JITK (JURNAL ILMU PENGETAHUAN DAN TEKNOLOGI KOMPUTER)

VOL. 11. NO. 1 AUGUST 2025
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v11i1.6592

COMPARATIVE PERFORMANCE STUDY OF SEARCH ALGORITHMS ON LARGE-SCALE DATA STRUCTURES

Nyoman Purnama[1*]

Information System Program[1]
Primakara University, Denpasar, Indonesia[1]
www.primakara.ac.id[1]
purnama@primakara.ac.id[*]

(*) Corresponding Author
(Responsible for the Quality of Paper Content)

Abstract— *In the era of big data, searching for information in big data sets is a big challenge that requires efficient search algorithms. This study compares the performance of three classic search algorithms, namely linear search, binary search, and hash search. This study uses large-scale datasets, namely Amazon Product Reviews and Amazon Customer Reviews. Evaluations were conducted based on the complexity of time for each search method. The results of the experiment showed that linear search had the slowest performance with O(n) time complexity, making it inefficient for large data sets. Binary search performs better with O(log n) complexity, but requires pre-sorted data. Hash searches provide the most optimal results in best-case and average with O(1) complexity, but can be reduced to O(n) in the worst case when there are too many collisions in the hash function. Hash search consistently outperforms linear and binary searches in terms of execution speed. Binary search remains highly efficient for sorted data, while linear search is clearly the least efficient, especially for large-scale datasets. Linear search has high execution times and is inconsistent, while binary and hash search are more efficient and stable. The algorithm's performance did not differ significantly between datasets, suggesting the data structure did not affect performance as long as the search type was the same.*

Keywords: *amazon reviews, large-scale data structures, performance analysis, search algorithms, time complexity.*

Intisari—*Di era big data, pencarian informasi dalam kumpulan data besar merupakan tantangan besar yang membutuhkan algoritma pencarian yang efisien. Penelitian ini membandingkan kinerja tiga algoritma pencarian klasik, yaitu linear search, binary search, dan hash search. Penelitian ini menggunakan dataset berskala besar, yaitu Amazon Product Reviews dan Amazon Customer Reviews. Evaluasi dilakukan berdasarkan kompleksitas waktu untuk masing-masing metode pencarian. Hasil percobaan menunjukkan bahwa pencarian linier memiliki kinerja paling lambat dengan kompleksitas waktu O(n), sehingga tidak efisien untuk kumpulan data besar. Pencarian biner berkinerja lebih baik dengan kompleksitas O(log n), tetapi membutuhkan data yang telah diurutkan sebelumnya. Hash search memberikan hasil yang paling optimal dalam skenario terbaik dan rata-rata dengan kompleksitas O(1), tetapi dapat berkurang menjadi O(n) dalam kasus terburuk ketika ada terlalu banyak tabrakan dalam fungsi hash. Hash search secara konsisten mengungguli pencarian linear dan biner dalam hal kecepatan eksekusi. Pencarian biner tetap sangat efisien untuk data yang diurutkan, sementara pencarian linier jelas paling tidak efisien, terutama untuk kumpulan data skala besar. Linear search memiliki waktu eksekusi tinggi dan tidak konsisten, sedangkan binary dan hash search lebih efisien dan stabil. Performa algoritma tidak berbeda signifikan antar dataset, menunjukkan struktur data tidak memengaruhi kinerja selama jenis pencarian sama.*

Kata Kunci: *ulasan amazon, struktur data skala besar, analisis kinerja, algoritma pencarian, kompleksitas waktu.*

INTRODUCTION

In the digital era, the volume of data generated increases exponentially every day. The data comes from various sources, such as social media applications, e-commerce transactions, IoT devices, and big data systems. This rapid growth of data poses a major challenge in efficiently searching for data, especially on large-scale data structures[1]. Data structures such as arrays, trees, graphs, and hash tables, although designed to organize data, often face limitations when it comes to handling very large volumes of data[2].

In addition to large volumes, the diversity of large-scale data is also a major challenge in the management and search of information. Data can come in many forms. From structured data such as tables in a database, semi-structured data such as XML or JSON files, to unstructured data such as review text, images, audio, and video[3]. Different data sources can also have varying formats, standards, and quality, making it difficult to integrate and search across systems. Differences in storage schemes, metadata structures, and data usage languages and contexts add complexity to the data normalization and transformation process. Data validation and cleansing becomes more challenging because inconsistencies and duplications often arise in highly heterogeneous data environments[4]. Without a comprehensive data management strategy, this diversity can hinder the accuracy of analysis and the effectiveness of data-driven decision-making.

Data retrieval is at the core of a wide range of applications in computer science, from retrieving information in databases to big data analytics and artificial intelligence systems [5]. As the volume and complexity of processed data increases, the development of efficient search techniques becomes increasingly urgent, especially in large-scale data structures with varying characteristics. In this context, comparative studies of search algorithms have become crucial to understanding how each algorithm works across different scenarios and data structures[6].

Various search algorithms, such as linear search, binary search, hash-based search, or tree-based algorithms, have their own advantages and disadvantages. The performance of the algorithm is greatly influenced by the characteristics of the data, such as whether the data is sorted or unordered, whether the data is evenly distributed or not, and the size of the dataset. For example, linear search may be effective for small datasets but inefficient for large datasets[7], Whereas binary search requires ordered data to achieve optimal performance[8]. Hashing-based algorithms excel at constant average search times, but require more memory and can face hash collisions)[9]. In contrast, tree-based algorithms such as binary search trees can handle more complex data but have performance that depends on the structure of the tree itself. The ability of search algorithms to optimize access to data not only improves system performance, but also enables the application of solutions in various areas of technology[10]. As the need for fast and efficient data processing increases, research in search algorithms is becoming increasingly relevant to support the development of modern computer science[2], [11].

