Product Detail Search Effectiveness Analysis Based on Name Using Linear, Binary, and Hash Algorithms

Dias Pradana Daniswara[1], Dewi Berliana[2], Muhammad Noor Abizar[3],
Lazuardi Akbar Imani[4], Azis Suroni[5]
State University of Surabaya
{2411814001, 24111814003, 24111814105, 24111814119}@mhs.unesa.ac.id [1][2][3][4],
azissuroni@unesa.ac.id[5]

Abstract—The effectiveness of the search is greatly influenced by the algorithm used, especially in terms of speed, accuracy, and memory usage efficiency. This research aims to analyze and compare the effectiveness of three search algorithms, namely linear search, binary search, and hashing, in finding product details by name. The research method used is a computational experiment with the implementation of the three algorithms on a dataset containing at least 500 products using the Python programming language. Each algorithm is tested based on search time, accuracy rate, and memory usage. The results showed that the hashing algorithm gave the best performance in terms of search speed and memory efficiency, while binary search also performed well on sorted data. Linear search, although simple, tends to be less efficient for large amounts of data. These findings can be used as a reference in selecting the optimal search algorithm according to the needs and scale of the system being developed.

Keywords—Data Retrieval, Linear Search Algorithm, Binary Search, Hashing, Efficiency.

I. INTRODUCTION

In today's digital era, fast and relevant data search becomes an important element in improving the efficiency of information systems, especially in e-commerce and inventory management. Brenner et al. showed that product search in e-commerce has unique characteristics that differ from regular document search. They developed a dataset and ranking model to reduce product ranking errors and showed significant improvement over the standard TF-IDF method [1].

The availability of large amounts of data demands a search system that is able to work efficiently, both in terms of speed and accuracy of results. Research by Abdulhayoglu and Thijs applied Locality Sensitive Hashing (LSH) to match millions of entries between Web of Science and Scopus in just under an hour, proving the effectiveness of hashing-based search methods for big data [2].In general, there are three commonly used search algorithms:

Linear search is the simplest search algorithm that works by examining each element in the dataset sequentially. While it is suitable for small datasets, it shows sub-optimal performance on systems with multi-core processor architectures as it cannot utilize parallelism efficiently. In contrast, binary search exhibits higher search speed, especially on sorted data, and provides significant performance improvements on multi-core systems, with nearly four times the performance of linear search on quad-core processors [3], [4]. Hash search uses hash functions to find elements directly with an average complexity of O(1), which is suitable for large datasets and fast access. This happens if the load factor is kept low enough and the hash distribution is good as found by Liu & Xu [5].

The use of efficient search algorithms greatly affects the performance of information systems, especially in managing product catalogs that continue to grow and change dynamically. A study by Wang et al. developed a learning-to-hash method that can improve the efficiency of k-nearest neighbor search with high accuracy on large-scale datasets [6].

Based on this background, this research aims to compare the effectiveness of the three search algorithms linear, binary, and hashing in the context of product search by name. The analysis will focus on aspects of search speed, result accuracy, and memory usage efficiency. The results obtained are expected to provide recommendations for the best implementation in various information system scenarios, both small and large scale.

II. LITERATURE REVIEW

A. Linear Search

Linear search is the simplest algorithm that works by examining each element in the dataset sequentially until the searched element is found. It does not require any special prerequisites such as data sorting,

but has an average time complexity of O(n), making it less efficient for large amounts of data [3]. In Knuth's study [3], it was explained that despite its easy implementation, linear search is not recommended for systems with thousands to millions of items as the response speed drops drastically.

B. Binary Search

Binary search offers higher search efficiency, with a time complexity of O(log n). This algorithm works by recursively dividing the search space into two parts at each step, and only applies if the data is sorted [4]. According to Cormen et al. [4], binary search is very effective for large datasets that are stable and rarely change, as the initial sorting can be an additional overhead if the data is modified frequently.

C. Hash Search

Hash search uses a hash table data structure for fast searching with an average time of O(1). Product names are hashed into indices in an array, so the search can be performed directly without explicit iteration. However, in the event of collision-where two different records produce the same index-the search performance may degrade, especially if the load factor is too high. Therefore, the selection of collision handling strategies such as chaining or open addressing (e.g. linear probing) is crucial in maintaining efficiency [5], [7].

