Exp. Theo. NANOTECHNOLOGY 10 (2026) 811-820 https://doi.org/10.56053/10.2.811

Performance of sorting algorithms on data generated using various statistical distributions: A comprehensive study of sorting techniques

Saleh Salous[1,]*, *Rami M. Amro*[2]

[1]Computer Science Department, Palestine Technical University – Kadoorie, Main Branch, Tulkarm, Palestine
[2]Department of Applied sciences, Palestine Technical University – Kadoorie, Aroub Branch, Hebron, Palestine
*) Email: s.salous@ptuk.edu.ps

Received 20/1/2026, Received in revised form 9/4/2026, Accepted 10/4/2026, Published 15/4/2026

Sorting numbers, objects, or living things is of immense importance in most scientific fields. It is of great importance in physics, computer science, statistics, and nanoparticle dynamics. This study presents a comparative evaluation of seven sorting algorithms: Merge, Heap, Quick, Bubble, Insertion, Radix, and Shell Sort, using datasets generated from nine statistical distributions, namely Normal, Uniform, Poisson, Gumbel, Laplace, Weibull, Exponential, Chi-Square, and Binomial. Algorithm performance is analyzed based on the execution time required to sort datasets under varying statistical conditions. The results confirm that input distribution strongly affects sorting efficiency. Bubble Sort and Insertion Sort consistently exhibited the longest execution times, highlighting their poor scalability for large datasets. Merge Sort and Heap Sort show stable and robust performance across all distributions, making them dependable general-purpose choices. Quick Sort and Shell Sort are most effective for symmetric datasets but show moderate sensitivity to skewed and heavy-tailed inputs. Radix Sort remains highly efficient for integer data, though wider value ranges increase computational cost. Overall, the findings highlight that optimal sorting performance depends on matching the algorithm to the statistical characteristics of the input data.

Keywords: Sorting Algorithms; Statistical Distributions; Nanotechnology; Time Complexity.

1. INTRODUCTION

The process of sorting is an essential problem in all the significant sciences, which can be extended to include activities in daily life. For example, sorting can be an important research problem in the areas of nanoparticles and nanocomposites [1,2]. Moreover, sorting can be helpful in solving DNA fragment assembly problem [3,4], which is an essential algorithmic problem in bioinformatics. The process of sorting can be carried out mechanically, for instance, in the process of nanoparticle sorting, where the rate of charged particles of various masses changes due to an electric field. It can also be carried out through numerical methods using sorting algorithms with the assistance of computers or machines. The role of sorting algorithms in the field of computer science is significant, as they efficiently handle large volumes of data [5]. Some of the most common sorting algorithms include Merge Sort, Heap Sort, Quick Sort, Bubble Sort, Insertion Sort, Radix Sort, and Shell Sort [6]. These sorting algorithms have their own complexity in terms of time and space, which is an important factor for considering their use with various types of data. This becomes an important issue, especially for dealing with large volumes of data, as the selection of a suitable sorting algorithm should be made on the basis of expected processing time. The complexity of sorting, however, is not only associated with the algorithm used for sorting [7-9]. Rather, in most cases, the data can be distributed in accordance with a statistical distribution, such as normal, uniform, Poisson, Gumbel, Laplace, Weibull, exponential, chi-squared, or binomial [10-18]. Each of these statistical distributions is associated with some specific characteristics, which can affect the effectiveness of the sorting algorithm used for sorting the data. For instance, the implications of using a sorting algorithm for sorting data from a uniformly distributed source are different from those of using a sorting algorithm for sorting data from a Poisson distributed source, due to the frequency of the values in the data.