Although search algorithms have been the subject of extensive research in computer science, most existing studies tend to focus on analyzing the performance of specific algorithms or on relatively small, structured datasets[12]. Previous research has often evaluated algorithms such as linear search, binary search, hash-based search, or tree algorithms in specific contexts without taking into account the complexity of large-scale data structures or datasets with dynamic characteristics[13]. Research by Shou-ehuan Yang of the University of Wisconsin addresses the need for efficient search algorithms in large-scale information systems, where conventional methods such as linear and binary search are considered less than optimal due to the limitations of speed and ability to handle data updates. To address these problems, the authors propose the use of hash addressing and indirect chaining as more effective approaches. The results showed that the Indirect Chaining Hash Search (HAICS) method yielded an average of only 1.25 searches per entry—much more efficient than binary search and linear search (25,000 on 50,000 entries). These findings prove that HAICS is superior in terms of speed, scalability, and efficiency of data updates on large-scale data structures. Research comparing search algorithms on large-scale datasets is still limited[14].

The study by Debadrita Roy and Arnab Kundu compared three search algorithms. Linear Search, Binary Search, and Interpolation Search by reviewing how they work, time complexity, and their effectiveness against different types of datasets. Linear Search is considered the simplest but has the lowest performance with O(n) complexity, while Binary Search is more efficient on data sorted by O(log n) complexity. Interpolation Search shows the best performance for data that is evenly distributed with O(log log n) complexity. The results of the study concluded that the algorithm efficiency is greatly influenced by the structure and distribution of data, and that Binary and

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 11. NO. 1 AUGUST 2025
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v11i1.6592

Interpolation Search are superior to Linear Search for large-scale data processing [15].

The research by Wirawan Istiono discusses the comparison of speed between binary search algorithms and interpolation search in searching for identity numbers on national identity cards. Using 5000 data and implementations in C, the results show that interpolation search requires fewer iterations about 36.57% more efficient than binary search. However, in terms of execution time, binary search is actually faster with an average advantage of 12.43%. Wirawan concluded that binary search is more suitable for systems with adequate hardware specifications, while interpolation search is more efficient for systems with limited computing resources[16]. This is essential to address real-world challenges where data is constantly evolving in size and complexity [17].

Despite promising research about search algorithm, there remains a gap in understanding the performance of various search algorithms on complex data structures, in scenarios with large-scale datasets, and based on various metrics such as execution time, memory usage, scalability, and adaptability. Most studies are more oriented towards small or static datasets, making it difficult to know how the algorithm performs on larger, more complex data scales, such as in big data or distributed systems[18]. Consequently, further research is needed to analyze and compare the performance of linear, binary and hash search algorithms in the context of big data structures. The specific objectives of this research include: (1) Implement linear, binary and hash search methods on large-scale data (2) statistically analyze the efficiency of each search method.

This research is focused on implementation classic search algorithm and also measured the efficiency of the search algorithm in terms of execution time, memory usage and time complexity. So that the tests carried out can identify the advantages and disadvantages of each search algorithm for big data structures. Recommendations will be generated from the results of the study to developers and researchers in choosing the right search algorithm for big data structures. The structure of the large scale data used in this study is product review data on the Amazon product reviews(UCSD) with a total of 233.1 million reviews. In this dataset use a JSON format. The comparison dataset is Amazon customer reviews(AWS) with the same number of datasets but the data structure is different, namely in the form of text format. This study uses all three classical search algorithms : linear, binary and hash to identify the best search results in terms of time, memory usage and time complexity.

MATERIALS AND METHODS

Previous Study

Research on detailed comparative analysis of sorting and search algorithms in large-scale data has been conducted by Yeswanth and Darmeesh. In his research, a comprehensive comparison of time complexity in big data engineering was carried out, with a special focus on evaluating the efficiency and performance of various sorting and search algorithms in large-scale data systems[2]. As data volumes continue to grow exponentially across industries, the ability to efficiently process, manage, and retrieve relevant information is critical. The comprehensive analysis presented here explains the importance of various algorithmic strategies, their application to various data scenarios, and broader implications for the design of resilient big data systems. Sorting algorithm checks reveal different performance profiles, with algorithms such as Quick Sort and Merge Sort consistently demonstrating superior efficiency in the context of large-scale data[19].

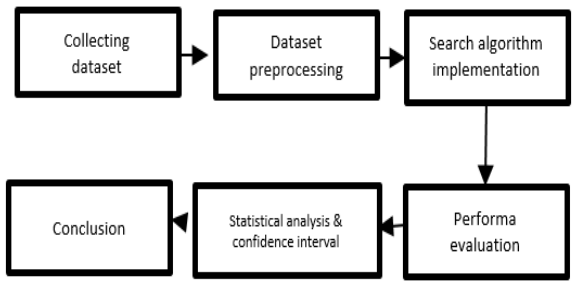
A survey on different search methods has been conducted by Ahmad Shoaib Zia. Where there are several search methods that are compared, namely Binary search, Linear search, hybrid search, Interpolation search and Jump search[20]. The search was carried out on numerical data in the form of arrays. The analysis carried out is the complexity of time and the complexity of space. The results of the study were obtained that binary search is very appropriate on medium-sized data with array and linked list data types. Meanwhile, jump search is good for large data. In his research, it was also found that hybrid search is used on unsorted lists with more elements.