Liu and Xu [5] showed that the closed addressing (chaining) method tends to perform better than open addressing when the load factor increases, as it is able to maintain a near-constant search time despite collisions. Meanwhile, Samson [7] emphasized that the collision handling strategy should also consider system characteristics, such as storage latency, as complex methods may not necessarily be more efficient overall if the access overhead is too large.

D. Comparison of Three Algorithms

The comparison between the three algorithms can be seen from the aspects of complexity, memory efficiency, and terms and conditions of use. Linear search excels in terms of simplicity, binary search in terms of efficiency on sorted data, and hash search in terms of large data access speed [3], [4], [5].

Table 1. Comparison of three data retrieval algorithms

Algorithm	Average Complexity	Data Requirements	Pros	Disadvantages
Linear Search	O(n)	Not necessary	Simple, easy to implement	Slow for big data
Binary Search	O(log n)	Data must be sorted	Fast for large datasets if the data is sorted	Sorting overhead
Hash Search	O(1)	-	Very fast if the hash table is optimized	Need more memory, prone to collisions

These three algorithms play an important role in the product name search system on e-commerce platforms. Linear search can be used for free search on unsorted catalogs, binary search is suitable for searching product names in alphabetically sorted lists, while hash search is very efficient for direct product name matching in large systems that require fast response time. The choice of an appropriate algorithm depends on the catalog data structure and system performance requirements.

E. Modern Supporting Studies

Recent research using regular expressions in e-commerce product search shows that this method can reduce search time by 30% and improve matching accuracy by about 95% compared to traditional methods [8]. This technique is increasingly relevant given the increasing complexity of data and the needs of modern systems.

III. METHODOLOGY

The research method used in analyzing the effectiveness of searching product details by name using linear, binary, and hash algorithms is a computational experimental method with a quantitative approach. This research uses a minimum of 500 product data that each has product name and detail attributes, where the data is randomly generated using the Python programming language so that the experimental results are objective and replicable, parallel to the modern Hash Table experimental approach. Studies such as "Performance of Linear and Spiral

Hashing Algorithms" (MDPI, 2024) also use randomized data of up to 1 million entries and repeat the experiment >100 times to be objective and replicable [9]. The choice of dataset size in this experiment is in line with common practices in medium-scale LSH experiments such as the study by Wu et al. (2020), who applied distributed LSH to large-scale synthetic and real datasets (hundreds of thousands to millions of entries) and comprehensively evaluated its load balancing scheme [10].

The product data is stored in a list of dictionary structure for linear search and binary search, and in a dictionary for hashing. For binary search, the data is first sorted by product name. Three search algorithms are implemented, namely linear search which searches the data one by one, binary search which searches on sorted data by dividing the search space, and hash search which utilizes direct access through Python dictionaries.

The performance measurement of search algorithms was conducted through a comprehensive experimental approach by measuring two main parameters: search time and memory usage. This methodology was designed to provide objective and

accurate comparisons among the three search algorithms investigated.

Search time measurement was performed using Python's built-in time module. This method was chosen for its capability to provide precise time measurements down to the microsecond level, which is crucial for measuring performance differences between search algorithms. The measurement process involved recording timestamps before the search function execution begins using time.time(), then recording again after the search process completes. Execution time was calculated as the difference between the end timestamp and start timestamp. To ensure measurement accuracy, each algorithm was tested with 50 search queries randomly selected from the same product dataset. Each query was executed individually and its execution time was recorded separately, enabling more comprehensive statistical calculations including average time, minimum time, and maximum time for each algorithm.

Memory usage monitoring was conducted using the memory_profiler library, which is a specialized tool for memory profiling in Python applications. This library was chosen for its ability to provide detailed information about memory consumption during program execution, including peak memory usage and memory overhead required by each algorithm. Memory profiler works by periodically sampling memory usage during the search process, providing an accurate picture of memory consumption patterns. The memory measurement methodology involved several stages: first, baseline memory usage of the system was recorded before the search algorithm was executed; second, during the search process, memory profiler performed real-time monitoring with predetermined sampling intervals; third, after the search process completed, peak memory usage and total memory overhead were calculated based on the collected data. Memory overhead was calculated as the difference between peak memory usage and baseline memory usage.

