



Optimized Radix Sort for Efficient Health Data Sorting in Biomedical Backend Systems

Tugay Direk*

*Department of Software Engineering
Izmir University of Economics, Izmir, Turkiye
Email: tugay.direk@ieu.edu.tr

Abstract—Efficient data processing is essential in modern biomedical systems, especially as hospital information systems, genomic pipelines, and centralized clinical data platforms generate increasingly large volumes of health related information. This paper presents an optimized variant of the radix sort algorithm tailored for use in backend biomedical data handling systems. The algorithm maintains linear time complexity $O(n)$ and offers improved performance over comparison based algorithms in specific scenarios involving high volume, mixed digit size integer inputs typical of centralized health data aggregation. Our method outperforms quicksort in runtime efficiency under large scale, heterogeneous numeric inputs, such as those found in genomic data sorting or patient record integration. However, due to its use of additional memory buffers, including an extra bucket for digit length categorization, our approach incurs moderately higher memory usage compared to in place quicksort implementations. We evaluate this trade off and demonstrate the performance advantage in use cases where memory availability is not a primary constraint. This contribution supports the development of faster biomedical informatics infrastructures, enabling more responsive analytics and scalable integration of heterogeneous medical data sources.

Keywords—Radix Sort, Biomedical Informatics, Health Data Processing, Sorting Algorithms, Backend Systems, Genomic Data Analytics

I. INTRODUCTION

Healthcare systems increasingly rely on large scale data collected from biomedical devices, hospital information systems, and research platforms. These data must be processed rapidly and accurately by centralized systems to support clinical decision making, research, and health record integration [1]. Efficient sorting of large scale genomic data is essential for numerous bioinformatics tasks including variant analysis, interval overlap detection, and feature merging. Sorting algorithms play a big role in this task. For instance, AIList data structure introduced by Feng et al. optimized interval search in genomics. Part of its design includes sorting intervals by start coordinate and then structuring them with precomputed end values to speed up overlap queries [2]. Another study by Mao et al. evaluates augmented two dimensional range trees for genomic interval queries, including refined operations based on Allen's interval algebra. The implementation builds range trees using interval start positions as primary keys, implicitly

relying on sorted start coordinate data to optimize search operations [3]. Therefore, efficient sorting algorithms is a need for these types of work pipelines.

The study of sorting algorithms has a rich history, which has been extensively included in resources such as from Knuth [4] to Cormen et al. [5]. Sorting remains a cornerstone of data processing and analysis, with direct applications across biomedical engineering, particularly in genomic data processing, electronic health records (EHR) management, clinical signal analysis, and medical imaging [6], [7], [8]. Efficient sorting is also critical in feature alignment and biological sequence comparison where the order of elements carries biological or clinical meaning [9]. Traditional sorting algorithms, such as quick sort and merge sort, exhibit varying degrees of efficiency depending on the nature of the comparison approach. These algorithms rely on comparisons for all input types, which limits their potential for optimizations that could achieve linear time complexity. Radix sort, a non comparative integer sorting algorithm, has demonstrated superior performance in specific contexts. Firstly conceptualized by Harold H. Seward in 1954 [10], sorts integers by processing individual digits. It basically uses digit values from least significant digit through the most significant one and places the numbers in correct order after comparison of every digit. However, its effectiveness diminishes when handling arrays with integers of widely varying digit lengths. The algorithm proceeds as follows: it first identifies the number with the maximum digit length in the input array. This value determines the total number of digit wise passes to perform. Regardless of how many elements share this maximum digit length, even if only one does, all numbers are treated as having the same length by padding shorter values with leading zeros during the sorting process. This feature of the method makes it very inefficient on varying types of input. Because of this reason, radix sort faces performance degradation when applied to data arrays containing heterogeneous digit lengths, a common case in real world biomedical applications. For instance, the human genome exhibits considerable variability in chromosome sizes, ranging from large chromosomes such as chr1 (248 million base pairs) to relatively shorter ones like chr21 (46 million base pairs) [11]. Consequently, genomic coordinates span a wide range of digit lengths, from six digit to nine digit integers, creating challenges for radix based sorting algorithms when processing heterogeneous data. Similarly, hospital logs



and clinical telemetry data often include event timestamps with inconsistent digit representations due to varying time resolutions [12]. In these cases, radix sort treats all elements as having the maximum digit length in the array, padding shorter values with leading zeros. This introduces unnecessary computation and leads to performance inefficiencies.

