting the total data using a conventional sorting algorithm and then selecting the predefined number of sorted data. Conv.ansm is the case that the sorted data can be gradually extracted to reach the required number of the sorted data. Known and Unknown methods are the proposed threshold-based versions of the sorting algorithm. If the data's pdf is known, the exact threshold value while in the unknown scenario, the approximate threshold value is extracted.

The time complexity order (TCO) of Quick, Merge, and K-S mean-based sorting algorithms, in the average-case [6], [7], is

$$TCO_{Conv.} = O(n_t log n_t) \tag{28}$$

For evaluating the mean value and standard deviation of the data, the complexity order is linear, $O(n_t)$. Besides, for making one subarray in the length of n_1, whose elements are greater than the exact (or approximate) threshold level, it is also in a linear order of $O(n_t)$. Moreover, according to the results of [6], [7], the average-case complexity order for sorting the selected subarray in known and unknown pdf versions, respectively in the length of n_1 and $n_1^{'} (> n_1)$ are $O(n_1 log n_1)$ and $O(n_1^{'} log n_1^{'})$. Hence, in the mentioned methods, the total complexity order is as (29), (30).

$$TCO_{known} = O(2n_t) + O(n_1 log n_1) \tag{29}$$

$$TCO_{unknown} = O(2n_t) + O(n_1^{'} log n_1^{'}) \tag{30}$$

Assuming that $n_t = 2^{M_t}$ and $n_1 = 2^{M_1} n_t$, which M_t and M_1 are respectively the number of equal-length divisions to reach one-element subarrays and subarrays of length n_1, total

complexity order of Quick and K-S mean-based algorithms, to get a subarray of length n_1, and sort this subarray, is

$$TCO_{Conv.ansm} = \sum_{i=0}^{M_1} O(\frac{n_t}{2^i}) + \sum_{i=1}^{M_t - M_1} O(n_1) \quad (31)$$

By some mathematical work, the total complexity order is

$$TCO_{Conv.ansm} = O(2n_t - n_1) + (M_t - M_1)O(n_1) \quad (32)$$

For large data sets, we have

$$TCO_{Conv.ansm} = O(2n_t) + O(n_1 log n_1) \quad (33)$$

In realistic cases, the assumptions mentioned above may deviate. Therefore, the complexity order will be increased for K-S mean-based sorting algorithm and much higher for Quick-sort. Hence, (33) is modified by considering $n_1^{''}(> n_1^{'})$ and $n_1^{'''}(> n_1^{''})$ instead of n_1 in K-S mean-based and Quick sorting algorithms, respectively. Due to the Merge-sort nature, it is as complex in Conv.ansm as it is in Conv.

V. Numerical Analyses and Comparisons

Using simulations in $C\#$, we compare the results of the proposed threshold-based algorithm with the conventional algorithms, such as Merge, Quick, and K-S mean-based sorts that are the best ones for large data sets. All simulations are run on a 64bit system, Intel core ™ i7-9750H with 16GB RAM, 12MB Cache, 2.6GHz Max. CPU speed. We evaluate the required running time for sorting a selected part of an extensive data set, including 1000000 data for four different random data, such as Gaussian, uniform integer, negative exponential, and Rayleigh. Moreover, we evaluate the average normalized similarity measure (ANSM) in different sampling times. Considering the sorted and unsorted data values in location ith at jth sampling time as $X_e(i,j)$ and $X_m(i,j)$ respectively, the non-negative metric, ANSM, is given as

$$ANSM(j) = \frac{(\sum_{i=1}^{K} X_e(i,j).X_m(i,j) - K\bar{X}_e.\bar{X}_m)^2}{(\sum_{i=1}^{K} X_e^2(i,j) - K\bar{X}_e^2).(\sum_{i=1}^{K} X_m^2(i,j) - K\bar{X}_m^2)} \quad (34)$$

n_t and K, are the lengths of the primary data and selected part of the data to be sorted, respectively. When ANSM reaches 1, the predefined number of primary data is sorted.

To answer questions, "How much is the approximate threshold value close to the exact value?" and "What is the proper value for the correction factor?", four different cases, Gaussian, uniform integer, negative exponential, and Rayleigh data, are examined for two scenarios, known and unknown pdfs.

According to the simulation results in Fig. 1-a, the correction factor is chosen as the minimum value that satisfies all pdfs. Simulations show that the correction factor does not depend on the mean, standard deviation, and data size. It means that the results of this work are valid for different data sizes and various data statistics. Fig. 1-b shows the difference between the number of data selected from the exact and approximate threshold values.