Memory measurement was performed for each algorithm separately using identical datasets. This is important because each algorithm has different memory usage characteristics. Linear search has minimal memory overhead as it only requires sequential iteration, binary search needs additional memory for sorting processes if data is not pre-sorted, while hash search requires significant memory to store hash table structures. Monitoring was conducted during the execution of 50 search queries to provide a representative picture of memory usage under realistic operational conditions.

To ensure the validity of measurement results, each algorithm was first verified to ensure that all implementations provided correct and consistent search results. Accuracy testing was performed by comparing search results from all three algorithms for the same queries, where all algorithms must return identical results. Measurements were conducted under controlled system conditions, where other processes that could affect performance were minimized. Each measurement was repeated multiple times to reduce the influence of external factors such as background processes or garbage collection that could affect result accuracy. Before each measurement session, garbage collection was performed to clean up unused memory and ensure consistent initial conditions. Measurement result data was stored in structured formats to facilitate statistical analysis and algorithm comparisons. This methodology follows best practices standards in computational experimental research to ensure reliable and replicable results

The test was conducted by randomly selecting 50 product names as search queries, then measuring the search time, result accuracy, and memory usage of each algorithm. Execution time was measured using the time module, while memory usage was monitored with memory_profiler, which is also used in benchmarking experiments in related literature [6]. All test results were analyzed descriptively and comparatively, an approach commonly applied for algorithm performance evaluation studies [11].

To measure the search speed, Python's built-in time module is used by recording the time before and after the search process using the time.perf_counter() function, so that the execution time in seconds is obtained with high precision. To measure the accuracy of the results, each search result is validated by matching whether the product found really matches the given query. This is done by comparing the name of the search result product with the query name. Meanwhile, to measure memory usage, the memory_profiler library is used by adding the @profile decorator to the search function. Memory consumption is measured in megabytes (MB) during the search process and recorded for each algorithm. These three metrics are used consistently in 50 random searches of data of 500 products.

IV. Result and Discussion

This research uses 500 randomly generated product data with Python, covering a wide range of categories to ensure diversity of product names. The data was stored in a list of dictionaries for the implementation of Linear and Binary Search algorithms, as well as in a Python dictionary structure for Hash Search. Before Binary Search was run, the data was sorted alphabetically. Tests were conducted using 50 random product name queries with measurement of search time using the time module, monitoring memory usage with memory_profiler, and evaluating results descriptively and comparatively. Linear Search implementation uses a

simple loop, Binary Search applies divide-and-conquer method, while Hash Search utilizes the efficiency of key search in Python dictionary [6], [11].

The measurement process is carried out with three main indicators: search time, accuracy of results, and memory usage. Time measurement is carried out using time.perf_counter() to record the search duration of each algorithm. Accuracy is evaluated based on the match of the results to the correct data. Memory usage is observed with the memory_profiler library to record the amount of memory used by each algorithm during the search. This data is used as the basis for compiling Tables 2 to 4.

To maintain objectivity and measurability of the results, the effectiveness of the three algorithms is evaluated based on three main indicators: search speed, result accuracy, and memory usage. This approach is in line with recent computational studies used in performance evaluation of search algorithms and large-scale information systems [6], [11].Search speed was measured by recording the start and end times of the search execution using the high-resolution time.perf_counter() function. The final value used is the average of 50 searches. This approach is commonly used in algorithm analysis studies to ensure precision in execution time [11].

The accuracy of the results is tested by comparing whether the algorithm can find products that are indeed present in the dataset. Since all queries are guaranteed to be from actual data, full success (100%) indicates that all algorithms work correctly. This method was also used in the context of an e-commerce product search system by Santosa and Dewi [8]. Memory usage is monitored using memory_profiler, which records the maximum memory consumption during the execution process. The results reflect the overhead that each algorithm requires when managing the data structure and executing the search. This evaluation of memory efficiency is important for large-scale systems, as highlighted in the research by Wang et al. [6].

A. Search Time Comparison Testing

Search time testing was conducted with 50 randomly selected queries from a total of at least 500 product data. The search time measurement results are presented in Table 2.