Previous research has explored various enhancements to radix sort. Several optimizations were proposed for parallel radix sort algorithms, improving efficiency on multicore processors. Advancements by Lee et al. [13], Cho et al. [14] and Satish et al. [15] have focused on optimizing radix sort for modern hardware architectures. However, the challenge of handling integers with varying digit lengths remains relatively unexplored. To the extent of our research, digit length based grouping has not been formally studied or applied in the context of radix sorting, including in biomedical data applications. Our work introduces and evaluates this approach specifically for genomic intervals and health record identifiers, demonstrating clear performance gains.

This paper proposes a digit length aware radix sorting algorithm designed to better handle digit diverse integer arrays found in biomedical datasets. By introducing an initial categorization phase that segments data by digit length, the method executes radix sorting more efficiently on each subgroup. This strategy reduces redundant passes over shorter integers and improves runtime under practical data conditions.

We demonstrate the real world applicability of our method in genomic interval analysis, a routine task in bioinformatics and biomedical research, where start-end intervals from variant files (such as BED file type) must be sorted for intersection, merging, and feature mapping. These coordinates often span chromosomes with digit length imbalance, making the proposed method well suited for this domain [16].

II. METHOD

Genomic data analysis often involves operations like detecting overlapping features and computing feature intersections [16]. These tasks require coordinate based interval sorting where chromosome positions range from a few digits (10-500 base pairs (bp)) to 15 digit representations for long chromosomes or genome assemblies [11]. Tools like BEDTools [17] need sorted BED files for operations like intersect, merge, and subtract for rapid execution, making efficient sorting vital [18]. Another tool BEDOPS also requires sorted input for high performance genomic feature operations [19]. Sorting genomic intervals accurately and efficiently has real world implications in biomedical research. For example, when comparing Single Nucleotide Variants (SNVs) across samples, sorted inputs enable fast binary-search-style lookups and efficient overlap detection, as implemented by tools like Tabix which require position-sorted inputs for interval retrieval [20].

Since sorting the numbers is a significant task in mentioned real world problems, optimizing radix sort algorithm which has promising outcomes is a contribution to the field. The main problem in traditional radix sort arises from the initial

assumption which requires every number must be assumed to have the same digit length of the number with maximum digit length. For this reason, zero is assigned for the missing digits for each number. The run time of radix sort is linear theoretically, but because of this assumption, it never reaches to its best case when even one number with high digit length is present in the input array. This assumption forces the algorithm for unnecessary iterations although we do not need. Proposed method solves the problem of having numbers with highly varying numbers of digits by separating each number to a different categorization based on their number of digits. The optimized radix sort algorithm consists of three main steps:

Firstly, the digit length categorization phase initializes a separate empty list to hold categories of integers based on their digit lengths and the input array is traversed, categorizing each integer by its digit length. Next, radix sort is applied to each category independently. Lastly, the sorted categories are merged to produce the final sorted array. This procedure actually mimics the human recognition for the numbers. To compare the numbers with different digit lengths, we just look at the digit length and know the one with higher value is greater. We do not need to compare the numbers with different digit lengths. This method also does not consider the numbers with different digit lengths and only focuses on numbers with the same digit lengths. By this key idea, the best case running time of radix sort can always be achieved independent from the varying input types. Pseudocode of the method is given in Algorithm 1:

Algorithm 1 OptimizedRadixSort(arr)

- 1: Initialize separate lists for each digit length, where $\text{digit_classes}[i]$ will hold numbers with $(i + 1)$ digits
 - 2: For each number in arr:
 - 3: Determine the number of digits d in the number
 - 4: Append the number to $\text{digit_classes}[d-1]$
 - 5: For each digit_class in digit_classes :
 - 6: If the digit_class is not empty:
 - 7: Apply $\text{RadixSort}(\text{digit_class})$
 - 8: Initialize sorted_arr as an empty list
 - 9: For each digit_class in digit_classes :
 - 10: Append all numbers in the digit_classes to sorted_arr
-

III. DISCUSSION

The running time of the optimized radix sort algorithm can be analyzed as follows;

Digit Length Categorization: Categorizing numbers by digit length has a time complexity of $O(n)$, where n is the number of integers. For each number, determining its digit length requires at most 15 simple integer comparisons (assuming a maximum of 15 digits), which is much faster than costly string conversions.

