GMR
The Original

DISTRIBUTION-SENSITIVE BENCHMARKING OF CLASSICAL AND ODD-EVEN PARITY BASED SORTING ALGORITHMS ACROSS STATISTICAL DATA MODELS

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

[1]Department of Applied Sciences, PalestineTechnical University – Kadoorie, Aroub Branch ,Hebron Palestine .
[2]Computer science Department, Palestine Technical University – Kadoorie, Tulkarm, Palestine.
***Corresponding Author:** Rami M. Amro, r.amro@ptuk.edu.ps

ABSTRACT
The sorting algorithms are essential entities in computer science, physics, biology and assist in a broad spectrum of data-intensive applications such as databases, search systems, operating platforms and big data analysis of biological data. In DNA sequence organization and genomic processing, where huge datasets have to be handled, efficient sorting is especially essential. Theoretical analysis often focused on time and space complexity of these algorithms, the statistical properties of input data can be very important in practice. This paper investigates how positive integers datasets generated from nine statistical distributions influence the sorting behavior of eight sorting algorithms: Merge Sort, Heap Sort, Quick Sort, Shell Sort, Bubble Sort, and Insertion Sort, and the non-comparative Radix Sort, as well as recently developed Odd-Even Parity Based sorting algorithm (OEPB). The primary criterion of evaluation is the execution time, and several trials are done to provide consistency and reliability of the results. The findings suggest that the performance of algorithms can be different in various data distributions. The specific focus is put on the OEPB algorithm that does exhibit a significant space efficiency with the gradually reduced memory consumption as the execution time proceeds. This feature renders it particularly appropriate in low memory resources settings. Whilst algorithms with complexity O ($n \log_2 n$) tend to be dominant, the findings emphasize the role of memory efficiency in design and sensitivity to input properties in determining the choice of sorting algorithms in practice.

KEYWORDS: Odd-Even Parity Based; Sorting; Distribution-Aware Algorithms; Sorting; DNA sequence; Genomic processing.

1. INTRODUCTION

Sorting numbers plays an essential role in computer science, physical sciences and many other scientific fields. It is not only pleasant but a necessity to manage data efficiently in the world overwhelmed with information. Although the core of sorting may seem straightforward, there are different issues that may demand distinct strategies. The various sorting algorithms have their advantages and disadvantages: some consume a large amount of memory, others are efficient with specific data distributions, and all have to grapple with computational complexity [1-4].

The analysis of sorting algorithms can't be based only on theoretical time complexity, since in practice the performance depends highly on the nature of the input data. In practical applications, the distribution of data, the repetition of data as well as structural properties are significant factors that are used to determine efficiency [5]. The statistical characteristics of the data, be it uniform, highly skewed or based on a normal, exponential, or Weibull distribution, or a chi-square distribution, can be of great importance in terms of execution time when handling large quantities of data [6]. Specifically, distorted or highly concentrated data can impair the efficiency of other algorithms that originally seem to be efficient in theory [10].

Bioinformatics is an interdisciplinary field that integrates computational methods, statistical modeling, and biological knowledge to analyze and interpret large-scale genomic data. In DNA sequence analysis, it plays a key role in managing the massive datasets generated by modern high-throughput sequencing technologies, where k-mer–based approaches are widely used for downstream analysis. A typical workflow begins with quality assessment and preprocessing of raw sequencing reads to ensure data reliability. In practice, DNA sequences such as ACTGAACT… are often transformed into numerical formats (e.g., A=0, C=1, G=2, T=3 or binary encodings) to facilitate efficient computation and algorithmic processing. This enables subsequent steps such as sequence alignment preprocessing, where reads are mapped and organized within computational pipelines. The processed data is then assembled into longer genomic sequences using de novo or reference-guided methods, followed by evaluation of assembly quality using standard metrics. Further stages include gene prediction and functional annotation to identify biologically meaningful regions, while visualization tools assist in interpreting genomic structures. Additional analyses, including alignment refinement, structural variation detection, and mutation profiling, contribute to deeper genomic interpretation. Overall, genomics has become central to modern biomedical research, supporting applications in precision medicine, cancer genomics, disease risk prediction,

evolutionary studies, and drug discovery. However, the growing scale of genomic data continues to demand efficient computational frameworks for effective biological insight extraction [11,12].

Most classical sorting algorithms, including Quick Sort, Merge sort, Heap sort, can be used on any numerical data, whether it is distributive or not. However, their implementation time, memory efficiency will suffer in case the data is not favorably distributed. This will make the reputation of the sorting algorithm, although it can be beautiful on the data created with a purpose of being sorted by the algorithm [13].

This paper uses Odd-Even Parity Based (OEPB), which is designed to address efficiency limitations associated with traditional sorting methods. The algorithm partitions the input into even and odd subsets and processes them based on data characteristics. A key characteristic of OEPB is its emphasis on memory efficiency, achieved through a progressive reduction of the active dataset during execution. Consequently, the size of the working set decreases over time, reducing memory requirements and making the method suitable for memory-constrained environments [14].