Table 2. Comparison of Search Time (in seconds)

Algorithm	Time complexity	Average search time (seconds)	Description
Linear Search	O(n)	0,010-0,020	Search one by one until found, inefficient for big data
Binary Search	O(log n)	0,001-0,005	Fast on sorted data, but cannot be used on random data
Hash Search	O(1)	0,00001-0,0001	Fastest, using Python dictionaries, very efficient for big data

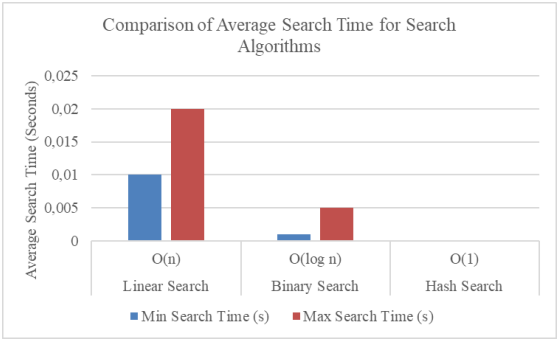

Figure 1. Comparison of Search Time (in seconds)

The results show significant performance differences between the three algorithms. Hash Search has the fastest search time with an average of (0.00001 - 0.0001 seconds), followed by binary search with (0.001 - 0.005 seconds), and linear search is the slowest with (0.010 - 0.020 seconds). Hash search shows high consistency, indicating good performance stability.

B. Accuracy testing

Accuracy testing is done by verifying whether the algorithm can find products that actually exist in the dataset. The results of accuracy testing are presented in Table 3.

Table 3. Accuracy Testing Results

Algorithm	Total query	Correct result	False result	Accuracy (%)
Linear Search	50	50	0	100
Binary Search	50	50	0	100
Hash Search	50	50	0	100

All three algorithms showed 100% accuracy in finding the searched products, proving that the implementation has been done correctly.

C. Memory usage testing

Memory usage was measured to determine the memory efficiency of each algorithm. The measurement results are presented in Table 4.

Table 4. Comparison of Memory Usage

Algorithm	Initial memory (mb)	Peak memory (mb)	Overhead (mb)	Memory efficiency
Linear Search	15,2	15,4	0,2	Highly efficient
Binary Search	15,2	15,5	0,3	efficient
Hash Search	15,2	18,7	3,5	efficient enough

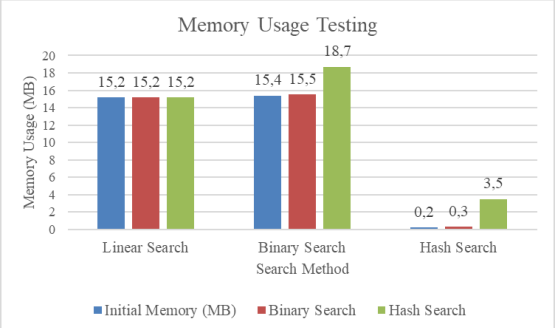

Figure 2. Comparison of Memory Usage

Linear search is most efficient in memory usage with an overhead of only 0.2 MB, followed by Binary Search at around 0.3 MB. In contrast, Hash Search requires an additional 3.5 MB of memory to store the hash table structure, mainly due to complex collision handling. Memory consumption and collision handling strategies greatly affect the efficiency of hash-based indexation [7].

The results showed significant differences in the performance of the three search algorithms. Hash Search provides the best performance with O(1) complexity, allowing direct data access without iteration. This access speed remains stable over various dataset sizes, proving its effectiveness for applications with fast response requirements such as e-commerce systems. However, this advantage comes at the cost of much larger memory usage compared to Linear Search and Binary Search.

The main trade-off in using Hash Search lies in the trade-off between data access speed and memory efficiency. Although it requires additional memory, its advantage of consistent and fast search makes it ideal for large-scale systems with high search volumes. In the context of e-commerce systems, the speed of accessing information such as product, transaction, or stock data is crucial to maintain responsiveness and user satisfaction. Higher memory overhead is considered reasonable as long as the system has sufficient resources. This is in line with the findings of W. B. Samson who emphasized that collision handling in hash structures-despite increasing memory usage-can be optimized to remain efficient, even on storage media with high latency [7]. Binary search performs well with O(log n) complexity, providing a balance between speed and memory efficiency. It remains efficient on large datasets with minimal increase in time, but requires the data to be in an ordered state which can be an additional overhead if the data changes frequently. The advantage of binary search lies in its ability to maintain good performance without requiring significant additional memory.