Although sorting algorithms have been extensively studied [19], most analyses still rely on simplified assumptions about input data, typically treating it as random or uniformly distributed [20]. In practice, however, real datasets often exhibit complex statistical characteristics [21] such as skewness, clustering, discreteness, and heavy-tailed behavior, all of which can significantly influence algorithm performance. Despite this reality, relatively limited attention has been given to understanding how different statistical distributions shape the behavior of sorting algorithms. This gap is particularly evident for algorithms like Quick Sort, Shell Sort, and Radix Sort, whose efficiency depends not only on input size but also on the structural properties of the data. A more nuanced, distribution-aware perspective is therefore necessary to better reflect real-world conditions. This is especially important in application domains such as bioinformatics and nanotechnology, where data rarely conforms to idealized models. Accordingly, advancing research toward integrating statistical data characteristics into algorithm evaluation and design represents a meaningful direction for improving both theoretical insight and practical efficiency. This paper focuses on the performance of ranking algorithms over datasets emanating from a range of statistical distributions. By performing a measure of the ranking algorithm's time complexity and that of the other types of distributions, we intend to contribute to the knowledge of what kind of sorting techniques are ideal for datasets possessing particular statistical features [22,23]. Reference [24] presents an important contribution focusing on the engineering aspects of radix sorting; however, the investigation does not address comprehensive comparisons across multiple sorting techniques or varied statistical distributions as this study. In this study, a wider perspective is adopted by analyzing seven fundamental sorting algorithms using datasets generated from nine different statistical distributions. These distributions are selected to reflect both idealized and highly irregular data patterns, allowing for a more realistic evaluation of algorithmic behavior.

While, previous studies have considered comparing these algorithms against each other [19] [21-24], our study complement and add comparisons among all these sorting algorithms using data drawn from nine distinct distributions commonly used in adverse scientific fields. In the next section we detail the methodology used for generating the data and processing it. In section 3, we detail the characteristic and discuss the space time complexity of the seven sorting algorithms, as well discussing the comparison results, we have obtained. In the conclusion section we reflect on the results of this study.

2. METHODOLOGY

To examine the impact of statistical distributions on sorting performance, a controlled computational framework is developed in which datasets of equal size are generated from nine distinct distributions and processed using multiple sorting algorithms under consistent conditions. Each dataset consisted of 100,000 positive integers, a size selected to balance computational feasibility with the ability to clearly differentiate between algorithms of varying time complexities, particularly quadratic $O(n^2)$ and O (n log n) time complexities. The datasets are generated to follow Normal, Uniform, Poisson, Gumbel, Laplace, Weibull, Exponential, Chi-Square, and Binomial distributions, providing a comprehensive representation of real-world data patterns, including symmetric, skewed, discrete, and heavy-tailed characteristics.

Seven widely used sorting algorithms - Merge Sort, Heap Sort, Quick Sort, Bubble Sort, Insertion Sort, Shell Sort, and Radix Sort - are chosen to reflect a wide range of algorithmic behaviors, particularly in terms of computational efficiency, memory requirements, and stability [25,26]. The seven algorithms are implemented in C++ without using any parallelization or multithreading, so all computations are carried out sequentially, one step at a time. Existing algorithms are executed under the same conditions on the same machine, one after another. Throughout the experiments, a stable power supply is maintained, and enough free memory is ensured to avoid interruptions. The system used has a single processor running at 2.30 GHz and 8 GB of virtual memory, providing sufficient capacity for smooth execution. This setup ensures that the results are fair and comparable, while also making sure each algorithm runs on a single core within the same environment.

An array of size n=100000 integers generated of one of the aforementioned distributions generated in Microsoft Excel software, are provided to each algorithm. Execution time, measured in seconds, is used as the primary performance metric, with each experiment repeated multiple times and averaged to reduce variability. The code for each algorithm is implanted in C++ programing language. The execution run time is calculated using the specific function (clock) divided by the clocks rate, that is

$$\boldsymbol{Run\ time} = \frac{\boldsymbol{number\ of\ clocks}}{\boldsymbol{clocks\ rate}} \tag{1}$$

Space complexity and stability are discussed alongside time performance to provide a more comprehensive evaluation. The results are organized and visualized (Fig. 1) to emphasize performance trends across distributions, as graphical representation facilitates clearer interpretation of differences, particularly the substantial gap between quadratic and log-linear algorithms.