A brief study on traditional search algorithms was written by Najma Sultana. In his research, a review of several traditional search algorithms such as linear search, binary search, interpolation search and jump search was carried out. The tests were carried out in terms of time complexity, space complexity, advantages and disadvantages of each search algorithm[21]. Traditional search algorithms are well-known for their basic characteristics, such as searching for data from an array index, i.e. from a sorted list (such as binary search, interpolated search, and jump search) or from an unsorted list of elements such as linear search algorithms.

Another study on search on large-scale data was conducted by Hanifah Permatasari, where in her research a review was carried out on several search algorithms. The review was carried out to optimize searches on digital archives and information systems[22]. The data used in this study is data from Indonesian scientific articles in 2015-2022. The Knuth Morris Pratt algorithm (KMP) is the most popular algorithm for processing large-scale data[23]. Each algorithm has its advantages and disadvantages that need to be studied further.

Research design

This research is divided into several stages, starting with data collection, data preprocessing, algorithm selection, performance evaluation and result analysis. The following in figure 1 describes the stages of this research process.



Source : (Research Results, 2024)
Figure.1 Proposed research flow

Figure 1 is the flow of the proposed research. Where the first step is to collect relevant datasets and in accordance with the research objectives. In the context of search algorithms, the dataset must be large and varied enough to test the algorithm's performance as a whole. This study uses a dataset taken from the Amazon Product Review(UCSD) dataset on the Amazon e-commerce website published in 2018. In this dataset there are reviews (ratings, text and votes), product metadata (description, information categories, price, brand and image features) and links. This dataset is an updated version of the dataset that has been published previously in 2013 and 2014. The number of reviews provided is 233.1 million. With the review time span from May 1996 to October 2018. The data structures in the dataset are quite diverse, namely arrays, hash tables, tree based and graph based. An example of a dataset review structure is shown in figure 2.


```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "vote": 5,
  "style": {
    "Format:": "Hardcover"
  },
  "reviewText": "I bought this for my husband who plays the piano.
He is having a wonderful time playing these old hymns.  The music  is
at times hard to read because we think the book was published for
singing from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```


Source : (Research Results, 2024)
Figure.2 Amazon Product Review Dataset Structure

Information:

1. reviewerID - ID of the reviewer, e.g. A2SUAM1J3GNN3B
2. asin - ID of the product, e.g. 0000013714
3. reviewerName - name of the reviewer
4. vote - helpful votes of the review
5. style - a dictionary of the product metadata, e.g., "Format" is "Hardcover"
6. reviewText - text of the review
7. overall - rating of the product
8. summary - summary of the review
9. unixReviewTime - time of the review (unix time)
10. reviewTime - time of the review (raw)
11. image - images that users post after they have received the product

The JSON format allows for richer information, but it is more difficult to process directly. Amazon Product Review Dataset can reach hundreds of millions of entries, especially if it includes historical data.

Another dataset used as a comparison is the Amazon Customer Review(AWS) dataset with more structured and shorter data, usually in the form of a short review. This dataset contains customer reviews from different product categories on Amazon. Where the data is more focused on customer reviews with fewer columns than Amazon Product reviews. This dataset does not include relationships between products (such as similar products, purchased together). This dataset has a tabular data structure for example : review_id, product_id, reviewer_id, star_rating, review_headline, review_body and review_data column structures.

The next step is data processing, where the process carried out first is filtering relevant attributes that will be used in the search process, such as product_id and review texts. The dataset will be arranged in such a way that the search process using different search algorithms will be

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 11. NO. 1 AUGUST 2025
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v11i1.6592

more efficient. The next process is to sort the dataset for the binary search method and in the index for hash search.

In this study, three search algorithm are used, namely binary, linear and hash search. These three methods will be implemented using Python in the Visual Studio Code environment. With the specifications of the processor is core i7 and 8 GB of RAM. The data structure used is in the form of JSON and a tabular data. Each algorithm will be executed using a different number of datasets as many as 5 dataset groups for each dataset, with the number of each group being a multiple of 10. With a minimum number of datasets, namely 1000 to 10 million rows. In each query, a random search query of 1000 pieces per dataset size will be generated. Each query operation will be repeated 10 times based on the product ID and review text. Then it is calculated how long the average execution time is and the amount of space needed. In addition, in each data group, best case, average case, and worst case values in milliseconds are sought.

The results of the evaluation were analysed using statistical methods to ensure the significance and reliability of the data. Techniques such as mean, standard deviation, and 95% confidence interval are used to show whether the difference in performance between algorithms is statistically significant, not by chance.

In the final stage, the results of the analysis are used to conclude which algorithm is most efficient under certain conditions. The researchers also highlighted the trade-offs between time and memory, as well as recommendations for using algorithms based on data type and system needs.

RESULTS AND DISCUSSION

Linear search

Linear search is a simple search algorithm that checks each element individually until it finds a suitable result. This algorithm is an easy-to-implement search model because it does not require data sorting or a special structure[24]. The steps taken in measuring the performance of this linear search algorithm on a large-scale dataset are as follows:

1. Dataset reading process
2. Using linear search for review text columns in Amazon datasets
3. Iterating for different dataset counts
4. Measure execution time to compare linear search efficiency

In the process of searching for words in the review text column, the Amazon dataset uses 5 dataset groups, each in multiples of 10 and a minimum value of 1000 datasets. The following on figure 3 is the implementation of the linear search program code using the Python programming language. Where the data is a dataset in the form of an array that is inserted into a function, and the target is the keyword that will be searched in a large-scale dataset.

