Optimization of Search and Sorting Algorithms for Transaction Data Processing using Olist E-Commerce Datasets

Muhammad Aditya Bayhaqie
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021282227122@student.unsri.ac.id

Nabila Nurhusna Yap
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021182227032@student.unsri.ac.id

Faradilla Maulia
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021282227116@student.unsri.ac.id

Aisyah Fitri Indar
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021182227109@student.unsri.ac.id

Adityo Pangestu
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021282227061@student.unsri.ac.id

Bunga Wiranti
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021182227120@student.unsri.ac.id

Nabila Kurnia Aprianti
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021183227003@student.unsri.ac.id

Nadiya Puspita Sari
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021182227110@student.unsri.ac.id

Annisa Reida Raheima
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021282227072@student.unsri.ac.id

Agi Agustian Davi
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021282227118@student.unsri.ac.id

Zweta Anggun Syafara
Departement of Informatics,
Faculty of Computer Science,
University of Sriwijaya
Indralaya, Indonesia
09021182227010@student.unsri.ac.id

***Abstract*— The rapid growth of e-commerce platforms has led to an exponential increase in transactional data, demanding efficient data processing techniques to ensure timely and accurate insights. In this study, we investigate the optimization of search and sorting algorithms specifically tailored for transaction data processing. Using the Brazilian E-Commerce Public Dataset by Olist, we evaluate the performance of five algorithmic combinations: Jump Search - Heap Sort, Jump Search - Merge Sort, Hash Search - Selection Sort, Jump Search - Bubble Sort, and Binary Search - Counting Sort. Each combination is tested for its ability to retrieve and organize transactional records efficiently, with execution time as the primary performance metric. The experimental results demonstrate varying degrees of effectiveness, where the combination of Binary *Search* - Counting Sort achieves the best performance with an average execution time of 3.916504 seconds, followed closely by Jump Search - Heap Sort with 4.442105 seconds. These findings provide a comparative perspective on algorithmic efficiency, guiding practitioners in selecting suitable techniques for processing large-scale e-commerce transaction datasets. Future work may extend this analysis to include parallel processing approaches and real-time applications.**

Keywords*— *Search algorithms, Sorting algorithms, Transaction data, Performance optimization, E-commerce dataset.

I. INTRODUCTION

In the modern era of digital commerce, one of the primary challenges facing e-commerce platforms is the

efficient processing of rapidly growing transactional data. As platforms like Olist continue to scale, they generate massive volumes of data daily — including order records, customer interactions, and payment logs — which are crucial for business intelligence, real-time order management, fraud detection, and personalized recommendation systems. According to Statista (2023), global retail e-commerce sales reached approximately $5.8 trillion in 2023 and are projected to surpass $6.3 trillion by 2024. This exponential growth underscores the urgent need for fast and scalable data processing techniques.

At the heart of transactional data processing lie two fundamental algorithmic operations: searching and sorting. These operations are essential for organizing, retrieving, and analyzing large datasets efficiently. However, traditional algorithms such as linear search and bubble sort often struggle with high execution times when handling large-scale data [1]. Selecting the right algorithms becomes even more critical as data volume and complexity increase.

This study focuses on optimizing search and sorting algorithms for processing e-commerce transaction data, specifically using the Brazilian E-Commerce Public Dataset by Olist [2], which contains over 100,000 records across orders, customers, payments, and logistics. We evaluate a range of classical and advanced algorithms including binary search, sequential search, jump search, interpolation search, hash search, and hash table lookup for searching; and quick sort, merge sort, heap sort, radix sort, bubble sort, selection sort, counting sort, and bucket sort for sorting.

Each algorithm offers distinct trade-offs. For instance, binary search provides logarithmic time complexity but requires sorted data [3], while hash-based methods enable near-constant lookup times with an effective hash function. Similarly, merge sort and quick sort offer reliable O(n log n) performance [3][4][5][6], while non-comparison-based algorithms like radix sort, counting sort, and bucket sort can outperform traditional methods under specific data characteristics.

In this research, we specifically assess the performance of algorithm combinations such as Jump Search - Heap Sort, Jump Search - Merge Sort, Hash Search - Selection Sort, Jump Search - Bubble Sort, and Binary Search - Counting Sort. Execution time is the primary performance metric used, as it directly impacts the responsiveness of transactional systems.

Previous studies have shown that algorithm performance can vary significantly depending on data structure and distribution [1][7]. Therefore, the research problem addressed is identifying which combinations of search and sorting algorithms deliver the best execution performance when applied to large-scale transactional datasets.