Figs. 2, 3, compare the proposed algorithm to the conventional one in Merge and Quick sorting algorithms in terms

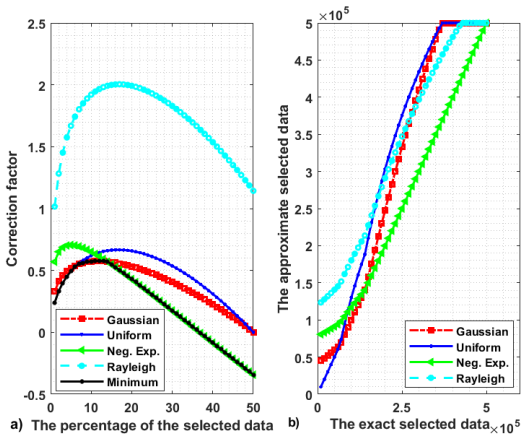

Fig. 1. a) The correction factor for different percentages of the selected data, b) The exact and approximate selected data.

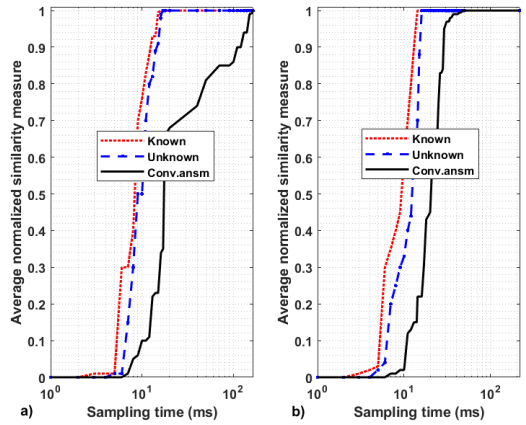

Fig. 2. ANSM versus the sampling time for sorting 5% of Uniform data with a size of 1000000, a) Merge-sort, b) Quick-sort.

of ANSM for a predefined number of selected data, i.e., 5% for symmetric (uniform) and non-symmetric (Rayleigh) data with a size of 1000000. Also, a similar trend much better than Quick-sort and Merge-sort can be achieved for the K-S mean-based sorting algorithm. Considering 100 simulation runs, Table I shows the average running time required for sorting the mentioned scenarios. A similar trend for sorting two other data distributions, Gaussian and Negative exponential, is valid.

Briefly, the achieved results are as follows:

- An approximate time can be detected for the Quick and K-S mean-based algorithms capable of gradual sorting. Still, for many sorting algorithms such as Merge, there is no guarantee that a certain amount of data is sorted by the end of the sorting process. It means that the time required to sort a predefined number of data in the later ones is much greater than the results reported in this work.
- When the data's probability distribution function is unknown, we achieve higher running time and the number of swaps compared to the case that pdf is known, both smaller than that for the conventional algorithms.
- As shown in Figs. 2, 3, over a longer time, in both

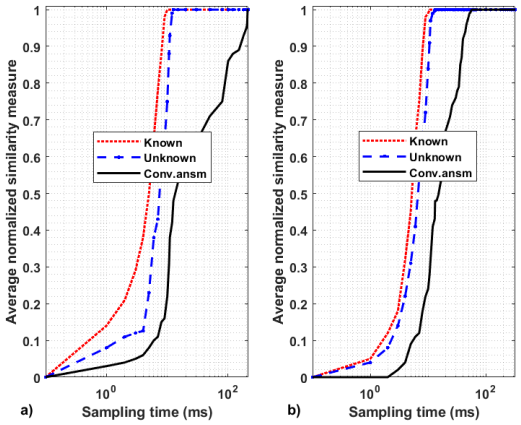

Fig. 3. ANSM versus the sampling time for sorting 5% of Rayleigh data with a size of 1000000, a) Merge-sort, b) Quick-sort.

TABLE I
THE REQUIRED RUNNING TIME FOR SORTING 5% OF A DATA SET WITH A SIZE OF 1000000.

Algorithm	**Method**	**Required running time (ms)**			
		Uniform	Gaussian	Rayleigh	Neg. Exp.
Merge	Known	15.9	20.1	10.6	16.2
	Unknown	17.1	23.2	12.3	18.4
	Conv.ansm	161.1	207.2	208.8.	256.7
	Conv.	163	208.2	213	258
Quick	Known	14.1	17.16	10.4	15
	Unknown	16.8	20.2	13.1	17.8
	Conv.ansm	47.2	56.1	69.5	63
	Conv.	214	264.6	321.2	273.1
Mean-based	Known	10.6	13.2	8.6	11.5
	Unknown	11.4	15.1	9.7	12.9
	Conv.ansm	21	31.1	27.3	29.2
	Conv.	95	102.1	121.6	184.3

known and unknown methods, a larger ANSM value than Conv.ansm and Conv. is obtained. It is helpful for wireless applications with rapid channel changes or in cases where processing power is limited and the time required to get the sorted data is short.