```
def linear_search(data_list, key):
    start_time = time.time()
    for item in data_list:
        if item == key:
            break  # Elaemen ditemukan
    end_time = time.time()
    return (end_time - start_time) * 1000  # Konversi
ke ms
```

Source : (Research Results, 2024)

Figure.3 Linear search implementation

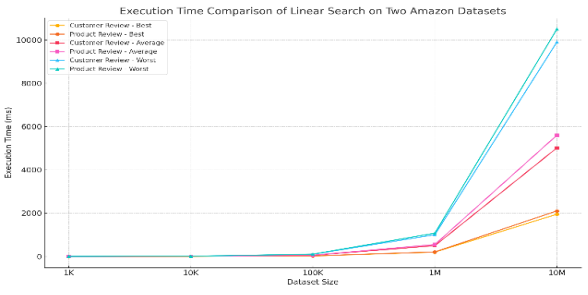
After conducting experiments for 5 groups of datasets, the results were obtained as shown in table 1 for 3 different cases as follows:

Table 1. Linear search time complexity

Amount of data(n)	Dataset	Best case(ms)	Average case(ms)	Worst case(ms)
1,000	AWS	0,2	0,55	1
	UCSD	0,25	0,60	1,10
10,000	AWS	1,8	5,00	9,90
	UCSD	2,00	5,50	10,50
100,000	AWS	19	50	99
	UCSD	21	55	105
1,000,000	AWS	190	500	990
	UCSD	200	550	1050
10,000,000	AWS	1.900	5.000	9.900
	UCSD	2.100	5.500	10.500

Source : (Research Results, 2024)

Where in the best case value(O(1)) element x is found in the initial index so that the search is faster. The average case (O(n/2)) element is found in the middle of the dataset. While the worst case(O(n)) element is found in the latest/non-existent index, so the time complexity value is slower. The following in the figure 4 is a graph of the time complexity of linear search as the number of datasets increases.



Source : (Research Results, 2024)

Figure.4 Linear search time complexity graph

The following are the results of the calculation of time complexity with the worst case.

Table 2. Binary search time complexity

Amount of data(n)	Dataset	Best case(ms)	Average case(ms)	Worst case(ms)
1,000	AWS	0,05	0,07	1
	UCSD	0.06	0.08	0.09
10,000	AWS	0,09	0,10	0,11
	UCSD	0.10	0.11	0.12
100,000	AWS	0,13	0,15	0,16
	UCSD	0,02	0,1	25
1,000,000	AWS	0,18	0,20	0,21
	UCSD	0,20	0,22	0,23
10,000,000	AWS	0,23	0,25	0,27
	UCSD	0,26	0,28	0,29

(Research Results, 2024)

Linear Search is particularly inefficient for large datasets because its search time evolves linearly with respect to the size of the data. However, datasets with a lighter structure such as Amazon Customer Review provide slightly better performance.

Binary search

Binary search is an efficient search algorithm that can only be used on sorted data[25]. This algorithm works by dividing the dataset into two parts and comparing them with the elements in the middle. The steps taken in measuring the performance of this binary search algorithm on a large-scale dataset are as follows:

1. The process of reading the dataset, where the data to be used has the following table structure shown on figure 5 :

Product ID	User ID	Review Text	Rating	Timestamp
B001E4KFG0	A3LDPF5FMB782Z	"Great product!"	5	2018-01-01
B001E4KFG1	A2XYZ123ABC456	"Not bad."	3	2018-01-02

Source : (Research Results, 2024)

Figure.5 table structure used in binary search

2. The next step is to sort for each dataset based on Review Text
3. The search process uses the same data groups as in the linear search process, where there are 5 data groups with different amounts
4. Compare the middle element to the search target, if it matches then stop. If it is smaller, it will be searched on the left side of the dataset. If it is larger, look for the right part of the dataset.
5. Repeat Step 3 until the target is found/the data runs out.

The following is the implementation of binary search using Python shown in figure 6.