Radix Sort on Categories: For each category, radix sort runs in $O(d * (n + b))$, where d is the number of digit, n is the input size and b is the base of the number. Since the b and d values



are small values for high input sizes, this asymptotic run time is considered to be linear. However, as mentioned in previous sections, for small number of integers with high digit lengths, this slows down the algorithm in considerable manner.

Merging Sorted Categories: This step involves concatenating the sorted arrays. Since each category would be sorted after radix sort. It will require only linear time to concatenate them.

The key idea for the best case running time in radix sort is achieved through the categorization of each number according to its digit length. By this procedure, unnecessary iterations for smaller numbers with less digit lengths are avoided and best running time is always guaranteed. Thus, the overall time complexity of the optimized radix sort is $O(n)$. To evaluate the performance of the algorithm, experiments with varying input sizes were conducted. The results in Table I, II and III demonstrate the robust performance of the proposed method across different dataset sizes, validating the theoretical analysis. Merge sort, quick sort and traditional radix sort methods are considered for the evaluation of the method. For the comparison of these algorithms, an everyday personal computer that any person can have access was used and it has Intel core i5-6600K CPU at 3.30 GHz and 8 GB RAM.

We designed test arrays of integers for evaluation purposes. The majority are five digit or less, while a minority have up to 15 digits, mimicking genomic coordinate distributions. Analyses using the GenomicDistributions package on BED formatted data demonstrate that interval lengths in real datasets exhibit long tailed distributions, spanning from small (hundreds of base pairs) to very large genomic regions (hundreds of billions) [21]. This variability naturally translates into digital length heterogeneity when represented as integer coordinates for sorting, a key challenge addressed by our proposed digit-length-aware radix sort algorithm. Our test arrays replicate BED data characteristics from organisms with large genomes or long read sequencing technologies. These data are also available in the BEDTools sample intervals repository¹. We synthetically generated BED files data exhibiting such features to realistically model digit length heterogeneity.

Genomic intervals in BED files often consist of a chromosome label and two integers representing start and end positions. For example:

```
chr1 123 150
chr2 45678 46000
chr3 123456789012345 123456789012360
```

Sorting is critical to run tools like BEDTools merge, which merges overlapping or adjacent intervals. These tools require lexicographical order on chromosome names and numerical order on start positions. Since intervals on different chromosomes can vary greatly in length (from hundreds to billions), sorting performance may degrade due to variable digit lengths.

¹<https://github.com/arq5x/bedtools2/blob/master/data/knownGene.hg18.chr21.bed>

Our method improves this by focusing on the numeric digit length component.

TABLE I: Running time comparison of sorting methods by varying input and digit sizes. Input array consists of equal number of integers for each digit length (s: seconds)

Input Size	Max Digit Length	Merge Sort	Quick Sort	Radix Sort	Optimized Radix Sort
1,000,000	15	8.7092 s	4.4872 s	16.4054 s	8.0623 s
1,000,000	12	9.3178 s	4.6163 s	11.7712 s	6.3523 s
1,000,000	8	9.3198 s	2.9651 s	7.2773 s	4.3360 s
1,000,000	5	9.1830 s	2.2092 s	4.7842 s	3.1801 s
10,000,000	15	106.8788 s	48.7653 s	165.4651 s	97.3188 s
10,000,000	12	126.5181 s	48.3542 s	164.3529 s	92.7429 s
10,000,000	8	117.3067 s	45.6204 s	84.8983 s	54.5895 s
10,000,000	5	111.7021 s	28.8122 s	46.8787 s	35.0197 s
10,000,000	3	108.2604 s	15.9188 s	26.1440 s	22.0906 s

Table I shows the results of different methods with varying input and digit sizes. When input size is one million, four different maximum digit lengths were considered. Input array consists of equal number of integers for each digit length.

As can be detected from the table, proposed method outperforms merge sort and radix sort in all maximum digit lengths. Quick sort performs better in this comparison. However, while the length of maximum digit length is reduced, our method closely approaches quick sort. The same phenomena can be observed for the input size of ten million.