3. RESULTS AND DISCUSSION

3.1. Sorting algorithms

Sorting is the process of arranging data in an ascending or descending order to facilitate searching and locating data more efficiently. Merge Sort is a divide-and-conquer algorithm that repeatedly divides the dataset into smaller sub-arrays until only one element is in all sub-arrays and then merges these arrays together in sorted order. Spatially, it is an efficient sorting algorithm with a time complexity of O (n $\log_2$ n) and suitable for large data sets however merge sorts also incur additional space during the merging so overall complexity falls to O(n). Merging sort maintains the stability of the sorting process (same order for equal items) [22,23]. Heap Sort transforms the entire dataset into a binary heap, which is a tree-based structure that allows access to the largest (or smallest) item in constant time. Similar to how Merge Sort connects continuous sequences of sorted numbers, Heap Sort keeps rearranging the heap by extracting the max (or min) repeatedly until a large sorted portion exists. Heap sort runs in O (n $\log_2$ n) time and does not need any additional memory (similar to merge sort). Nevertheless, heap sort is not a stable sorting algorithm [25,26].

The process of sorting is used to arrange the data in ascending or descending order to make the process of searching and retrieving data more efficient. Merge Sort is an algorithm used to sort data in ascending order by using the divide-and-conquer technique, which divides the data into smaller sub-arrays, making them contain only a single element, and then merges them in ascending order. From a spatial analysis perspective, it is an efficient sorting algorithm with a time complexity of O (n $\log_2$ n) for large data sets. The space complexity of the merge sort algorithm is O(n) due to the extra space used in the merging process. The process of sorting in the heap sort algorithm is to transform the entire data set into a binary heap, which is a tree-like data structure that allows constant-time access to the maximum (or minimum) element. Like the merge sort algorithm, which connects two or more sequences of consecutive numbers, the heap sort algorithm connects the maximum (or minimum) elements in the heap to form a significantly large sorted portion of the data. The time complexity of the heap sort algorithm is O (n $\log_2$ n) with no extra space, just like the merge sort algorithm. However, the stability of the sorting process is not maintained by the heap sort algorithm [27].

Radix Sort is a non-comparative sorting technique that sorts numbers digit by digit from the least significant digit to the most significant digit. A stable sorting technique like Counting Sort is used as a subroutine for sorting numbers digit by digit. The time complexity of Radix Sort is O(nk), where k represents the number of digits. The technique is highly efficient for sorting integers and strings; however, it is applicable only for specific data types and requires additional space [28,29].

Shell Sort is defined as a generalization of Insertion Sort that produces a sorted sequence by making comparisons and swaps at widely separated elements in order to increase the efficiency of sorting. With time, elements to be compared get closer and eventually result in a sorted array. The average time complexity of Shell Sort is between $O(n^{3/2})$ and $O(n^{7/6})$. Shell Sort is significantly faster compared to Bubble Sort and Insertion Sort for larger lists due to its average time complexity, as has been reported in the past [30].

3.2 Data statistical distributions

Distributions are beneficial mathematical tool in many scientific fields, physics, computer vision, climate phenomena, humanities, nano science, and many other scientific fields. Depending on the

application to which sorting algorithm employed, one or more of the aforementioned algorithms can be used. The nature of the particle distribution can be a determinant factor for the sorting speed: Assuming we have collected a data set composed of the diameter of nanoparticles, we can order or sort them in ascending or descending order based on their diameters' statistical distribution, albeit sorting them in the fastest way. Allowing the speed up for experiments and development of related nanotechnology devices. Here, we detail the characteristic of the distribution used in the comparison study:

1- Normal Distribution: Also called the Gaussian Distribution, it is a symmetrical distribution that is bell-shaped. The Normal Distribution is defined by the variance and the mean. It is used to define the way data points are clustered around the average value. The Normal Distribution is used in various fields such as physics, data analysis, machine learning, finance, quality control, nanostructures, signal processing, etc. [10,11].
2- Uniform Distribution: The uniform distribution is characterized by the equal probability of all values in the range of the distribution. The uniform distribution is commonly used in different fields and is often used in situations where all values are equally likely to occur. The distribution is used in different applications, including simulation and random sampling, decision theory, quality control, cryptography, and game theory, etc. [10].
3- Poisson Distribution: The Poisson distribution is a discrete probability distribution that, for a known average rate of occurrence, can be used to model the number of occurrences within a fixed time or space [11]. The Poisson distribution has various applications in biology, particularly in areas involving rare event occurrences, such as in population biology, cell biology, ecology, epidemiology, and dynamics [12].
4- Gumbel distribution: This is the standard form of the Gumbel family, which enjoys several beneficial characteristics that have made it a favorite in extreme value theory. The Gumbel distribution acts as a fundamental tool in extreme value theory for various domains, such as engineering, biotechnology, finance, meteorology, environmental science, and operations research [13].
5- Laplace distribution: This distribution has a more pronounced peak at the mean and heavier tails than the normal distribution. This distribution is commonly used in finance, signal processing, machine learning, and environmental science, where extreme values are possible with relatively low variability [16].
6- Weibull Distribution: This distribution is widely used as an analytical tool in engineering and sociology. Its shape parameter is used to describe different modes of failure. This distribution is applied in reliability engineering, survival analysis, wind energy, material testing, prediction of the life cycle of products, survival analysis in medicine, prediction of the speed of the wind for energy production, etc., to manage the uncertainties of the real world [14].
7- Exponential distribution: it is commonly used to model the time between random, independent events that occur at a constant rate. It is widely applied in areas such as reliability engineering, survival analysis, queuing theory, and risk assessment, as well as in fields like telecommunications, network traffic, biology, and economics. Its simplicity and memoryless property make it especially useful for systems where events happen continuously and independently [18].
8- Chi-Square distribution: it is widely used in statistics, particularly in hypothesis testing, analysis of variance, and categorical data analysis. It is commonly applied across fields such as biology, economics, engineering, and machine learning, mainly to compare observed data with expected values, making it an essential tool in both research and practice [21].
9- Binomial distribution: it is widely used in fields such as quality control, clinical research, finance, and marketing to model the probability of a specific number of successes in repeated binary trials.

It is especially useful for analyzing outcomes with two possibilities, such as success/failure or yes/no [15].

3.3 Main results

Figure 1 shows the results of the sorting algorithm performance tests across different data distributions. We notice based on the execution time that we have two separate groups in general: on one hand we have slower performing group composed of Bubble and Insertion sorting algorithms. On the other hand, we have the rest of the algorithms have run time that is about three orders of magnitudes less than the former group, which indicates a superiority in sorting for the fast group. Moreover, we notice that some of the sorting algorithms showed nearly no preference for nature of the data regardless of its original distribution. Merge, Heap, Bubble and Insertion have showed no preference, while, Shell and Quick have performed better and worse respectively for the normally distributed data. Radix, has showed nearly same computational time for all distributions except for Poissonian or Laplacian distributed data; it shows higher run time for the latter two distributions. Based on these findings we recommend avoid using Bubble or Insertion for sorting inters, Nevertheless, we recommend the usage of any other sorting algorithms as they have much smaller run time and nearly show no preference for distributive nature of the input datasets.

Figure 1 presents the measured execution times of seven sorting algorithms - Merge Sort, Heap Sort, Quick Sort, Shell Sort, Insertion Sort, Radix Sort, and Bubble Sort - have been applied to datasets of 100,000 positive integers generated from nine statistical distributions, including Normal, Uniform, Poisson, Gumbel, Laplace, Weibull, Exponential, Chi-Square, and Binomial. The objective is not only to report execution times but also to compare theoretical expectations with empirical behavior, particularly in terms of time complexity, space considerations, and sensitivity to data distribution.

From a theoretical perspective, Bubble Sort and Insertion Sort exhibit $O(n^2)$ time complexity with constant space and stability. For large inputs, such as n = 100,000, this implies an impractically high number of operations. The results strongly confirm this expectation, as both algorithms consistently demonstrate execution times several orders of magnitude higher than the more efficient methods, regardless of the underlying distribution. This indicates that favorable properties such as stability and in-place operation do not compensate for poor scalability in large datasets.

Merge Sort, with its O (n log n) time complexity and stable behavior, performs consistently among the fastest algorithms across all distributions. The results confirm its theoretical independence from data distribution, as its divide-and-conquer strategy remains unaffected by value patterns. Similarly, Heap Sort achieves comparable performance while maintaining constant space usage, validating its efficiency as an in-place alternative, although its lack of stability remains a theoretical limitation in applications requiring order preservation.