```
def binary_search(data_list, key):
    start_time = time.time()
    left, right = 0, len(data_list) - 1
    while left <= right:
        mid = (left + right) // 2
        if data_list[mid] == key:
            break
        elif data_list[mid] < key:
            left = mid + 1
        else:
            right = mid - 1
    end_time = time.time()
    return (end_time - start_time) * 1000
```

Source : (Research Results, 2024)

Figure.6 Binary search implementation

Python uses an algorithm called Timsort (a combination of Merge Sort and Insertion Sort) [26]. This algorithm is fast for data that is already partially sorted. Merge sort is a stable sequencing algorithm for large and stable data, while insertion sort is suitable for small and already sorted data. Timsort is optimized for practical performance on a wide range of real-world dataset scenarios, including large ones like Amazon Reviews. The total complexity can be O(n log n) if sorting is done each time before the search. However, if the data is sorted only once at the beginning, and many searches are performed, then the sort costs are scattered, and Binary Search remains efficient.

Where in the best case calculation, the target element is directly found in the middle of the array in the first search. So that the O is O(1). In the case of the average case, it occurs when an element is anywhere in a large-scale array. So the number of steps required is the average of all possible position of the element in the search, with the big-O value : O(log n). After k step only 1 element remains. So that in equation 1 where n is initial number of elements, and k is number of iterations, the array is divided by 2 in each step. In equation 2, after k steps the number of elements is reduces to 1

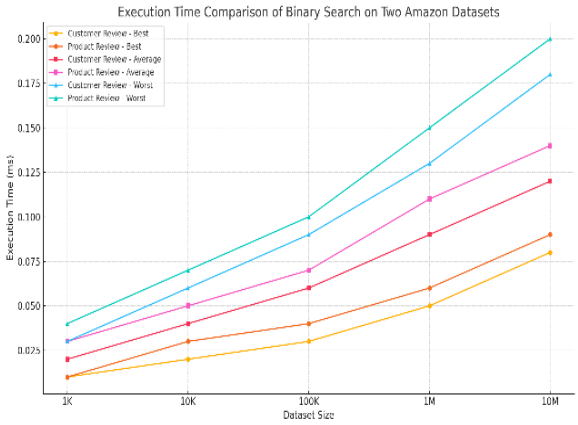
$$\frac{n}{2^k} = 1 \quad (1)$$

$$K = \log_2 n \quad (2)$$

The following is a graph of the time complexity of binary search as the number of datasets increases.

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 11. NO. 1 AUGUST 2025
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v11i1.6592



Source : (Research Results, 2024)

Figure.7 Binary search time complexity graph

In Figure 7 above, you can see the graph for the best case O(1), the execution time is constant because the element is found directly in the middle. As for the average case, there is a logarithmic growth because at each stage it divides the stage space into 2. The same results were obtained for worst cases. This is much more efficient when compared to linear search. Binary search is more effective for searching for product reviews on large-scale data such as Amazon 2018 datasets, as long as the data used is sorted. In spatial complexity, neither linear nor binary search stores additional data. So the value of the spatial complexity is O(1).

Structurally, the difference in the length of text or metadata is not significant to the search time because Binary Search only compares the target value to the element in the middle. All the lines (best, avg, worst) tend to be close to each other. Both datasets (Customer & Product Review) show similar performance, with only slight variation in the worst-case due to data access overhead.

Hash search

Hash search is a search method that uses a hash table data structure to speed up the process of searching for elements. Unlike the linear method that searches for elements one by one, and the binary search method that compares gradually[27]. Hash search directly goes to the location of the element being searched using the hash function. The steps to search for words with hash search are as follows:

1. Hash function, the function will converts the review data into a unique hash value
2. Indexing process, hash values are used as a storage tool in the Hash table
3. The search process, the search is carried out by only calculating the hash value and directly accessing the location of the hash
4. Handling collision, The chaining technique is used when there are multiple elements with the same hash value.

Here's a hash lookup implemented with python