This study is conducted as an offline evaluation and does not cover real-time or distributed implementation aspects. Nevertheless, the findings aim to provide practical insights for system designers, engineers, and developers in selecting and tuning algorithmic strategies to process e-commerce transaction data efficiently.

II. THEORETICAL FOUNDATION

Algorithms form the core foundation of data processing, particularly in search and sorting operations. Selecting the most efficient algorithm is crucial to optimizing system performance, especially in terms of execution time and memory usage [1]. Therefore, benchmarking is essential to evaluate and compare the performance of various algorithms under different data conditions.

A. Jump Search

Jump Search is an efficient searching algorithm for sorted data. It works by jumping ahead by fixed intervals (commonly $\sqrt{n}$) to locate a block where the target element may reside, and then performing a linear search within that block. Its time complexity is $O(\sqrt{n})$, offering improved performance over linear search for large datasets [8].

B. Binary Search

Binary Search is a well-known searching algorithm used exclusively on sorted data. It repeatedly divides the search space in half and compares the target element to the middle element. It has a time complexity of O(log n), making it very efficient for large datasets [7].

C. Hash Search

Hash Search utilizes a hash table to enable constant-time average search performance, i.e., O(1). It is extremely fast, although its worst-case performance can degrade to O(n) due to hash collisions, which are dependent on the quality of the hash function used [9].

D. Heap Sort

Heap Sort is a comparison-based sorting algorithm that relies on the heap data structure. It has a time complexity of O(n log n) for all cases and is memory efficient since it does not require additional storage beyond the input array. However, it is not a stable sorting algorithm [10].

E. Merge Sort

Merge Sort is a divide-and-conquer algorithm that recursively splits the dataset and merges the sorted subarrays. It is stable and performs well on large datasets with a time complexity of O(n log n).

However, it requires additional space proportional to the input size [11].

F. Selection Sort

Selection Sort is a simple comparison-based algorithm that repeatedly selects the minimum element and places it in the correct position. It has a time complexity of O(n^2) for all cases and is not suitable for large datasets, but it is useful for educational purposes due to its simplicity [12].

G. Counting Sort

Counting Sort is a non-comparative sorting algorithm that counts the occurrence of each element and uses this count to determine the position of each element in the output array. It is particularly efficient for integer values within a limited range, with a time complexity of O(n+k), where k is the range of input values [1].

Benchmarking these combinations of search and sort algorithms allows for the evaluation of their performance under various dataset sizes. This comparison helps determine the most optimal pairings in terms of execution time and memory usage across different scenarios.

III. RESEARCH METHODS

The research is divided into two stages as follows :

A. *Algorithm Searching using Synthetic Datasets*

a. *Research System*

- *Comprehensive literature review of search algorithms:*
  1. *Linear Search*
  2. *Binary Search*
  3. *Jump Search*
  4. *Hash Search*
  5. *Interpolation Search*
- *Analysis of sorting algorithms:*
  1. *Bubble Sort*
  2. *Selection Sort*
  3. *Insertion Sort*
  4. *Merge Sort*
  5. *Quick Sort*
  6. *Heap Sort*
  7. *Counting Sort*
- *Theoretical complexity analysis of algorithms*
- *Identification of potential optimal combinations*

b. *Synthetic Data Generation*

- *Creation of five synthetic datasets with varying characteristics:*
  1. *Dataset 1: Large-scale transaction data with Skewed distribution*
  2. *Dataset 2: Large-scale transaction data with normal distribution*
  3. *Dataset 3: Large-scale transaction data with modifications.*
  4. *Dataset 4: Large-scale transaction data with complex relationships*
  5. *Dataset 5: Large-scale transaction data with temporal patterns*
- *Controlled data volume scaling (10^3 to 10^4 records)*
- *Systematic variation of data attributes and distributions containing*

TABLE I. Data Synthetic Feature name and Data type

Feature Name	*Feature Data Type*
ID	*int64/object*
Nama/Nama Pelanggan	*object*
Jumlah Pembelian/Total Pembelian	*int64/float64*

c. *System Development*

- *Implementation of all search and sorting algorithms in Python 3.10*
- *Development of benchmarking framework with timing instrumentation*
- *Creation of algorithm combination testing infrastructure*
- *Implementation of performance measurement system with microsecond precision*
- *Development of result logging and analysis framework*

d. *System Benchmarking*

- *Systematic pairing of search and sorting algorithms*
- *Execution of all algorithm combinations on synthetic datasets*
- *Multiple runs (30 iterations) for statistical significance*
- *Measurement of execution times and resource utilization*
- *Identification of the best-performing algorithm combinations*