TABLE II: Running time comparison of sorting methods by varying input and digit sizes. Input array consists of only 100 integers with at most 15 digits length and the others are at most 5 digits length.

Input Size	Merge Sort	Quick Sort	Radix Sort	Optimized Radix Sort
10,000,000	128.3958 s	79.6171 s	164.2375 s	66.8402 s
1,000,000	10.2567 s	5.4478 s	15.3449 s	6.1318 s
100,000	0.8134 s	0.3913 s	1.2569 s	0.5317 s

TABLE III: Running time comparison of sorting methods by varying input and digit sizes. Input array consists of only 10,000 integers with at most 15 digits length and the others are at most 5 digits length.

Input Size	Merge Sort	Quick Sort	Radix Sort	Optimized Radix Sort
10,000,000	127.2835 s	78.7963 s	167.2343 s	67.8085 s
1,000,000	9.9886 s	5.0389 s	14.9283 s	6.3450 s

Table II and III demonstrates the clear superiority of our method. Table II consists of inputs with only 100 integers with at most 15 digits length and the others are at most 5 digits length. Input sizes are one thousand, one million and ten million. When the input size is ten million, our method outperforms the quick sort by more than 13 seconds. This corresponds to a performance gain exceeding 15% in execution time. On the input sets with one thousand and one million, the proposed method and quick sort are very close to each other. The same pattern can also be seen in Table III. These results highlight the clear performance advantage of the proposed method.

The biomedical engineering domain, especially in genomics, provides a compelling use case for this optimization. Genomic interval processing tasks, such as merging overlapping intervals or identifying intersections between genomic features and known variants, frequently involve large datasets encoded in BED or VCF formats. These datasets inherently contain coordinate values with varying digit lengths (positions ranging from a few hundred to over a billion base pairs). Accurate and efficient sorting of these genomic intervals is critical for tools like BEDTools, BEDOPS, and others that rely on linear ordering of features to perform interval comparisons and overlaps. By employing the proposed sorting algorithm, such preprocessing steps can become significantly more time and memory efficient, particularly when scaled to whole genome analyses.

While the primary application context discussed has been genomic data, the proposed method may also be applicable in clinical data systems where sorting variable length numeric keys (such as national IDs or hospital specific patient codes) is required. Although these identifiers might not exhibit the same degree of digit variability or sorting load as genomic coordinates, systems with large scale patient data could still benefit from the performance enhancements offered by digit aware sorting methods.

Future works can include extending this method to parallel or distributed computing environments, which are increasingly used in genomics and biomedical data processing. It can also be integrated into established genomic libraries or clinical informatics platforms to evaluate real world system level performance.

IV. CONCLUSION

This paper presents an optimized radix sort algorithm for arrays containing integers with varying digit lengths. By categorizing integers based on digit length, applying radix sort to each category, and merging the results, the algorithm achieves a time complexity of $O(n)$ theoretically. Experimental results confirm the algorithm's efficiency across different input sizes, making it a robust choice for sorting large datasets with diverse digit lengths. Close and sometimes better results of quick sort algorithm is based on its implementation details which does not require high overhead. However, based on the types of inputs, the proposed method outperforms quick sort although it requires more overhead such as the inclusion of extra arrays to hold each category and merging at the last step. Parallel computing with threading may be a future work to increase its time efficiency even more. A different thread can be initialized for each digit class and better results can be achieved. This work opens up a promising direction for bridging the efficiency gap in sorting operations across biomedical domains where numerical inputs with variable digit lengths are common.

REFERENCES

[1] B. Kornelia and S. Andrzej, "The use of big data analytics in healthcare," *Journal of Big Data*, 2022. [Online]. Available: <https://doi.org/10.1186/s40537-021-00553-4>

[2] J. Feng, A. Ratan, and N. C. Sheffield, "Augmented interval list: a novel data structure for efficient genomic interval search," *Bioinformatics*, vol. 35, no. 23, pp. 4907–4911, 05 2019. [Online]. Available: <https://doi.org/10.1093/bioinformatics/btz407>

[3] M. Chengsheng, E. Alal, and L. Yuan, "Efficient genomic interval queries using augmented range trees," *Scientific Reports*, 2019. [Online]. Available: <https://doi.org/10.1038/s41598-019-41451-3>

[4] D. E. Knuth, *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2009.