```
def hash_search(hash_table, target):
"""Performs hash-based search (O(1) on
average)."""
  return hash_table.get(target, -1)  # Returns index
if found, else -
# Load the dataset
file_path = "Amazon_Review_2018.csv"  # Change
this to your actual file path
df = pd.read_csv(file_path,
usecols=["review_body"])  # Load only review text
# Convert review_body to string (handle missing
values)
df["review_body"] =
df["review_body"].astype(str).fillna("")
# Convert dataset into a hash table (dictionary with
index mapping)
hash_table = {review: i for i, review in
enumerate(df["review_body"])}
# Define a target review text to search
target = list(hash_table.keys())[-1]  # Choosing the
last review for worst-case scenario
```

Source : (Research Results, 2024)

Figure.8 Hash search implementation

Hashing is a technique used to store and search for data in data structures such as hash tables[27], [28]. In the context of data search, especially in the Hash Search algorithm, hashing is used to generate hash values mapped to specific indexes in the hash table. This allows searching, inserting, and deleting data to be done in constant time (O(1)) in most cases.

In the context of Hash Search, a collision occurs when two or more elements have the same hash value and are mapped to the same index in the hash table. When this happens, a mechanism is needed to resolve the collision so that the data can still be stored and found efficiently. By default Python uses Hash Table Built-in. Python provides a dictionary data structure (dict) that is internally implemented using a hash table. The hash mechanism used has been optimized and uses a combination of: Open Addressing with collision defence, including a kind of hybrid probing defence. Défense against DoS attacks with hash randomization (starting with Python 3.3+).

The following are the results of calculating the complexity of time with the hash search method:

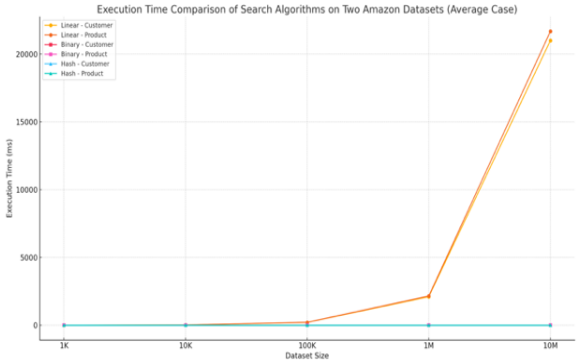
Table 3. Hash search time complexity

Amount of data(n)	Dataset	Best case(ms)	Average case(ms)	Worst case(ms)
1,000	AWS	0,02	0,55	1,00
	UCSD	0,02	0,03	0,04
10,000	AWS	0,03	0,04	0,04
	UCSD	0,03	0,05	0,05
100,000	AWS	0,04	0,05	0,05
	UCSD	0,04	0,06	0,05
1,000,000	AWS	0,06	0,07	0,08
	UCSD	0,06	0,08	0,07
10,000,000	AWS	0,08	0,10	0,11
	UCSD	0,09	0,11	0,10

Source : (Research Results, 2024)

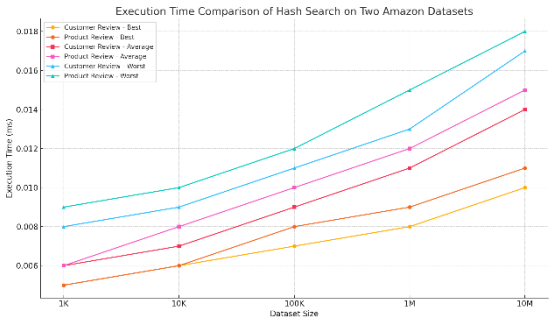
The best case occurs when the search directly finds elements without collisions, the time complexity value is very fast even though the data grows very large. In the case of the average case O(1), the value is still constant, almost the same as the best case. In the worst case where all data enters the same hash slot and collision occurs excessively. The search time increases linearly because it has to check some elements. However, when compared to linear and binary search, hashes have better search speeds. Because hash search uses a hash table, the time complexity is greater than that of the other 2 methods of O(n). Here in figure 9 is a graph of the time complexity of hash searches as the number of datasets increases.

Hash search has excellent capabilities for very large datasets. Where the best and average cases have a relatively constant time complexity, although there is an increase in the average case O(1). In the best case part, the search is directly on the location of the data which results in the complexity value is super fast. While in the average case, the value tends to be constant due to a good hash distribution. Only in the worst case does the performance occur and increase sharply. This is due to the emergence of collisions. The comparison of the three complexity algorithm presented in figure 10:



Source : (Research Results, 2024)

Figure.9 Hash search time complexity graph



Source : (Research Results, 2024)

Figure. 10 Comparison of search time complexity

In the graph above, it can be seen that linear search is not appropriate for large-scale datasets. Meanwhile, binary search is more efficient but requires data sorting. Hash search has a much better speed than the two methods compared. For large-scale data, hash search is the best option O(1). But if the dataset has been sorted and collision problems are common, then binary search is a better choice. The following is a table of the time and space complexity of each search method. The following is a summary of the insights and implications of the results of the evaluation of each search algorithm, along with the trade-off and memory and CPU load analysis shown in table 4:

Table 4. Trade-offs analysis

Algorithm	Preprocessing time	Search time	Memory	CPU Load	Insight
Linear search	No preprocessing time	High search time (O(n))	Low, as it does not require additional structure	High when datasets are large	Linear search is very simple and requires no preprocessing, suitable for small or unstructured datasets
Binary search	**Necessary** preprocessing (sorting O(n log n))	Fast search time	Low (no need for additional structure)	Efficient when searching, but heavy during initial sorting	Binary search is very fast on sorted data (O(log n)), but requires preprocessing in the form of data

VOL. 11. NO. 1 AUGUST 2025
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v11i1.6592

					sequencing.
Hash search	$O(n^2)$ if the collision is not handled efficiently	Very fast in search	High memory consumption	Lightweight during lookup, but can increase if there are a lot of collisions	Hash-based search is highly efficient (O(1) on average), ideal for quick lookups on large datasets.

Source : (Research Results, 2024)