In contrast to traditional sorting algorithms that operate on a fixed dataset throughout execution, OEPB dynamically reduces the active workload, thereby lowering overall memory overhead. Based on these properties, the algorithm is particularly appropriate for large-scale and data-intensive applications, especially in domains such as genomic data analysis and high-dimensional scientific computing, where both computational efficiency and storage optimization are essential. OEPB is implemented in C++ and evaluated against widely used sorting algorithms, including merge sort, heap sort, quick sort, insertion sort, and radix sort. Theoretical analysis indicates that the algorithm exhibits a worst-case time complexity of O ($n \log_2 M$) and a linear space complexity of O(n)[14]. The detailed pseudocode of the proposed OEPB algorithm is provided in Appendix A.

Several previous studies have considered comparing the performance of few to many of the previous aforementioned sorting algorithms for various types of datasets drawn from various statistical distributions [15-17]. But none has considered all.

In this study, we benchmark all eight sorting algorithms against each other and against the recent OEPB for integer datasets of varying size generated most common, scientifically diverse data distributions. The article is arranged in the following order: in the next subsection we briefly describe the operational principle of each of these algorithms, their time and spatial complexity. In the subsection after, we detail the distributions used in this study. Later, in section 3, the methodology section, we discuss the experimentations condition. In section 4, the results and discussion section, we show and compare across these algorithms through figures reflecting the runtime for each algorithm for nine different statistical distributions. lastly, section 5, the conclusions, we summary our results and make further recommendations.

1.1 Related Work

Merge Sort works by recursively separating the data into smaller subarrays, sorting them, and finally combining them into a single ordered sequence. It has time complexity of O ($n \log_2 n$), but with extra memory of O(n). One of the major merits of Merge Sort is that it is stable [18,19]. Instead, Heap Sort takes the data and sorts it using a binary heap structure, and repeatedly removes the largest (or smallest) element to form an output that is sorted. It also has a runtime of O ($n \log_2 n$) and it is a place-value sort, but does not maintain relative order of identical elements [20,21].

Quick Sort is based on a divide and conquer approach, whereby a pivot element is chosen, the data is split in relation to the pivot, and the subarrays are sorted again using a recursive method. The average-case complexity of the algorithm is O($n \log_2 n$), but may drop to O(n^2) with poor choices of pivot. Nonetheless, it is generally considered to be very efficient in practice, even though it is not stable [22,23].

Bubble Sort is a simple algorithm that repeatedly makes comparisons between neighboring items and exchanges them in case of necessity. Its time complexity is O(n^2) and therefore it is impractical with large data sets, although it is easy to implement, making it practical only in educational settings [24,25].

In The Insertion Sort, the sorted array is constructed one element at a time, by inserting each element into the already sorted part of the array. Although it is quadratic, it can be used effectively with small or nearly sorted data sets and it is stable and in-place [26].

Shell sort can be considered an improvement of the Insertion sort, whereby elements are initially compared with large gaps which are narrowed down. This will enhance performance through fewer shifts to be taken. It has a median complexity of O($n^{3/2}$) to O($n^{7/6}$), and thus it is much more efficient than the straightforward quadratic algorithms on fairly large data sets [27,28]. Each of the above algorithms is a comparison based one.

Radix Sort is a non-comparison algorithm which works by sorting values digit by digit instead of making comparisons. It has a complexity of O(nk), making it suited to fixed-length integer data, but flexible and increased memory usage are limited by it [29,30].

The Odd-Even Parity Based (OEPB) is an algorithm that works with positive integers sorting them first into odd and even arrays. It uses repetitive subdivision by 2, eliminating elements which are equal to one in a systematic way. Then, it keeps dividing the remaining integers by 2 until the whole array is sorted. The worst time complexity of the algorithm is O (n log2 M) where the number of elements is n and the largest integer value is M. The space complexity begins with O(n), but decreases when the elements are eliminated, in particular, the ones not within a specific range [14].

1.2. Statistical Distributions Considered

Our preprocessing of data distribution plays an important role in evaluating algorithm efficiency because data distribution affects how values spread throughout different ranges and thus impacts algorithm computation in different conditions. The importance of considering data distribution in relation to algorithm computation increases even further in studying the efficiency of sorting algorithms working with massive datasets. In this research work, efficiency of the developed sorting algorithm is tested on sets of integers that come from nine different data distributions. The normal (Gaussian) distribution is a continuous, symmetric, bell-shaped distribution defined by the probability density

