JCBI
Journal of Computing & Biomedical Informatics
ISSN: 2710 - 1606
ICASET
2024

Research Article
Collection: Intelligent Computing of Applied Sciences and Emerging Trends

Empirical Analysis of Quaternary and Binary Search

Muhammad Nauman Saeed[1*], Mutiullah Jamil[1], Zoha Iqbal[3], Rimsha Tariq[1], Maria Kanwal[2], Abiha Ejaz[4], Syed Ali Nawaz[1], and Abdul Waheed[2]

[1]Institute of Computer Science, Khwaja Fareed University of Engineering and Information Technology, Rahim Yar Khan, 64200, Pakistan.
[2]Department of Information Technology, Institute of Southern Punjab, Multan, Pakistan.
[3]Department of Artificial Intelligence, The Islamia University of Bahawalpur (IUB), Bahawalpur, 63100, Pakistan.
[4]Department of Information Technology, The Islamia University of Bahawalpur (IUB), Bahawalpur, 63100, Pakistan.
[*]Corresponding Author: Muhammad Nauma Saeed. Email: naumansaeed0077@gmail.com

Academic Editor: Salman Qadri **Published:** April 01, 2024

Abstract: This paper presents an empirical analysis between traditional binary search and quaternary search algorithm in sorted arrays. Binary search algorithm divides the list into two equal halves each time, whereas quaternary search technique divides it into four segments each time. An experiment was conducted with an objective to search the element which is causing the worst time complexity in case of sorted array. It was observed through experiment results that iterative binary search is better than recursive binary search, iterative quaternary search is better than recursive quaternary search and definitely iterative quaternary search is better than iterative binary search. So, the evaluation results prove the best time complexity of binary search algorithm is O(log2 n) and quaternary search algorithm has the best time complexity of O(log4n) and therefore, it's decided that quaternary search algorithm is better in searching as compared to binary search algorithm.

Keywords: Quaternary search; Binary search; Empirical analysis.

1. Introduction

Searching in sorted data structures is one of the key problems in theoretical computer science. In one of its most basic variations, the objective is to discover a particular element of a sorted set by creating queries that iteratively slight the possible locations of the preferred element. A search method is a process that decides the subsequent query to be posed based at the final results of preceding queries [1].

Search algorithms are significant and usually utilized in most computer systems. Searching for an item in an arranged array is an effective operation in information processing. There are numerous search algorithms supposed for sorted arrays located inside the literature. The binary search and interpolation search are the most conventional search algorithms used to examine an already sorted array.

The binary search algorithm is a famous example utilized in courses such as data structures and algorithms to determine the logarithmic time complexity search algorithm [2], finding a value in an arranged array of items. An arranged array is occupied, as the selected data structure for key searches. In each iteration, the central item of an interval is tested [3], and if it isn't the desired key, half(1/2) of the array where the key cannot lie is eliminated, and the search continues at the ultimate half. It is repeated till either the key's discovered, or the remaining 1/2 is empty, which means that it is not inside the array. Since binary search itself does not essentially imply implementing an equal splitting criterion, we call the usual technique as overhead the ordinary binary search.

Quaternary search is like binary search but divides the array into four parts instead of two [4]. After evenly dividing the array, the three divisors are compared to the input value [5]. If it matches the index is returned. If not, the algorithm is recursively called on the subarray that contains the value.

One of the most important problems of computer science is the binary search. There are several applications of the binary search. To achieve the objective, researchers have modified the method in several aspects. Dobkin [6] has confirmed the usefulness and efficiency of binary search. Analysis of multidimensional search problems shows that the binary search is comprehensive and efficient. Zemel [7] studied overall performance relating to randomized binary search. The randomized binary search algorithm is used for discovering the global minimum of a multi-modal one-dimensional function.

Baqai [8] integrated an advanced analytical model to address dot connections in printers with the Direct Binary Search (DBS) halftoning algorithm. They proposed a streamlined approach to assess how the computational cost fluctuates during the search process. Empirical results are presented to demonstrate the effectiveness of this technique.

Binary search algorithm is a vital technique for investigation and exploration of discrete computational systems [9]. It is simply a searching method with logarithmic time complexity. It locates the position of a key in an ordered array. The need often arises in database applications to simultaneously search for multiple key elements during a single iteration. The binary search algorithm has been altered by the author to enable the search for multiple items at once. X. Li [10] studied that the tag collision is a major problem for quick tag identification in Radio Frequency Identification (RFID) systems. They presented an improved binary anti-collision search algorithm for RFID systems based on a novel binary search of backtracking [11].