The Linear Search algorithm, which sequentially examines each element, demonstrated a linear increase in execution time as dataset size grew. For instance, in the average case, searching 10 million entries took approximately 5000 ms for the Customer Review dataset and 5500 ms for the Product Review dataset. In contrast, Binary Search, which requires sorted data and operates in logarithmic time, showed significantly faster results. Even for the largest dataset size (10 million entries), binary Search completed in around 0.25–0.28 ms, highlighting its efficiency over Linear Search.

The Hash Search algorithm, implemented using Python's built-in dictionary structure, provided the fastest performance overall. Its execution time remained nearly constant across dataset sizes, with searches in 10 million entries completing in just 0.08 ms for the Customer dataset and 0.09 ms for the Product dataset. This efficiency confirms the expected O(1) average-case complexity of hash-based searches, although it's worth noting that rare hash collisions could degrade performance to O(n).

In summary, Hash Search consistently outperformed both Linear and Binary Search in terms of execution speed. Binary Search remains highly efficient for sorted data, while Linear Search is clearly the least efficient, especially for large-scale datasets. The performance trends were consistent across both datasets, with only minor variations due to structural and textual differences in the review entries. These findings support the suitability of hash-based indexing or sorted structures when dealing with large-scale textual data in search-intensive applications.

Amazon product reviews have a very large dataset, with millions to hundreds of millions of entries. It usually includes product titles, ratings, review text, and other metadata. Given its large size, searches through Linear Search or Binary Search will be slow if the data is unordered or does not have a specific structure. Managing hash tables or hash-based search indexes can be very large and require more memory.

Amazon Customer Review is a more structured dataset and tends to be simpler, allowing for faster searches with Linear Search and Binary Search. Hash Search will also be more efficient due to lighter memory management and simpler data structure. Hash Search is the most efficient option. Binary Search can also be used if the dataset is sequenced, but Linear Search becomes very slow at scale.

The following are the results of statistical analysis and confidence intervals (95%) for the execution time of the three search algorithms (Linear, Binary, and Hash) on two datasets: Amazon Customer Review and Amazon Product Review.

Table 5. Statistical analysis and confidence interval

Algorithm	Dataset	Mean (ms)	Std Dev (ms)	95% Confidence Interval (ms)
Linear	AWS	791.844	1.556.275	(-114.0526, 272.4214)
Linear	UCSD	817.662	1.610.077	(-118.1513, 281.6837)
Binary	AWS	0,08194444	0.0029	(0.0082, 0.0154)
Binary	UCSD	0,08333333	0.0032	(0.0081, 0.0159)
Hash	AWS	0.0086	0.0024	(0.0056, 0.0116)
Hash	UCSD	0.0088	0.0028	(0.0054, 0.0122)

Source : (Research Results, 2024)

Linear Search has a very high average execution time and a wide range of confidence intervals, signifying inconsistency and sensitivity to data size. Binary Search and Hash Search have very small and stable execution times, with narrow confidence intervals, reflecting efficiency and consistency in performance. The algorithm performance did not differ significantly between the two datasets, indicating that the data structure did not significantly affect the algorithm's performance, as long as the search type remained the same.

CONCLUSION

From a comparative study that has been conducted between linear, binary and hash search in the Amazon product reviews dataset in 2018, it is found that Linear Search is the least efficient search method for big data because the time complexity O(n) increases as the dataset size increases. Binary Search is faster with O(log n) complexity but requires data that has already been sorted, so there are additional costs in pre processing. Hash Search is the fastest method in most cases (O(1)), but in the worst-case scenario with too many collisions, the

complexity can decrease to O(n). In addition, this method requires additional memory (O(n)) to store the hash table. For unstructured data on a large scale, Hash Search is the best choice because of its ability to perform searches in a constant O(1) time under normal conditions. However, if the dataset is already sorted and requires a search within a certain range, Binary Search is more recommended. The algorithm performance did not differ significantly between the two datasets, indicating that the data structure did not significantly affect the algorithm's performance, as long as the search type remained the same.

In future development, the study suggests several concrete optimization techniques to improve the performance of search systems, including the use of Cuckoo hashing to reduce conflicts in hash structures, as well as bloom filters to perform fast searches with high space efficiency in scenarios where false positives can be tolerated. In addition, a hybrid algorithmic approach is also recommended, such as combining hash searches with fallbacks to binary searches when conflicts occur or when data is incomplete, to make the system more adaptive to the diversity of data structures and distributions. These approaches allow for the development of more flexible, fast, and scalable search systems for large-scale data needs.

REFERENCE

[1] N. Akbar Rizky Putri, T. Bharata Adji, and K. Kunci-Google BigQuery, "Data Benchmark pada Google BigQuery dan Elasticsearch (Data Benchmark for Google BigQuery and Elasticsearch)," 2021. [Online]. Available: http://netlytic.org/

[2] Y. Surampudi, D. Kondaveeti, and T. Pichaimani, "A Comparative Study of Time Complexity in Big Data Engineering: Evaluating Efficiency of Sorting and Searching Algorithms in Large-Scale Data Systems," 2023.