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \quad (1)$$

where μ denotes the mean and σ^2 represents the variance [7]. Most observations are concentrated around the mean, making this distribution widely applicable in fields such as physics, finance, and data analysis. In contrast, the continuous uniform distribution assumes equal probability across a bounded interval and is defined by

$$f(x)=\frac{1}{b-a}\quad, a \le x \le b \quad (2)$$

with mean $\frac{a+b}{2}$ and variance $\frac{(b-a)^2}{12}$, making it suitable for modeling unbiased or random processes [7]. The Poisson distribution is a discrete model used to describe the number of events occurring within a fixed interval, with probability mass function

$$P(X=k)=\frac{\lambda^k e^{-\lambda}}{k!} \quad (3)$$

where both the mean and variance are equal to λ [8]. For modeling extreme values, the Gumbel distribution is employed, defined by

$$f(x)=\frac{1}{\beta}\exp\left(-\frac{x-\mu}{\beta}-\exp\left(-\frac{x-\mu}{\beta}\right)\right) \quad (4)$$

with mean $\mu+\gamma\beta$ and variance $\frac{\pi^2}{6}\beta^2$, where γ is the Euler constant [9]. The Laplace distribution, which exhibits a sharper peak and heavier tails than the normal distribution, is given by $f(x)=\frac{1}{2b}e^{-\frac{x-\mu}{b}}$, with mean μ and variance $2b^2$, making it effective for modeling data with large deviations [8]. The Weibull distribution, commonly used in reliability and lifetime analysis, is defined by:

$$f(x)=\frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1}e^{-(x/\lambda)^k} \quad (5)$$

where its flexibility is controlled by the shape parameter k. The exponential distribution, a special case of the Weibull distribution, models the time between independent events and is expressed as $f(x)=\lambda e^{-\lambda x}$ for $x\ge 0$

(6)

with mean $\frac{1}{\lambda}$ and variance $\frac{1}{\lambda^2}$ [7]. The chi-square distribution, widely used in hypothesis testing and statistical inference, is defined by

$$f(x)=\frac{1}{e^{k/2}\,\Gamma\left(\frac{k}{2}\right)}x^{\frac{k}{2}-1}e^{-x/2} \quad (7)$$

with mean k and variance 2k [7]. Finally, the binomial distribution models the number of successes in a fixed number of independent trials and is defined by

$$P(X=k)=\binom{n}{k}p^k(1-p)^{n-k}$$

(8)

with mean *np* and variance *np(1-p)* [8]. Collectively, these distributions capture a diverse range of data behaviors, including symmetric, skewed, discrete, and heavy-tailed patterns, providing a comprehensive basis for evaluating sorting algorithms under realistic and varied conditions.

3. Experimental Methodology

All numerical experiments are run on the same computer. The computer specifics are: 2.3 GHz processor, and 8 GB of RAM. The data sets were generated in Microsoft Excel, with the proper function corresponding the specific statistical distribution. Three dataset sizes were used: 100,000, 150,000, and 200,000 positive integers. The C++ codes for the for the sorting algorithms were fed with the integer array. The execution time which is calculated by the clock function divided by the clock speed. The time was recorded in seconds. Each code was run alone sequentially on the machine. These experimental conditions were maintained for all results generated in this study, ensuring fair comparison between the eight sorting algorithms across all nine different statistical distributions. The runtime for each algorithm-distribution combination was calculated five times, and the average value was recorded and used in the results below. Insignificant difference in the five trials run time was observed.

4. RESULTS AND DISCUSSION

The results detailing the differences between these algorithms are presented in the figures 1-3. Figure 1, shows how the run time for various sorting algorithms compares for the nine statistical distributions for dataset of 100000 positive integers. The findings illustrated in Figure 1 indicate that the OEPB is always better than Bubble Sort in all the distributions and all sizes of the dataset, but still slower than Insertion Sort in terms of execution time. However, the OEPB also has a slower growth rate in execution time with a larger dataset, and is more pronounced with larger datasets, where the performance difference between OEPB and Bubble Sort is more pronounced. This

can be credited to the Odd-Even Parity Based comparison strategy that minimizes unnecessary comparisons and the unnecessary movement of elements.

Moreover, Figure 1 indicates that the sophisticated sorting algorithms perform much better than all the classical algorithms, and the running-time is usually two or three orders of magnitude less. Comparison of these more sophisticated algorithms with each other is largely independent of the data distribution, except that the Gumbel distribution has a crossover in terms of execution time between Bubble Sort and the OEPB. The results also demonstrate the powerful impact of the data distribution on sorting performance since significant differences in the execution time can be seen even when working with the datasets of the same size. Specifically, the OEPB performs the best on Weibull and Exponential distributions, whereas advanced algorithms like Merge Sort, Heap more favorable performance under the Gumbel distribution, with comparatively reduced efficiency for Uniform, Normal, Exponential, and Weibull datasets.