2. Materials and Methods

2.1. Binary Search

The performance can be greatly improved when searching for a key in an array by first placing items in the array then sorting its items in ascending or descending order before using the binary search algorithm. In this algorithm, it first compares the search key with the item in the middle position of the array. If they match, then the search is successful. Otherwise, if the key is less than the middle item, the sought item must be in the lower half of the array; if greater, it must be in the upper half [12]. This process continues on the appropriate half until the item is found or the entire array is searched as shown in Figure 1. While binary search requires a more complex program compared to a simple linear search, for large datasets, its time complexity of O(log(n)) makes it much faster than the O(n) complexity of linear search.

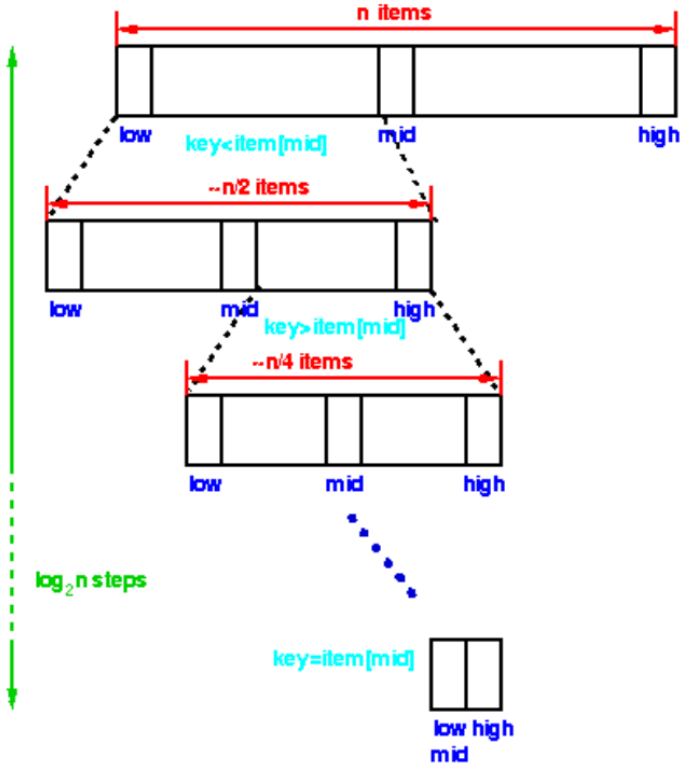

Figure 1. Working of binary search algorithm

The binary search algorithm involves dividing the block of items being searched in half at each step. Since a set of n items can only be divided in half at most $\log_2 n$ times, the running time of a binary search algorithm is proportional to $\log_2 n$ and can be expressed as an $O(\log_2 n)$.

2.2. Quaternary Search

In the proposed algorithm, significant performance improvements over binary search can be achieved by first sorting the items in an array in either ascending or descending order. This sets the stage for utilizing the quaternary algorithm, which involves calculating the middle element, one-quarter element, and three-quarter element of the array. Subsequently, the algorithm compares the search key with the items at these positions. If a match is found, the algorithm concludes successfully. However, if the key does not match any of these elements, the algorithm proceeds to check if the key is less than the middle element, the algorithm searches in the first quarter of the array [13]. If the key is greater than the middle element but less than the one-quarter element, the algorithm searches in the second quarter. If the key is greater than the one-quarter element but less than the three-quarter element, the algorithm searches in the third quarter. Otherwise, the algorithm searches in the fourth quarter. This process continues on the selected quarter until the key is found or the array is exhausted as shown in Figure 2. If the key is not found after examining all quarters, the algorithm concludes that the key is not present in the array.