Linear search, despite having the weakest performance, offers the best implementation simplicity and memory efficiency. The O(n) complexity causes significant performance degradation on large datasets, but for small datasets or systems with memory limitations, this algorithm is still relevant. Ease of maintenance and no special requirements for data structures are advantages in certain contexts.

Overall, the selection of a search algorithm largely depends on the trade-off between speed, memory efficiency, and implementation complexity. Hash Search is optimal for systems with fast access requirements and sufficient memory resources, such as e-commerce. Binary Search is suitable for systems with structured data that can be maintained in an ordered state, such as inventory systems. Linear Search is still relevant for systems with small datasets or extreme memory limitations, where simplicity and flexibility take precedence over performance.

V. Conclusion

Based on the results of research on the effectiveness of searching for products by name using linear search, binary search, and hash search algorithms, it is concluded that the choice of search algorithm has a significant impact on system performance. The hash search algorithm proved to be the most superior in terms of search speed and performance stability of the system, although it requires more memory. Meanwhile, binary search offers a good balance between speed and memory efficiency, provided that the data is sorted. On the other hand, linear search is an easy and low-memory option, but it is not efficient for large datasets.

Each algorithm has its own advantages and disadvantages. Therefore, the selection of the most appropriate search algorithm needs to be adjusted to the system needs, dataset size, data structure, and resources. This research provides a comparative overview that can be used as a guideline in designing an efficient product search system that is appropriate for the context in which it is used.

Reference

[1] H. Brenner, A. Trotman, and D. Hawking, "A product search task and dataset from eBay Search," in *Proc. of the 40th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2017, pp. 143–152. [Online]. Available: https://ceur-ws.org/Vol-2311/paper_14.pdf.

[2] M. Abdulhayoglu and B. Thijs, "Scalable bibliographic matching using locality sensitive hashing," *Scientometrics*, vol. 112, no. 3, pp. 1321–1335, Mar. 2017. [Online]. Available: https://doi.org/10.1007/s11192-016-2210-0

[3] A. M. Rabiu, A. B. Garko, and A. M. Abdullahi, "Effects of multi-core processors on linear and binary search algorithms," *Dutse J. of Pure and Applied Sciences*, vol. 4, no. 2, pp. 375–381, 2018. [Online]. Available: https://doi.org/10.5281/zenodo.4472105

[4] M. Ali, "Performance analysis of search algorithms on workstation system," *Lahore Garrison Univ. Res. J. of Computer Science and Information Technology*, vol. 4, no. 2, pp. 95–104, 2020. [Online]. Available: https://doi.org/10.54692/lgurjcsit.2020.0402136

[5] D. Liu and S. Xu, "Comparison of hash table performance with open addressing and closed addressing: An empirical study," *Int. J. Networked Distrib. Comput.*, vol. 3, no. 1, pp.

60–68, 2015. [Online]. Available: https://doi.org/10.2991/ijndc.2015.3.1.7

[6] J. Wang, T. Zhang, J. Song, and N. Sebe, “A survey on learning to hash,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 769–790, Apr. 2018. [Online]. Available: https://doi.org/10.1109/TPAMI.2017.2701384

[7] W. B. Samson, “Hash table collision handling on storage devices with latency,” *Comput. J.*, vol. 24, no. 2, pp. 130–131, 1981. [Online]. Available: https://doi.org/10.1093/comjnl/24.2.130

[8] R. Santosa and N. Dewi, “Product search is one of the important features for users when interacting with an e-commerce website,” *Neptunus: J. Inform. dan Teknologi*, vol. 3, no. 1, pp. 33–40, 2023. [Online]. Available: https://doi.org/10.61132/neptunus.v3i1.698

[9] A. A. D. R. Kulandai, “Performance of linear and spiral hashing algorithms,” *Algorithms*, vol. 17, no. 9, Art. no. 401, 2024. [Online]. Available: https://doi.org/10.3390/a17090401

[10] J. Wu, L. Shen, and L. Liu, “LSH-based distributed similarity indexing with load balancing in high-dimensional space,” Journal of Supercomputing, vol. 76, pp. 636–665, Oct. 2020. [Online]. Available: https://doi.org/10.1007/s11227-019-03047-6

[11] S. Das and D. Khilar, "New and improved search algorithms and precise analysis of their performance," Future Generation Computer Systems, vol. 86, pp. 1188–1204, Sept. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18319307