Figure 2 is an extension of Figure 1, in which the scaling performance of the execution time against the size of the dataset is shown with respect to 100,000, 150,000 and 200 000 elements, each with the identical statistical distributions. The results of the experiment are in line with theory. The sorting algorithms above such as Merge Sort, Heap Sort, Quick Sort, Radix Sort and Shell Sort have slow yet gradual increase of execution time since they are O ($n \log_2 n$) or almost linear and are effective in large data sorting. Quite on the contrary, Bubble Sort has a steep quadratic increase rate, which causes steep increase in computational time and limits its use on large inputs. The growth of Insertion Sort follows the same trend as $O(n^2)$ but is not that steep. The proposed OEPB has a medium scaling pattern, as its execution time grows steadily as compared to Bubble Sort. This difference becomes more pronounced with bigger input size particularly at higher volumes such as 200,000 elements. However, OEPB cannot be as quick as the work of the Insertion Sort and is even less productive than more advanced algorithms. There is also variation in performance due to underlying data distribution. With Weibull and Exponential distributions, OEPB is more efficient and more complicated distributions, such as Gumbel and Laplace, introduce a slight increment to the cost of computation, which can be explained by the extreme values.

In general, OEPB is never worse than Bubble Sort and is predictable and stable. It can be a potentially viable option in cases where either efficiency of resources or expediency of implementation is a crucial determinant due to its simplicity and low overhead.

Unlike traditional algorithms, which usually have a more or less constant allocation, OEPB reduces its memory usage throughout execution as Figure 3 illustrates. This trend is always the case in all the input sizes tested implying a more adaptive method of resource utilization. This behavior is particularly beneficial to use when working with large-scale DNA or genomic data, when there is limited memory. OEPB reduces the system resource load on the processor by reducing the memory requirements as time goes by. It is a flexible aspect that renders it highly suitable in conditions where the memory capacity is minimal. In general, the results highlight the strength of OEPB in situations involving the need to be both scalable as well as resource-efficient.

Comparisons of Merge, Heap, Quick, Insertion and Radix sort on datasets with 10,000 to 200,000 elements indicate the speed of this algorithm when dealing with redundant datasets. In some cases, it may run as much as 50 percent faster than others, when there are many repeated values. However, when datasets are not redundant, its performance is reduced since additional processes are required to move data between arrays. It is deterministic, so its execution time and memory consumption are predictable, and it is a good choice in applications such as embedded systems and real-time applications where reliability is important. It also has an advantage in dealing with redundant datasets and therefore can be a good choice to do database management and statistical computing; both of which frequently encounter duplicates. In conclusion, odd-even sorting algorithm puts a touch of class on the classical comparison-based sorting algorithms, particularly when data redundancy and memory constraints were the order of things [14]. Specifically, when it comes to large-scale processing of DNA sequences, where k-mer is generating extremely high-volume datasets that must be effectively organized. The datasets are also commonly characterized as redundant and non-uniform to the statistical behavior, and therefore, are computationally costly and enable the design of lightweight and adaptive sorting algorithms which can scale to biological size.