B. *Benchmarking and Validation using Olist Dataset*

a. *Research System*

- *Systematic pairing of search and sorting algorithms*

- *Execution of all algorithm combinations on synthetic datasets*
- *Multiple runs (30 iterations) for statistical significance*
- *Measurement of execution times and resource utilization*
- *Identification of the best-performing algorithm combinations*

b. *Data Collection*
- *Acquisition of the Brazilian Olist E-Commerce public dataset from Kaggle (https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce)*
- *Dataset consisting of 9 related tables with a total of 52 columns:*

TABLE II. Real World datasets information

Dataset Name	*Total Columns*
olist_customers_dataset.csv	*5*
olist_geolocation_dataset.csv	*5*
olist_order_items_dataset.csv	*7*
olist_order_payments_dataset.csv	*5*
olist_order_reviews_dataset.csv	*7*
olist_orders_dataset.csv	*8*
olist_products_dataset.csv	*9*
olist_sellers_dataset.csv	*4*
product_category_name_translation.csv	*2*

- *Initial assessment of data quality and completeness*
- *Identification of key transaction attributes relevant for search and sort operations*

c. *Data Preparation*
- *Data cleaning and normalization procedures:*
  1. *Handling missing values*
  2. *Standardizing formats and units*
  3. *Resolving data type issues*
- *Creation of unified data views for transaction processing*
- *Data partitioning into test and validation sets*
- *Format standardization across all tables*

d. *System Development*
- *Implementation of the four best algorithm combinations from Phase 1*
- *Adaptation of algorithms for Olist dataset characteristics*
- *Development of specialized data structures for e-commerce transaction processing*
- *Implementation of performance monitoring and logging systems*
- *Integration with Olist data processing pipeline*

e. *Benchmarking*
- *Execution of the four best algorithm combinations on the Olist dataset*
- *Measurement of performance metrics:*
  1. *Execution time*
- *Throughput (transactions processed per second)*
- *Comparison of algorithm performance across different transaction types*
- *Stress testing with varying data volumes*
- *Identification of the optimal algorithm combination for Olist transaction data*

f. *Data Visualization and Benchmark Comparing*
- *Visualization of algorithm performance patterns*
- *Comparative analysis between synthetic and real-world performance*
- *Creation of decision support visuals for algorithm selection*

IV. RESULT AND DISCUSSION

A. *Comparison Results and Algorithm Testing*

This section presents the results of testing and performance comparison of various search and sorting algorithms conducted on several datasets. The analysis aims to determine which algorithms are the most efficient and stable for processing transaction data.

1. First Comparison

The first comparison shows that Jump Search has the fastest and most stable search time across all attributes and datasets, with an average time below 0.003 seconds. In contrast, Interpolation Search tends to be slower and less stable. For data sorting, Heap Sort outperforms Radix Sort, exhibiting lower and more consistent execution times across all datasets. These results indicate that Jump Search and Heap Sort are the best choices in this scenario due to their efficiency and stability.

TABLE III. Best Algorithms in the First Comparison

Algorithm	Execution Time (ms)
Jump Search - Heap Sort	0,0038265
Jump Search - Merge Sort	0,0062123
Hash Search - Selection Sort	0,0193012
Binary Search - Counting Sort	0,0212561

2. Second Comparison

The results of the second table show that the combination of Binary Search with Counting Sort is the most optimal compared to other combinations. Jump Search with Heap Sort is still relatively efficient and slightly better than Jump Search with Merge Sort. Meanwhile Hash Search with Selection Sort has the lowest performance in this comparison.

TABLE IV. Best Algorithms in the Second Comparison

Algorithm	Execution Time (ms)
Jump Search - Heap Sort	0,000126
Jump Search - Merge Sort	0,000338
Hash Search - Selection Sort	0,001046
Binary Search - Counting Sort	0,000107

3. Third Comparison

Based on the third result of this table, the combination of Binary Search with Counting Sort still shows the most optimal performance with the lowest execution time. Jump Search with Heap Sort is in the middle position with quite good efficiency, while Jump Search with Merge Sort shows slower performance. On the other hand, Hash Search with Selection Sort has the worst performance, with the highest execution time compared to other combinations.