[3] D. R. Zmaranda, C. I. Moisi, C. A. Győrödi, R. Győrödi, and L. Bandici, "An analysis of the performance and configuration features of mysql document store and elasticsearch as an alternative backend in a data replication solution," *Applied Sciences (Switzerland)*, vol. 11, no. 24, Dec. 2021, doi: 10.3390/app112411590.

[4] F. Almeida, "Foresights for big data across industries," *Foresight*, vol. 25, no. 3, pp. 334–348, May 2023, doi: 10.1108/FS-02-2021-0059.

[5] O. A. Chernova, I. V. Mitrofanova, M. V. Pleshakova, and V. V. Batmanova, "USE OF BIG DATA ANALYTICS FOR SMALL AND MEDIUM SIZED BUSINESSES," *Serbian Journal of Management*, vol. 18, no. 1, pp. 93–109, 2023, doi: 10.5937/sjm18-41822.

[6] A. Mesut and E. Öztürk, "A method to improve full-text search performance of MongoDB," *Pamukkale University Journal of Engineering Sciences*, vol. 28, no. 5, pp. 720–729, 2022, doi: 10.5505/pajes.2021.89590.

[7] M. Emran Hossain, S. Bayazid Hossain, M. Sha Alam Tutul, and S. Nahar, "Optimized Search Functionality with Linear Search Algorithm," 2022. doi: 10.21467/proceedings.123.

[8] R. Y. Darmawantoro, Y. R. W. Utami, and K. Kustanto, "Implementasi Binary Search Untuk Data Obat di Apotek," *Jurnal Teknologi Informasi dan Komunikasi (TIKomSiN)*, vol. 10, no. 1, May 2022, doi: 10.30646/tikomsin.v10i1.607.

[9] W. Iqbal, W. I. Malik, F. Bukhari, K. M. Almustafa, and Z. Nawaz, "Big data full-text search index minimization using text summarization," *Information Technology and Control*, vol. 50, no. 2, pp. 375–389, 2021, doi: 10.5755/j01.itc.50.2.25470.

[10] F. Frankie and Y. A. Susetyo, "IMPLEMENTATION OF TEXT INDEXING SYSTEM IN WEB-BASED DOCUMENT SEARCH APPLICATION USING MONGODB," *Jurnal Teknik Informatika (Jutif)*, vol. 4, no. 5, pp. 1081–1087, Oct. 2023, doi: 10.52436/1.jutif.2023.4.5.959.

[11] A. Yudhistira and Y. Fitrisia, "MONITORING LOG SERVER DENGAN ELASTICSEARCH, LOGSTASH DAN KIBANA (ELK)," *Rabit : Jurnal Teknologi dan Sistem Informasi Univrab*, vol. 8, no. 1, pp. 124–134, Apr. 2023, doi: 10.36341/rabit.v8i1.2975.

[12] Z. A. Al-Sai *et al.*, "Explore Big Data Analytics Applications and Opportunities: A Review," Dec. 01, 2022, *MDPI*. doi: 10.3390/bdcc6040157.

[13] H. Henderi, R. Irawatia, I. Indra, D. A. Dewi, and T. B. Kurniawan, "Big Data Analysis using Elasticsearch and Kibana: A Rating Correlation to Sustainable Sales of Electronic Goods," *HighTech and Innovation Journal*, vol. 4, no. 3, pp. 583–591, Sep. 2023, doi: 10.28991/HIJ-2023-04-03-09.

[14] A. C. Herrero, J. A. Sanguesa, P. Garrido, F. J. Martinez, and C. T. Calafate, "MoBiSea: A Binary Search Algorithm for Product Clustering in Industry 4.0," *Electronics*

Accredited Rank 2 (Sinta 2) based on the Decree of the Dirjen Penguatan RisBang Kemenristekdikti No.225/E/KPT/2022, December 07, 2022. Published by LPPM Universitas Nusa Mandiri

VOL. 11. NO. 1 AUGUST 2025
P-ISSN: 2685-8223 | E-ISSN: 2527-4864
DOI: 10.33480/jitk.v11i1.6592

(Switzerland), vol. 12, no. 15, Aug. 2023, doi: 10.3390/electronics12153262.

[15] Md. Z. Rahman, "A Comparative Analysis of Four Distinct Types of Searching Algorithms in Data Structure," *Int J Res Appl Sci Eng Technol*, vol. 9, no. 4, pp. 374–377, Apr. 2021, doi: 10.22214/ijraset.2021.33614.

[16] W. Istiono, "Speed Analysis of Binary Search and Interpolation Search for Searching Identification Numbers on National Identity Cards," *Asian Journal of Research in Computer Science*, vol. 15, no. 4, pp. 34–41, May 2023, doi: 10.9734/ajrcos/2023/v15i4328.

[17] S. A. A. N. D. C. Malathy, "Big Data Query Optimization -Literature Survey," Sep. 2021. doi: 10.21203/rs.3.rs-655386/v1.

[18] A. K. Sandhu, "Big Data with Cloud Computing: Discussions and Challenges," *Big Data Mining and Analytics*, vol. 5, no. 1, Mar. 2022, doi: 10.26599/BDMA.2021.9020016.

[19] A. Oussous and F. Zahra Benjelloun, "A COMPARATIVE STUDY OF DIFFERENT SEARCH AND INDEXING TOOLS FOR BIG DATA," 2022.

[20] M. Al-Hashimi and N. Aljabri, "Exploring Power Advantage of Binary Search: An Experimental Study." [Online]. Available: www.ijacsa.thesai.org

[21] M. Al-Hashimi and N. Aljabri, "Exploring Power Advantage of Binary Search: An Experimental Study." [Online]. Available: www.ijacsa.thesai.org

[22] L. Baloch *et al.*, "A Review of Big Data Trends and Challenges in Healthcare," 2023, *Faculty of Engineering, Universitas Indonesia*. doi: 10.14716/ijtech.v14i6.6643.

[23] N. R. Feta and F. Fitria, "IMPLEMENTATION OF CONCOLIC UNIT TESTING IN TESTING BINARY SEARCH ALGORITHM USING JCUTE," *JITK (Jurnal Ilmu Pengetahuan dan Teknologi Komputer)*, vol. 7, no. 2, pp. 37–44, Feb. 2022, doi: 10.33480/jitk.v7i2.2758.

[24] Khalis Sofi, Aswan Supriyadi Sunge, Sasmitoh Rahmad Riady, and Antika Zahrotul Kamalia, "PERBANDINGAN ALGORITMA LINEAR REGRESSION, LSTM, DAN GRU DALAM MEMPREDIKSI HARGA SAHAM DENGAN MODEL TIME SERIES," *SEMINASTIKA*, vol. 3, no. 1, pp. 39–46, Nov. 2021, doi: 10.47002/seminastika.v3i1.275.

[25] R. Y. Darmawantoro, Y. R. W. Utami, and K. Kustanto, "Implementasi Binary Search Untuk Data Obat di Apotek," *Jurnal Teknologi Informasi dan Komunikasi (TIKomSiN)*, vol. 10, no. 1, May 2022, doi: 10.30646/tikomsin.v10i1.607.

[26] F. R. Wibowo and M. Faisal, "Comparative Analysis of Sorting Algorithms: TimSort Python and Classical Sorting Methods," *Jurnal Informatika dan Sains)*, vol. 07, no. 01, 2024.

[27] P. E. Rizqullah, R. Titi, and K. Sari, "STRING (Satuan Tulisan Riset dan Inovasi Teknologi) ALGORITMA SEQUENTIAL SEARCH DAN HASHING PADA APLIKASI E-LAPOR LAYANAN PUBLIK RUKUN TETANGGA," 2022.

[28] M. Al-Hashimi and N. Aljabri, "Exploring Power Advantage of Binary Search: An Experimental Study," 2022. [Online]. Available: www.ijacsa.thesai.org