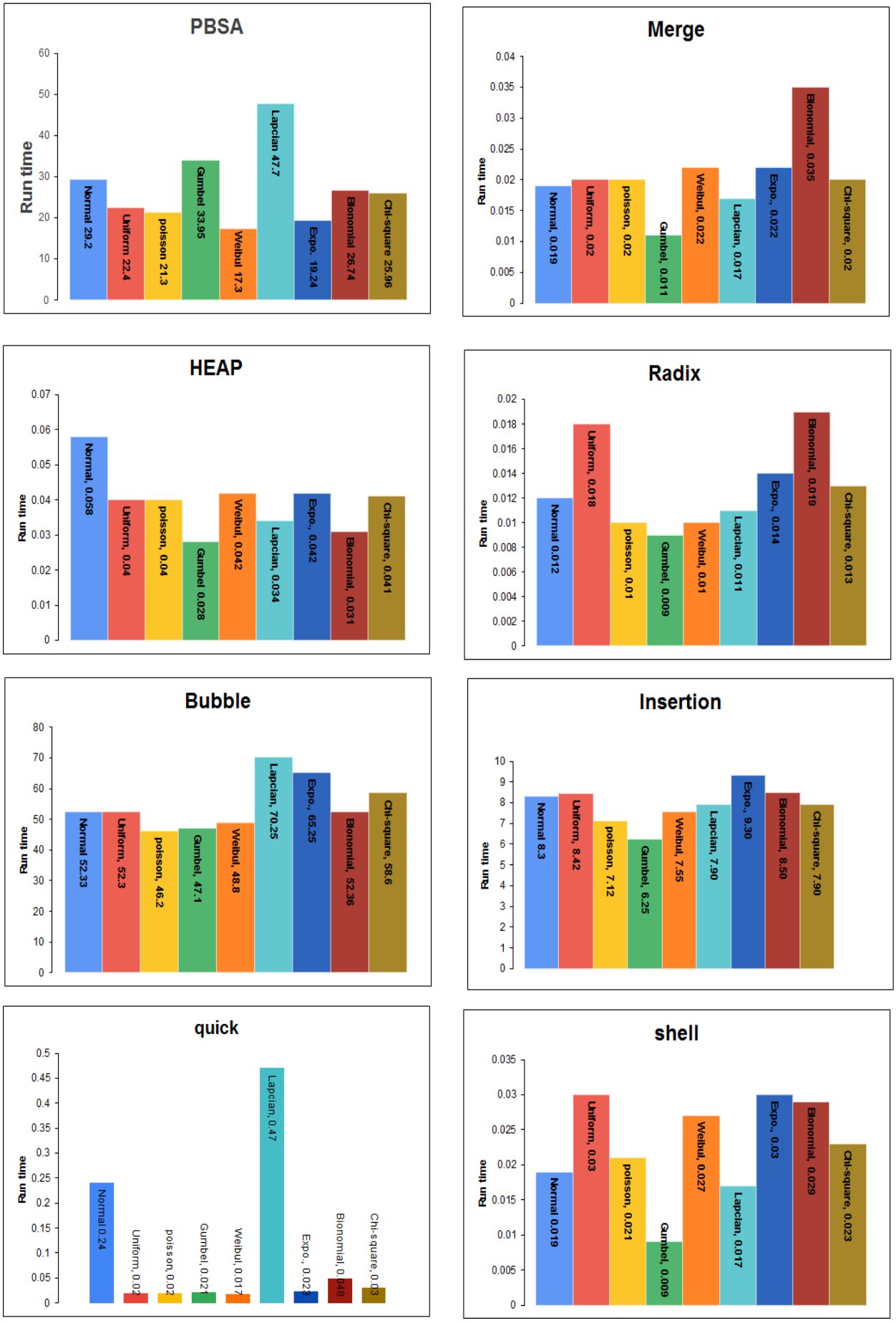

Figure 1 Evaluation of the execution time for the sorting algorithms reviewed in the main text. Each panel indicates the performance of the shown sorting algorithm for 9 different distributions for 100000 integers.

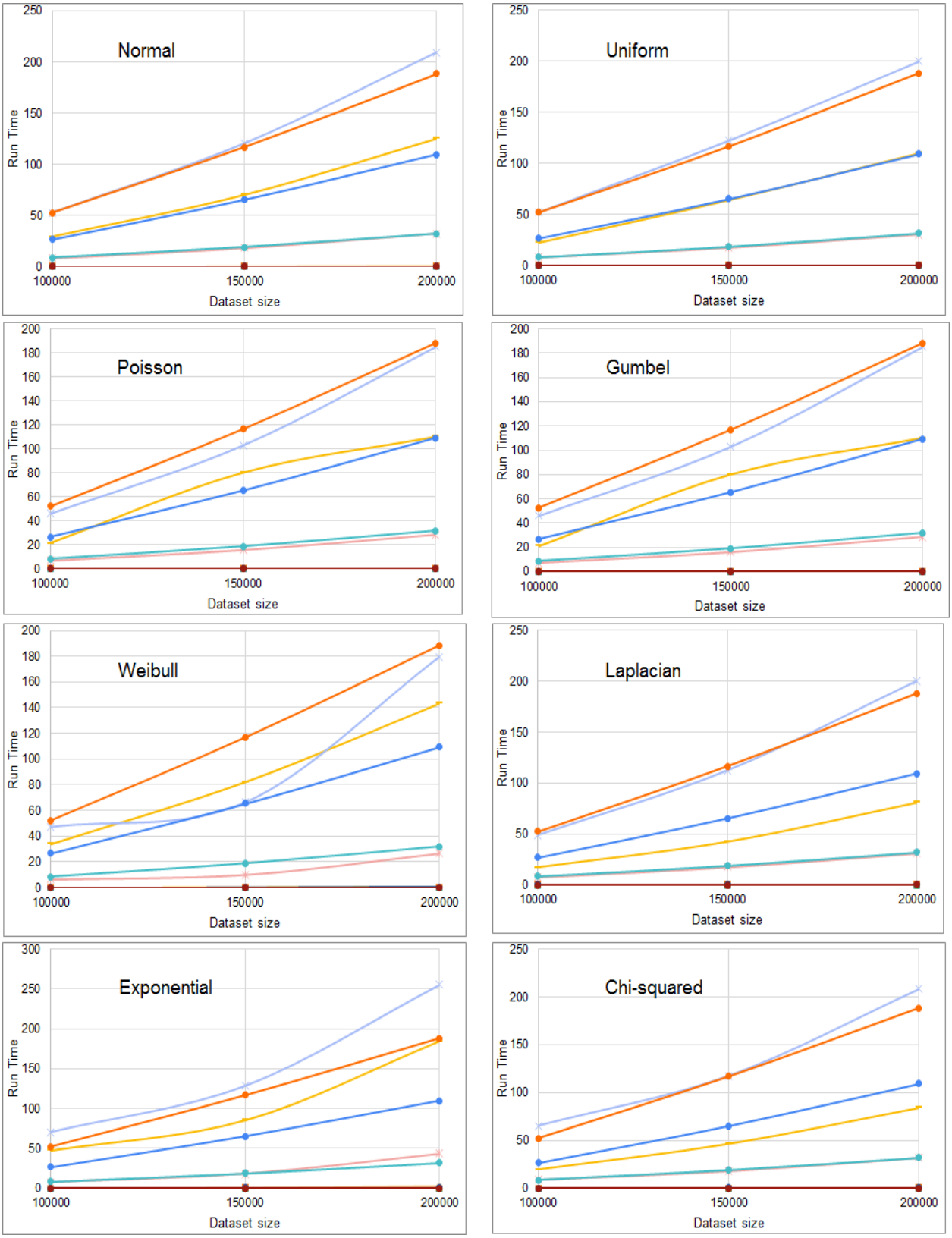

Figure2: Run time of different sorting algorithms versus the size of the data sets used (100 K, 150 K, and 200 K integer numbers) for various distributions used in the study.