TABLE V. Best Algorithms in the Third Comparison

Algorithm	Execution Time (ms)
Jump Search - Heap Sort	6,573295
Jump Search - Merge Sort	11,222200
Hash Search - Selection Sort	4.162,894191
Binary Search - Counting Sort	0,000172

4. Fourth Comparison

This fourth comparison shows that Hash Selection is consistently the most superior search algorithm, with wide attribute coverage and stable performance on both datasets. Jump - Merge recorded the lowest execution time overall, showing high efficiency and stability in data search. Meanwhile, Binary Counting is only applicable to some attributes. These results confirm that Hash Selection and Jump - Merge are the best algorithm choices in search scenarios on large and diverse datasets.

TABLE VI. Best Algorithms in the Fourth Comparison

Algorithm	Execution Time (ms)
Jump Search - Heap Sort	12.676375
Jump Search - Merge Sort	7.267151
Hash Search - Selection Sort	15159.86269

Binary Search - Counting Sort	0.282717

5. Fifth Comparison

This Fifth shows that Binary Search with Counting Sort is the fastest algorithm combination, with the lowest total processing time compared to the other algorithms. Meanwhile, Hash Search with Selection Sort resulted in the highest runtime, indicating a lack of efficiency in this scenario. Jump Search with Heap Sort and Merge Sort recorded moderate runtimes, but still far above Binary Search. These results confirm that Binary Search - Counting Sort is the most optimal algorithm choice for time efficiency in searching and sorting on this dataset.

TABLE VII. Best Algorithms in the Fourth Comparison

Algorithm	Execution Time (s)
Jump Search - Heap Sort	4.442105
Jump Search - Merge Sort	82.629320
Hash Search - Selection Sort	191.159439
Binary Search - Counting Sort	3.916504

B. Analysis of the Best Algorithm Selection

Based on the results of the five comparisons conducted, consistent conclusions can be drawn regarding the performance of each algorithm. For search algorithms, Hash Search emerged as the best-performing method overall due to its consistent speed and stability, particularly in handling large and diverse datasets. Although Jump Search showed excellent results in the early comparisons with very low execution times, Hash Search demonstrated superior performance in broader attribute coverage and maintained stability across all test cases. Binary Search also showed strong performance, especially when paired with Counting Sort, but its applicability was limited to certain attributes. Meanwhile, in the sorting algorithm category, Merge Sort proved to be the most efficient and stable across various dataset conditions. While Heap Sort and Counting Sort offered good performance in some scenarios, Merge Sort consistently delivered low and reliable execution times. In contrast, algorithms like Selection Sort and others such as Insertion and Bubble Sort exhibited much higher runtimes, making them less ideal for medium to large-scale data. Considering execution speed, consistency, and adaptability across datasets, Hash Search was selected as the best search algorithm and Merge Sort as the best sorting algorithm for the final implementation phase.

C. Final Testing and Evaluation of the Best Algorithms

The final testing was conducted on the two best algorithms, namely Hashing Search for searching and Merge Sort for data sorting. This test used a real transaction dataset to measure performance under more realistic conditions. The results showed that both algorithms continued to deliver efficient and stable performance. Hashing Search demonstrated high speed with a time complexity of O(1), as the search is performed directly via indexing without iteration. Meanwhile, Merge Sort, with a time complexity of O(n log n), remains a good choice for sorting due to its stable and consistent nature. The test results indicate that both algorithms are still capable of providing efficient and stable performance on real data structures.

TABLE VIII. Final Test Results in the Jump Search and Heap Sort

Algorithm	Data Attribute	Average Execution Speed (s)
Jump Search - Heap Sort	*olist_customers_dataset.csv*	4,95905
	olist_geolocation_dataset.csv	4,433326
	olist_order_items_dataset.csv	9,575943
	olist_order_payments_dataset.csv	0,104876496
	olist_order_reviews_dataset.csv	0,1272580933
	olist_orders_dataset.csv	5,166480
	olist_products_dataset.csv	1,406815

	olist_sellers_dataset.csv	0,008779
	product_category_name_translation.csv	1,406815
Average Execution Time (ms)		3,021038065

TABLE IX. Final Test Results in the Jump Search and Merge Sort

Algorithm	Data Attribute	Execution Speed (s)
Jump Search - Merge Sort	*olist_customers_dataset.csv*	2,602627
	olist_geolocation_dataset.csv	82,594050
	olist_order_items_dataset.csv	4,664524
	olist_order_payments_dataset.csv	5,904933966
	olist_order_reviews_dataset.csv	4,4535982
	olist_orders_dataset.csv	9,629015
	olist_products_dataset.csv	1,593185
	olist_sellers_dataset.csv	0,035270
	product_category_name_translation.csv	0,000338
Average Executions Time (ms)		12,386393466