- In terms of the minimum time elapsed for sorting in each of the algorithms, Table I shows the priority, known, unknown, Conv.ansm and Conv scenarios, respectively. Because, the number of divisions and swaps, and the average distance between the swapped elements are reduced.
- In the lower percentage of data, the effectiveness of the proposed method to the conventional one is more.
- In the Quick algorithm, if in one of the two known and unknown methods the pivot is almost equal to the threshold, their performance results are identical in terms of ANSM, the number of swaps, and runtime, but due to the randomness of pivot value in this algorithm, the probability of such a case is minimal. If the data set is non-integer uniform, this probability is $\frac{1}{n_t}$.
- Based on the elapsed time for sorting a predefined number of data and ANSM, the superiority of K-S mean-based to Quick and Merge algorithms in [6] is also valid for Conv., Conv.ansm, known, and unknown methods.

VI. CONCLUSION

In this letter, we proposed a time-efficient and applicable sorting algorithm that gradually sorts a predefined number of the primary data based on a threshold level. Besides, we theoretically extracted these thresholds for four widely used random distributions in wireless communications, i.e., Gaussian, uniform, negative exponential, and Rayleigh. Moreover, we derived an equation to find the approximate threshold level for unknown random distributions. Finally, we analyzed and compared the proposed threshold-based algorithm to Quick, Merge, and K-S mean-based sorting algorithms by simulations for large data sets in $C\#$. Theoretical and numerical results showed the effectiveness of the proposed algorithm in terms of the time complexity order, required running time, and the similarity measure. To find the K largest (or smallest) values, this method offers a faster solution than finding K maximums (or minimums) in the popular manners.

REFERENCES

[1] S. Mishra, S. Saha, and S. Mondal, "Divide and conquer based non-dominated sorting for parallel environment," presented at *the IEEE Congr. Evol. Computation*, Vancouver, Canada, July 24–29, 2016.

[2] I. Hayaran and P. Khanna, "Couple sort," presented at *the 4th Int. Conf. Parallel, Distrib., Grid Comput.*, Waknaghat, India, Dec. 22–24, 2016.

[3] M. Yan, W. Shang, and M. Zhang, "The analysis of coordinate-recorded Merge-sort based on the divide-and-conquer method," presented at *the 15th Int. Conf. Comput. Inf. Sci.*, Okayama, Japan, June 26–29, 2016.

[4] W. Xiang, "Analysis of the time complexity of Quick sort algorithm," Presented at *the Int. Conf. Inf. Manage., Innov. Manage. Ind. Eng.*, Shenzhen, China, 2011.

[5] Z. Marszalek, M. Wozniak, and D. PoBap, "Fully flexible parallel Merge sort for multicore architectures," *Complexity*, vol. 2018, pp. 1–19, 2018.

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

[7] S. Shirvani Moghaddam and K. Shirvani Moghaddam, "A general framework for sorting large data sets using independent subarrays of approximately equal length," *IEEE Access*, vol. 10, Early Access, 2022 (doi: 10.1109/ACCESS.2022.3145981).

[8] X. Huang, Z. Liu, and J. Li, "Toward trusted open data and services," *IET J. Eng.*, vol. 2019, no. 5, pp. 3455–3459, 2019.

[9] S. M. Cheema, N. Sarwar, and F. Yousa, "Contrastive analysis of Bubble Merge sort proposing hybrid approach," presented at *the 6th Int. Conf. Innov. Comput. Technol.*, Dublin, Ireland, Aug. 24–26, 2016.

[10] P. Gupta, "Conventional vs enhanced sorting algorithm: A review," *Int. J. Res. Scientific Innov.*, vol. 5, no. 1, pp. 120–127, 2018.

[11] L. Khreisat, "A survey of adaptive Quicksort algorithms," *Int. J. Comput. Sci. Secur.*, vol. 12, no. 1, pp. 1–10, 2018.

[12] S. Kaur Gill, V. Pal Singh, P. Sharma, and D. Kumar, "A comparative study of various sorting algorithms," *Int. J. Adv. Stud. of Scientific Res.*, Special Issue, pp. 367–372, 2018.

[13] S. Shirvani Moghaddam and K. Shirvani Moghaddam, "Efficient base-centric/user-centric clustering algorithm based on thresholding and sorting," presented at *the 14th IEEE Int. Conf. Innov. Inf. Technol.*, AlAin, UAE, Nov. 17–18, 2020.

[14] K. Shirvani Moghaddam and S. Shirvani Moghaddam, "A fast sub-optimum access point selection in ultra-dense networks," presented at *the 10th IEEE Int. Conf. Commun., Netw., Satell.*, Purwokerto, Indonesia, July 17–18, 2021.

[15] P. Hao, X. Yan, Y.-N. Ruyue, and Y. Yuan, "Ultra dense network: Challenges, enabling technologies and new trends," *China Commun.*, vol. 13, no. 2, pp. 30–40, Feb. 2016.

[16] W. Saad, M. Bennist, and M. Chen, "A vision of 6G wireless systems: Applications, trends, and open problems," *IEEE Netw.*, vol. 34, no. 3, pp. 134–142, May/June 2020.

[17] A. Papoulis and S. Unnikrishna Pillai, "Probability, Random Variables and Stochastic Processes," 4th Ed., McGraw-Hill, New York, USA, 2002.