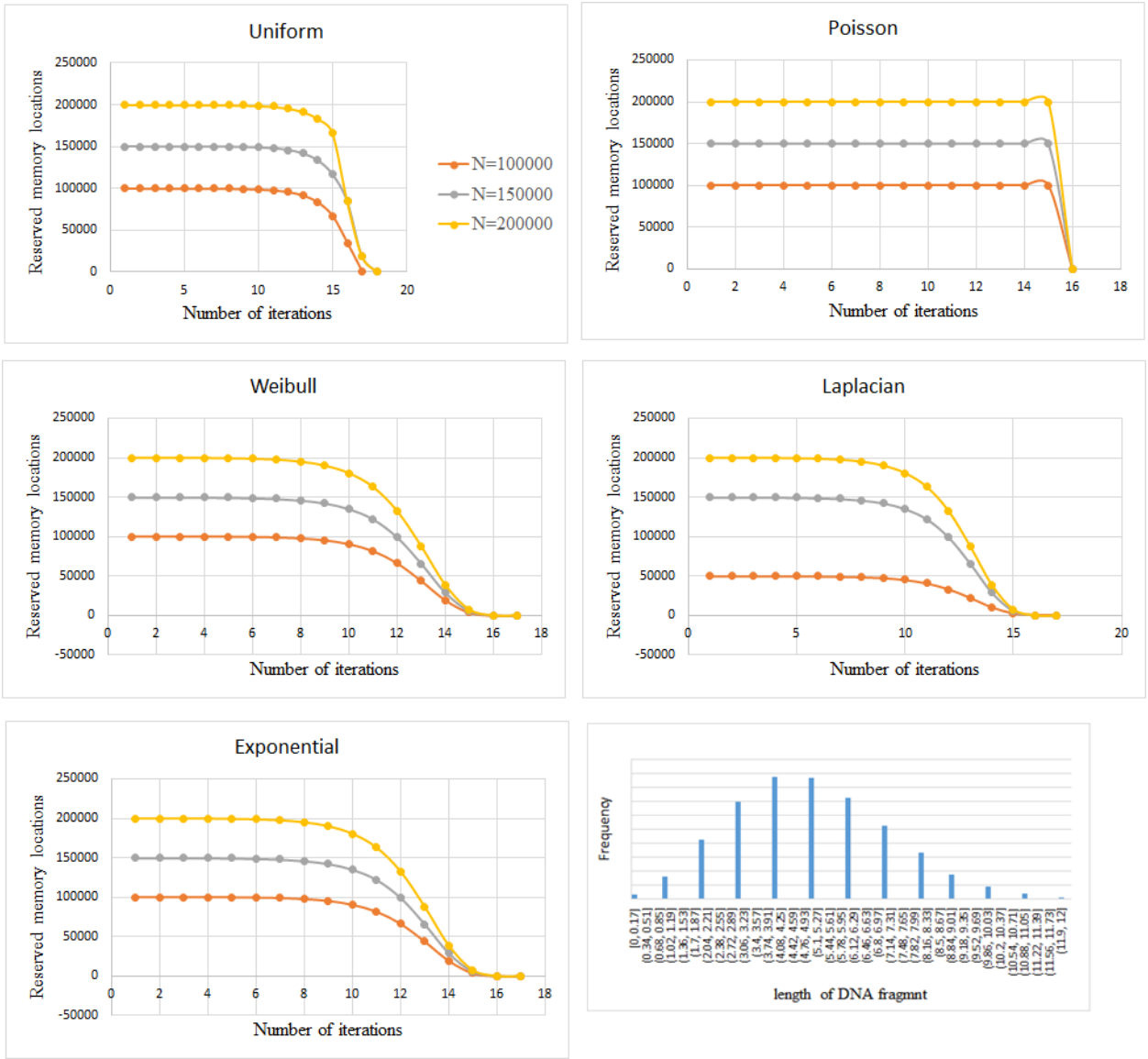

Figure 3: the number of reserved memory locations versus the number of iterations needed to completely sort the input datasets that was generated from the distribution shown on the top of each panel. Each panel shows the plots for three different sizes of the input dataset.