TABLE X. Final Test Results in the Hash Search and Selection Sort

Algorithm	Data Attribute	Average Execution Speed (s)
	olist_customers_dataset.csv	2861,79341
	olist_geolocation_dataset.csv	188,72467
	olist_order_items_dataset.csv	1923,851715
Hash Search - Selection Sort	*olist_order_payments_dataset.csv*	3987,05029
	olist_order_reviews_dataset.csv	3484,43132
	olist_orders_dataset.csv	5,166480
	olist_products_dataset.csv	380,237917
	olist_sellers_dataset.csv	2,434769
	product_category_name_translation.csv	0,001046
Average Execution Time (s)		1425,965735

TABLE XI. Final Test Results in the Binary Search and Counting Sort

Algorithm	Data Attribute	Average Execution Speed (s)
Binary Search - Counting Sort	*olist_customers_dataset.csv*	0,060148
	olist_geolocation_dataset.csv	3,908850
	olist_order_items_dataset.csv	1923,851715
	olist_order_payments_dataset.csv	0,3280410558
	olist_order_reviews_dataset.csv	0,168864915
	olist_orders_dataset.csv	0,000124
	olist_products_dataset.csv	0,000048
	olist_sellers_dataset.csv	0,007654
	product_category_name_translation.csv	0,000107
Average Execution Time (ms)		214,2583947

D. Discussion

Based on the results of the five comparisons that have been conducted, it can be concluded that Hash Search is

the search algorithm with the best overall performance. Although Jump Search showed the fastest execution time in some initial tests, Hash Search excels in terms of wider attribute on large and diverse datasets. Binary Search also proved to be efficient, especially when combined with Counting Sort, but its use is limited to only a few attributes. Considering speed, stability, and flexibility, Hash Search was chosen as the best search algorithm for the final implementation.

Meanwhile, in the sorting algorithm category, Counting Sort proved to be the fastest, especially when combined with Binary Search. However, Counting Sort has limitations in terms of flexibility because it is only suitable for use on data with a certain range of values. On the other hand, Merge Sort showed more stable and consistent performance across various dataset conditions. Therefore, Counting Sort is suitable for use on data that matches its characteristics for maximum efficiency, while Merge Sort is the superior choice for more general and versatile sorting needs.

REFERENCES

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[2] Olist, "Brazilian E-Commerce Public Dataset." Retrieved from: https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce

[3] P. A. Rahayuningsih, "Analisis Perbandingan Kompleksitas Algoritma Pengurutan Nilai (Sorting)," *Jurnal Evolusi*, vol. 4, no. 2, pp. 64–69, 2016. Retrieved from: https://evolusi.bsi.ac.id

[4] M. E. Al Rivan, "Perbandingan Performa Kombinasi Algoritma Pengurutan Quick-Insertion Sort dan Merge-Insertion Sort," *Annual Research Seminar*, vol. 2, no. 1, pp. 6–10, 2016.

[5] M. Lamont, "The sorting algorithms." Retrieved from: http://linux.wku.edu/~lamonml/algor/sort/index.html. Accessed: May. 21, 2025, 19:00

[6] Sonita and F. Nurtaneo, "Analisis Perbandingan Algoritma Bubble Sort, Merge Sort, dan Quick Sort dalam Proses Pengurutan Kombinasi Angka dan Huruf," *Jurnal Pseudocode*, vol. 2, no. 2, pp. 75–80, 2015.

[7] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. Boston, MA, USA: Addison-Wesley, 1998.Z

[8] R. Sedgewick and K. Wayne, "*Algorithms*", 4th ed., Boston, MA, USA: Addison-Wesley, 2011.

[9] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "*Data Structures and Algorithms*", Boston, MA, USA: Addison-Wesley, 1983.

[10] M. T. Goodrich and R. Tamassia, "*Algorithm Design and Applications*", Hoboken, NJ, USA: Wiley, 2015.

[11] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani, "*Algorithms*", New York, NY, USA: McGraw-Hill, 2008.

[12] J. L. Bentley, "*Programming Pearls*", 2nd ed., Boston, MA, USA: Addison-Wesley, 2000.