Quick Sort exhibits a more distribution-sensitive behavior. While it performs efficiently under symmetric distributions such as normal distributed data, slight performance degradation is observed for skewed or heavy-tailed distributions, reflecting the impact of partition imbalance. Although it has randomized

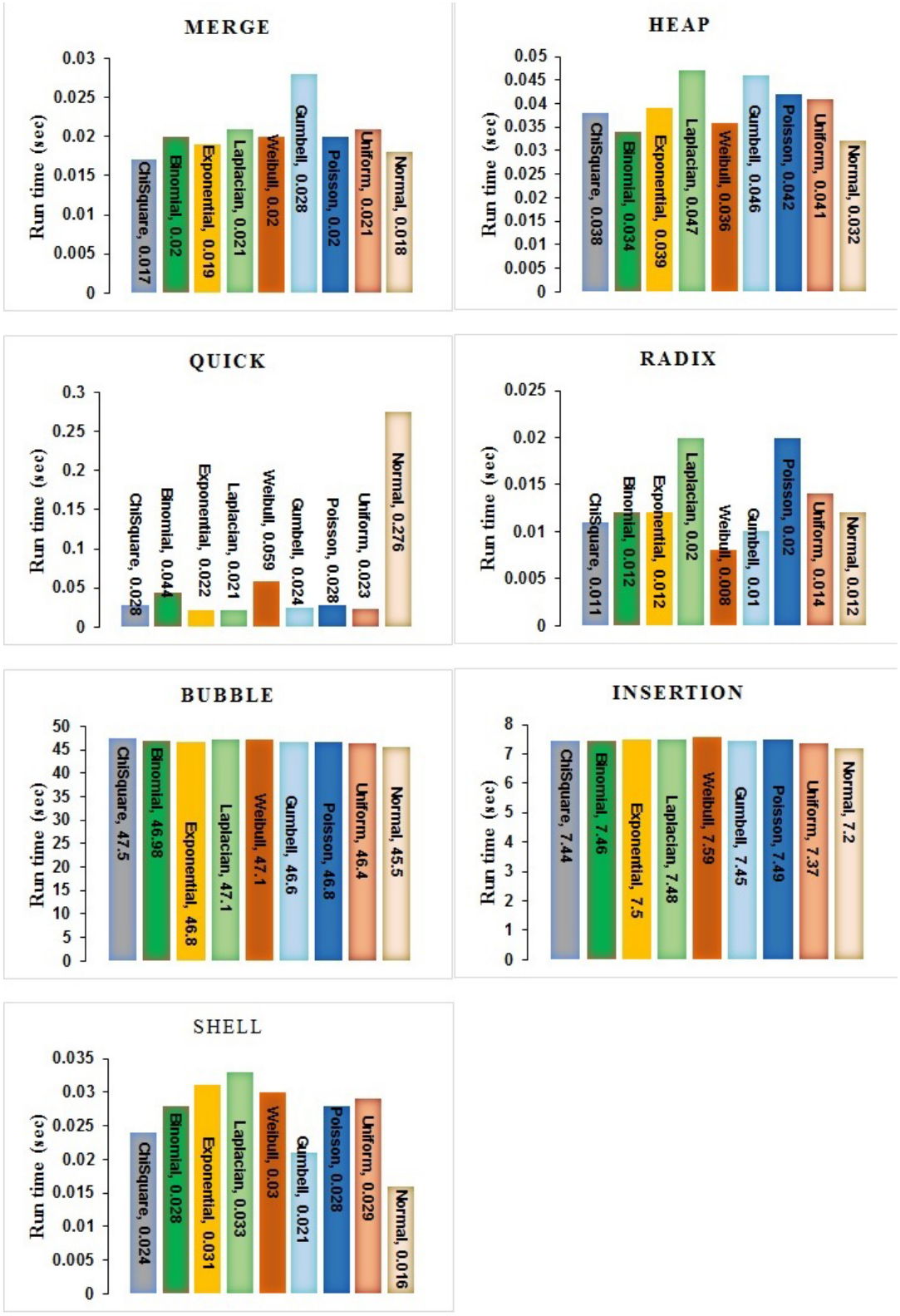

Figure 1 Time performance of seven sorting algorithms - Merge, Heap, Quick, Shell, Insertion, Radix, and Bubble Sort - across datasets generated from different statistical distributions. The time for each distribution is plotted as a bar and the value of bar-peak and name of the distribution is shown within the longitudinal bounds of the bar.

Pivot selection which prevents worst-case $O(n^2)$ behavior, the observed variations confirm that input distribution can influence practical performance.