The biological data shown in the lower-right panel of Figure 3, derived from DNA fragments, presents distribution pattern that closely to Poisson- or Weibull-type distribution. This indicates that real genomic data follows structured statistical distribution rather than purely random variation. In this context, sorting algorithms serve as a crucial intermediate step in DNA-based computations, facilitating the efficient organization and analysis of large-scale genomic data

6. CONCLUSION AND FUTURE WORK.

The most influential factor that defines the efficiency of the sorting algorithms is time and space complexity but is actually strongly affected by the statistical characteristics of the input data. It implies that practically, there must be some sorting strategy, which relies not only on the theoretical assurances but also on the characteristics of the data sets. Well-known algorithms such as Merge Sort, Heap Sort, Quick Sort, Radix Sort, Shell Sort, etc. will tend to have predictable performance on a broad input distribution. Simple quadratic algorithms, like Bubble sort and Insertion sort are though inefficient as the size of the dataset increases and hence the less helpful in large scale or performance critical applications.

In reaction to this, Odd-Even Parity Based (OEPB) is proposed as a straightforward integer sorting algorithm, and it capitalizes on the gradual shrinking data storage structure so as to reduce the number of computational resources. Although in most situations it does not achieve the same performance as optimum O (n log n) sorting algorithms, it always performs better when compared to Bubble Sort, and competes with Insertion Sort, especially when the distribution is skewed such as Weibull and Laplace. This demonstrates that even the most basic of properties including numerical parity may be useful to limit the amount of redundant comparisons and optimize the run behavior of some data patterns.

The overall findings support the hypothesis that execution of lightweight heuristic principles during the design of the sorting can provide measurable performance benefits with no complexity to be added to the algorithm. This confirms the importance of distribution-wise approaches in developing useful data processing techniques to special applications. Moreover, extensive testing on genomic workload at a large scale - specifically, DNA k-mer datasets- will be necessary to evaluate its feasibility in bioinformatics applications.

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 regarding the publication of this manuscript.

REFERENCES

[1] A. H. Al-Khalidi and R. A. Abd-Alhameed, optimizing sorting for IoT data streams using parity-based preprocessing, Sensors, vol. 23, no. 4, 2156, 2023 http://doi.org/10.2478/amns-2024-2978
[2] A. M. Al-Badawi, Performance analysis of comparison-based sorting algorithms on modern CPU architectures, Journal of Computer Science and Technology, vol. 35, no. 4, pp. 812-825, 2020 http://doi.org/10.47392/IRJAEM.2025.0430
[3] N. S. Alghamdi and H. A. Aljamaan, The impact of distribution shape on sorting efficiency: An experimental analysis, Computers, Materials & Continua, vol. 72, no. 3, pp. 4567-4582, 2022 http://doi/10.2991/icecee-15.2015.224
[4] T. A. Puranik, Performance Analysis of Sorting and Searching Algorithms, International Research Journal on Advanced Engineering and Management (IRJAEM), vol. 3, no. 8, pp. 2741-2746, 2025 http://doi.org/10.47392/IRJAEM.2025.0430
[5] N. Faujdar and S. P. Ghrera, Analysis and Testing of Sorting Algorithms on a Standard Dataset, Fifth International Conference on Communication Systems and Network Technologies, Gwalior, India, pp. 962-967, 2015 http://doi.org/10.1109/CSNT.2015.98
[6] C. Bunse, Alshahrani and H. Höpfner, S. Roychoudhury and E. Mansour, Energy Efficient Data Sorting Using Standard Sorting Algorithms, Communications in Computer and Information Science, vol. 50, pp. 100567, 2021 http://doi.org/10.1007/978-3-642-20116-5_19
[7] Z. Xiaoke, Z. Qi, Z. Wei and L. Ling, Deep Learning Service for Efficient Data Distribution Aware Sorting, Future Generation Computer Systems, vol. 124, pp. 234-248, 2021 http//doi.org/10.48550/arXiv.1907.08817
[8] S. Sujarwo, HYBRID QUICKSORT: AN EMPIRICAL STUDY. CommIT (Communication and Information Technology) Journal. vol 7. pp 41, 2013 http//doi.org/10.21512/commit.v7i2.582
[9] G. Mateescu, Parallel sorting on heterogeneous platforms, Parallel sorting on heterogeneous platforms, pp. 116-117, 2002 http://doi.org/10.1109/HPCSA.2002.1019143
[10] Z. A. binti Zakaria and M. F. M. Said, A NEW EFFICIENT ALGORITHM BASED ON ODD-EVEN TRANSPOSITION SORT ALGORITHM FOR MERGESORT ON GRID COMPUTER. Journal of Advanced Computing Research, Vol 2, pp. 17-20, 2016 http://doi.org/10.21512/commit.v7i2.582
[11] F. Guo, R. Guan, Y. Li, Q. Liu, X. Wang, C. Yang and J. Wang, Foundation models in bioinformatics, National Science Review, Vol 12, 2025 https://doi.org/10.1093/nsr/nwaf028
[12] J. Fan, J. Khan, N. P. Singh, G. E. Pibiri & R. Patro, FULGOR: A FAST AND COMPACT K-MER INDEX FOR LARGE-SCALE MATCHING AND COLOR QUERIES. Algorithms for Molecular Biology, vol 19, pp.3, 2024
[13] A. S. Sabah, S. S. Abu-Naser, Y. E. Helles, R. F. Abdallatif, F. Y. A. Abu Samra, A. H. Abu Taha, N. M. Massa and A. A. Hamouda, COMPARATIVE ANALYSIS OF THE PERFORMANCE OF POPULAR SORTING ALGORITHMS ON DATASETS OF DIFFERENT SIZES AND CHARACTERISTICS, International Journal of Academic Engineering Research (IJAER), vol. 7, pp. 76-84, 2023
[14] S. Salous and R. Amro, "A novel odd-even based sorting algorithm for integers: An experimental comparison," Journal of Engineering Sciences and Information Technology, vol. 10, pp. 41-64, 2026 http//doi.org/10**.263**89/AJSR R26102**5** (In Press)