[6] B. A. G. et al, "Genomic data in the all of us research program," *Nature*, 2019. [Online]. Available: <https://doi.org/10.1038/s41586-023-06957-x>

[7] R. MK, W. W, and O.-M. L., "'big data' and the electronic health record," *Yearb Med Inform.*, 2014. [Online]. Available: <https://pmc.ncbi.nlm.nih.gov/articles/PMC4287068/>

[8] M. Santos and N. P. Rocha, "A big data approach to explore medical imaging repositories based on dicom," *Procedia Computer Science*, vol. 219, pp. 1224–1231, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050923004143>

[9] S. H. et al, "Next-generation sequencing technology: Current trends and advancements," *PubMed-not-MEDLINE*, 2024. [Online]. Available: <https://pmc.ncbi.nlm.nih.gov/articles/PMC10376292/>

[10] A. Cassar and Emmanuel, "In search of the fastest sorting algorithm," *Symposia Melitensia*, 2018. [Online]. Available: <https://www.um.edu.mt/library/oar/handle/123456789/30001>

[11] W. L. G, G. J. G. R, H. K, T. S, H. T, and H. J. L, "The vertebrate genome annotation (vega) database," *Nucleic Acids Research*, 11 2007. [Online]. Available: <https://doi.org/10.1093/nar/gkm987>

[12] G. A. et al, "Timing errors and temporal uncertainty in clinical databases-a narrative review," *PubMed-not-MEDLINE*, 2022. [Online]. Available: <https://pmc.ncbi.nlm.nih.gov/articles/PMC9433547/>

[13] S.-J. Lee, M. Jeon, D. Kim, and A. Sohn, "Partitioned parallel radix sort," *Journal of Parallel and Distributed Computing*, vol. 62, no. 4, pp. 656–668, 2002. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0743731501918088>

[14] M. Cho, D. Brand, R. Bordawekar, U. Finkler, V. Kulandaisamy, and R. Puri, "Paradis: an efficient parallel algorithm for in-place radix sort," *Proc. VLDB Endow.*, vol. 8, no. 12, p. 1518–1529, aug 2015. [Online]. Available: <https://doi.org/10.14778/2824032.2824050>

[15] N. Satish, M. J. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore gpus," *2009 IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–10, 2009. [Online]. Available: <https://api.semanticscholar.org/CorpusID:7697594>

[16] O. Burçak and C. Tolga, "Joa: Joint overlap analysis of multiple genomic interval sets," *BMC Bioinformatics*, 2019. [Online]. Available: <https://doi.org/10.1186/s12859-019-2698-4>

[17] Q. A. R. and H. I. M., "Bedtools: a flexible suite of utilities for comparing genomic features," *Bioinformatics*, vol. 26, no. 6, 01 2010. [Online]. Available: <https://doi.org/10.1093/bioinformatics/btq033>

[18] Q. A. R., "Bedtools: The swiss-army tool for genome feature analysis," *Current Protocols in Bioinformatics*, vol. 47, no. 1, pp. 11.12.1–11.12.34, 2014. [Online]. Available: <https://currentprotocols.onlinelibrary.wiley.com/doi/abs/10.1002/0471250953.bi1112s47>

[19] N. Shane, K. M. Scott, R. A. P., H. Eric, T. R. E., J. A. K., R. Eric, M. M. T., V. Jeff, T. Sean, S. Richard, H. Richard, and S. J. A., "Bedops: high-performance genomic feature operations," *Bioinformatics*, vol. 28, no. 14, pp. 1919–1920, 05 2012. [Online]. Available: <https://doi.org/10.1093/bioinformatics/bts277>

[20] L. Heng, "Tabix: fast retrieval of sequence features from generic tab-delimited files," *Bioinformatics*, vol. 27, no. 5, pp. 718–719, 01 2011. [Online]. Available: <https://doi.org/10.1093/bioinformatics/btq671>

[21] K. K. et al, "Genomicdistributions: fast analysis of genomic intervals with bioconductor," *BMC Genomics*, 2022. [Online]. Available: <https://doi.org/10.1186/s12864-022-08467-y>