Shell Sort demonstrates intermediate performance, consistently outperforming quadratic algorithms but remaining less efficient than log-linear methods. Its performance varies with data distribution, indicating that its gap-based mechanism is sensitive to data structure. Radix Sort, which operates with O(nk) complexity, shows excellent performance for datasets with limited value ranges, such as Binomial and Chi-Square distributions. However, its performance degrades for distributions with larger value ranges or extreme outliers, such as Poisson and Laplace, confirming its dependence on the number of digit passes.

Overall, the results confirm that asymptotic time complexity is a strong predictor of algorithm performance, with O (n log n) algorithms consistently outperforming $O(n^2)$ methods. However, distribution sensitivity remains an important factor for certain algorithms, particularly Quick Sort, Shell Sort, and Radix Sort, whereas Merge Sort and Heap Sort demonstrate robust, distribution-independent behavior. These results highlight the importance of selecting sorting algorithms based not only on theoretical efficiency but also on the statistical characteristics of input data and practical constraints such as memory usage and stability requirements.

It is important to note that the findings are subject to certain assumptions and limitations. The chosen dataset size represents medium-scale scenarios, and results may differ for significantly larger or smaller inputs. Additionally, the use of sequential execution without optimization reflects baseline performance rather than fully optimized implementations. Despite these limitations, the framework provides a consistent basis for analyzing how data distribution influences sorting behavior and offers practical insights for algorithm selection in real-world applications.

4. CONCLUSIONS

Sorting numerical data and structured entities is essential across many scientific disciplines. This study compared seven sorting algorithms using datasets from nine statistical distributions. Performance was assessed by measuring the execution time needed to sort positive integer arrays under identical conditions. Two clear performance groups were identified. Bubble Sort and Insertion Sort showed significantly higher runtimes, confirming poor scalability for large datasets. Merge, Heap, Quick, Shell, and Radix Sort performed substantially better overall. Merge and Heap Sort remained stable and largely unaffected by data distribution. Quick and Shell Sort were most efficient for symmetric distributions but degraded moderately for skewed data. Radix Sort was also stable; though wider value ranges slightly increase its runtime. The results confirmed that sorting efficiency strongly depended on the statistical properties of input data. For unknown distributions, Merge and Heap Sort were the most reliable choices. Overall, no single algorithm was universally optimal; selection must match data characteristics. When the data distribution was unknown, Merge Sort and Heap Sort provided the most reliable and robust performance. Quick Sort and Shell Sort were advantageous for near-symmetric datasets, while Radix Sort was highly effective for integer data with limited value ranges. These findings reinforced the strong link between data distribution and sorting efficiency.

Acknowledgements

The authors would like to thank the Palestine Technical University-Kadoorie for their financial support to conduct this research.

Conflicts of Interest

The authors declare that there is no conflict of interest of any kind.