[15] K. A. Bakare, A. A Okewu, Z. A. Abiola, A. Jaji and A. Muhammed, A Comparative Study Of Sorting Algorithms: Efficiency And Performance In Nigerian Data Systems, The FUDMA Journal of Sciences, vol 8, pp 1-5, 2024 https://doi.org/10.33003/fjs-2024-0805-2730
[16] D. A. Shaik and A. K. Srinivas, "BQM Hybrid Sorting Algorithm, 7th International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), Bangalore, India pp. 1-5, 2023 http//doi.org/10.1109/CSITSS60515.2023.10334130
[17] K. Bhagat, A. K. Das, S. K. Agrahari, S. A. Shah, D. R. T, G. Ramasamy, Cross-Language Comparative Study and Performance Benchmarking of Sorting Algorithms, Proceedings of the 3rd International Conference on Optimization Techniques in the Field of Engineering (ICOFE -2024)
[18] A. Fenyi, M. Fosu & B. Appiah, Comparative Analysis of Comparison and Non-Comparison Based Sorting Algorithms, International Journal of Computer Application, vol 175, 2020

[19] D. E. Knuth, The Art of Computer Programming, Volume 3: Sorting and Searching, 2nd ed., Addison-Wesley, 1998.
[20] S. Sen and N. Gupta, Distribution-Sensitive Algorithms, Algorithm Theory, pp. 335-346, 2006 https://doi.org/10.1007/BFb0054380
[21] M. A. Ala'anzy, N. Tolendi, B. Baubek and A. Algarni, Performance Evaluation Of Gpu-Based Parallel Sorting Algorithms, vol 21, 2026 https://doi.org/10.1371/journal.pone.0342167
[22] L. V. Kale and S. Krishnan, A Comparison Based Parallel Sorting Algorithm, International Conference on Parallel Processing-ICPP'93, vol 3, pp196-200 http://doi.org/ 10.1109/ICPP.1993.17
[23] R. M. Dreger, Sorting Data Sets and Computing Medians for Skewed Distributions, Educational and Psychological Measurement, vol 55, pp. 141-153 1995 https://doi.org/10.1177/0013164495055005001
[24] V. Estivill-Castro and D. Wood, A Survey of Adaptive Sorting Algorithms, ACM Computing, vol 24, pp. 441-476 1992 https://doi.org/10.1145/146370.14638
[25] S. H. Park and J. S. Kim, "Parity-aware sorting for efficient database indexing," Journal of Database Management, vol. 33, no. 4, pp. 1-18, 2022.
[26] M. A. Rahman and S. S. Islam, "Performance prediction of sorting algorithms using statistical distribution features," IEEE Access, vol. 9, pp. 152341-152355, 2021.
[27] R. Sedgewick and K. Wayne, Algorithms, 4th ed., Addison-Wesley, 2011.
[28] L. M. Silva and R. C. Santos, "A survey of distribution-aware sorting techniques for big data," Big Data Research, vol. 29, 100334, 2022.
[29] D. P. Singh and R. K. Dwivedi, "Parity-based sorting for embedded systems with limited memory," Microprocessors and Microsystems, vol. 89, 104451, 2022.
[30] H. R. Tohidi and M. R. Hashemi, "A new perspective on sorting by partitioning based on numerical properties," Applied Soft Computing, vol. 112, 107789, 2021.

Appendix A: Pseudocode

Algorithm: OEPB (Odd-Even Parity-Based)

Algorithm: OddEvenSort(A)

Input: Unsorted array A[0..n-1] of positive integers Output: Sorted list Sorted

1. Initialize empty lists: OddList, EvenList, Sorted
2. move all even integers from A into evenList and odd integers from A to OddList
3. While (OddList is not empty OR EvenList is not empty):
4. Step 1: Move all elements equal to 1 from evenList to Sorted oddList to Sorted
5 Step 2: Move all elements equal to 1 from evenList to Sorted
6 Step 3: Divide all remaining elements in oddList and EvenList by 2
7. Step 4: move odd elements from evenList into oddList
8. Step 5: move even elements from oddList into evenList
9. Repeat Steps 1–5 until the oddList and evenList become empty.
10. End While