References

[1] F. Nan, Z. Yan, Nano Letters, 18 (2018) 4500. https://doi.org/10.1021/acs.nanolett.8b01672
[2] N. M. Slaber, J. S. Kith. Experimental and Theoretical NANOTECHNOLOGY 9 (2025) 9 https://doi.org/10.56053/9.1.9
[3] H. Lee, H. Tang, Evolutionary Genomics: Statistical and Computational Methods, 1 (2012) 155 https://doi.org/10.1007/978-1-61779-582-4_5
[4] A. K. Shihab, A. H. Al-Mashhadani, R. M. Shalaby, Experimental and Theoretical NANOTECHNOLOGY 10 (2026) 207 https://doi.org/10.56053/10.S.207
[5] A. Shatnawi, Y. AlZahouri, M. A. Shehab, Y. Jararweh, M. Al-Ayyoub, Cluster Computing, 22 (2019) 819 https://doi.org/10.1007/s10586-018-2860-1
[6] M. Shabaz, A. Kumar, Journal of Computer Networks and Communications, 2019 (2019) 3027578 https://doi.org/10.1155/2019/3027578
[7] S. Abdel-Hafeez, A. Gordon-Ross, IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, 25 (2017) 1930 https://doi.org/10.1109/TVLSI.2017.2661746
[8] A. Zutshi, D. Goswami, International Journal of Information Management Data Insights, 1 (2021) https://doi.org/10.1016/j.jjimei.2021.100042
[9] J. Krithikadatta, Journal of Conservative Dentistry and Endodontics, 17 (2014) 96 https://doi.org/10.4103/0972-0707.124171
[10] H. Torabi, N. H. Montazeri, Communications in Statistics – Simulation and Computation, 43 (2014) 2551 https://doi.org/10.1080/03610918.2012.737491
[11] H. Fakhry, M. Rasheed, O. Salman, R. Ismail, Experimental and Theoretical NANOTECHNOLOGY 10 (2026) 81 https://doi.org/10.56053/10.1.81
[12] P. C. Consul, G. C. Jain, Technometrics, 15 (1973) 791 https://doi.org/10.1080/00401706.1973.10489112
[13] S. Nadarajah, S. Kotz, Mathematical Problems in Engineering, 4 (2004) 323 https://doi.org/10.1155/S1024123X04403068
[14] A. J. Hallinan Jr., Journal of Quality Technology, 25 (1993) 85 https://doi.org/10.1016/j.ifacol.2018.08.369
[15] P. M. Altham, Journal of the Royal Statistical Society Series C: Applied Statistics, 27 (1978) 162 https://doi.org/10.2307/2346943
[16] T. Eltoft, T. Kim, T. W. Lee, IEEE Signal Processing Letters, 13 (2006) 300 https://doi.org/10.1109/LSP.2006.870353
[17] B. Li, E. B. Martin, Computational Statistics & Data Analysis, 40 (2002) 21 https://doi.org/10.1016/S0167-9473(01)00097-4
[18] E. J. Gumbel, Journal of the American Statistical Association, 55 (1960) 698 https://doi.org/10.1080/01621459.1960.10483368

[19] T. Zhang, M. R. Azghadi, C. Lammie, A. Amirsoleimani, R. Genov, Journal of Neural Engineering, 20 (2023) 021001 https://doi.org/10.1088/1741-2552/acc7cc
[20] C. Heumann, M. Shalabh, Introduction to Statistics and Data Analysis, Springer, Switzerland (2016) https://doi.org/10.1007/978-3-319-46162-5
[21] H. Chen, I.C. Covert, S.M. Lundberg, S.I. Lee, Nature Machine Intelligence, 5 (2023) 590 https://doi.org/10.1038/s42256-023-00657-7
[22] S. Sepahyar, R. Vaziri, M. Rezaei, Comparing Four Important Sorting Algorithms Based on Their Time Complexity, ACAI '19: 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence, Sanya, China (2019) 320. https://doi.org/10.1145/3377713.3377808
[23] J. Lobo, S. Kuwelkar, 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), (2020) 110. https://doi.org/10.1109/ICESC48915.2020.9155623
[24] M. Axtmann, S. Witt, D. Ferizovic, P. Sanders, ACM Transactions on Parallel Computing, 9 (2022) 1 https://doi.org/10.1145/3505286
[25] M. A. Suchenek, The Computer Journal, 67 (2024) 812 https://doi.org/10.1093/comjnl/bxad007
[26] N. M. Aljulaidan, R. S. Almalki, S. B. Alqarni, Z. R. Alramadan, A. A. A. Ali, 2024 IEEE Open Conference of Electrical, Electronic, Information Sciences (eStream) 58 (2024) 1 https://doi.org/10.1109/eStream61684.2024.10542576
[27] K. A. Bakare, A. A. Okewu, Z. A. Abiola, A. Jaji, A. Muhammed, FUDMA Journal of Sciences 8 (2024) 1 https://doi.org/10.33003/fjs-2024-0805-2730
[28] I. M. Al-Amin, A. Okeyinka, A. Ibrahim, Journal of Science Innovation and Technology Research, 3 (2024) 61 http://doi.org/10.70382/ajsitr
[29] R. T. Bushman, T. M. Tebcherani, A. S. Yasin, arXiv preprint 99 (2024) 33 https://doi.org/10.48550/arXiv.2411.07526
[30] A. Aftab, M. A. Ali, A. Ghaffar, A. U. R. Shah, H. M. Ishfaq, M. Shujaat, International Journal of Computer Science and Information Security (IJCSIS), 19 (2021) 114 https://doi.org/10.5281/zenodo.4602255