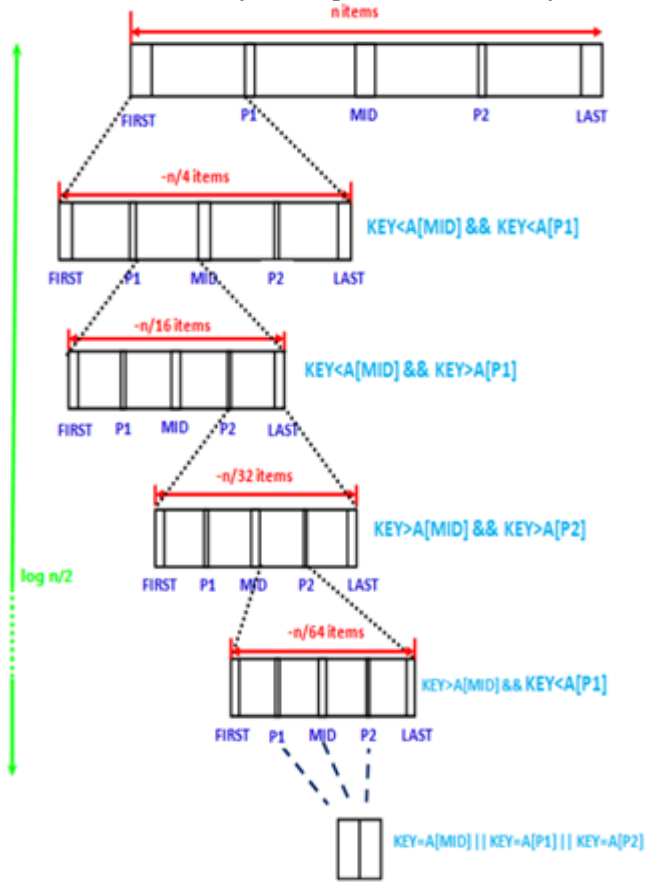

Figure 2. Working of quaternary search algorithm

The quaternary search algorithm involves dividing the array of items being searched into four parts. This division can occur at most $\log_4 n$ times for a set of n items. Thus, the running time of a quaternary search algorithm is proportional to $\log_4 n$, which can be expressed as an $O(\log_4 n)$ [14].

3. Results

A sorted array containing 1000 elements was utilized for the experiment. The values stored in the array are uniformly distributed with random numbers. Both the Binary Search and Quaternary Search algorithms were applied to search for the same elements that took the worst time for the algorithms to find. The results of both algorithms are presented in Figure 3, along with the number of steps each algorithm took, compared to their respective values of comparison.

Elements in Array	Size	Key	Binary Search			Quaternary Search		
			Recrsive		Iterative	Recrsive		Iterative
			Invocation	Time (ms)	Time (ms)	Invocation	Time (ms)	Time (ms)
12,13,15,23,27,48,55,76,90	10	55	2	0.018	0.011	1	0.013	0.013
12,13,15,23,27,48,55,76,90,94,97,135,147,167,168,185,192,194,195	20	185	4	0.042	0.010	2	0.020	0.014
12,13,15,23,27,48,55,76,90,94,97,135,147,167,168,185,192,194,195,207,223,227,231,233,235,238,278,284,290	30	233	4	0.039	0.015	2	0.031	0.013
12,13,15,23,27,48,55,76,90,94,97,135,147,167,168,185,192,194,195,207,223,227,231,233,235,238,278,284,290,297,304,315,326,332,334,341,353,368,369	40	90	6	0.053	0.011	4	0.041	0.011
12,13,15,23,27,48,55,76,90,94,97,135,147,167,168,185,192,194,195,207,223,227,231,233,235,238,278,284,290,297,304,315,326,332,334,341,353,368,369,372,386,393,407,414,416,421,435,445,472	50	238	5	0.118	0.016	3	0.025	0.010
12,13,15,23,27,48,55,76,90,94,97,135,147,167,168,185,192,194,195,207,223,227,231,233,235,238,278,284,290,297,304,315,326,332,334,341,353,368,369,372,386,393,407,414,416,421,435,445,472,475,479,492,501,504,510,518,536,542,555	60	326	4	0.027	0.008	4	0.029	0.010
12,13,15,23,27,48,55,76,90,94,97,135,147,167,168,185,192,194,195,207,223,227,231,233,235,238,278,284,290,297,304,315,326,332,334,341,353,368,369,372,386,393,407,414,416,421,435,445,472,475,479,492,501,504,510,518,536,542,555,568,581,596,599,609,614,620,626,642,644	70	555	6	0.054	0.009	3	0.037	0.010
12,13,15,23,27,48,55,76,90,94,97,135,147,167,168,185,192,194,195,207,223,227,231,233,235,238,278,284,290,297,304,315,326,332,334,341,353,368,369,372,386,393,407,414,416,421,435,445,472,475,479,492,501,504,510,518,536,542,555,568,581,596,599,609,614,620,626,642,644,681,700,713,763,765,775,790,791,799,800	80	334	4	0.042	0.008	4	0.031	0.010
12,13,15,23,27,48,55,76,90,94,97,135,147,167,168,185,192,194,195,207,223,227,231,233,235,238,278,284,290,297,304,315,326,332,334,341,353,368,369,372,386,393,407,414,416,421,435,445,472,475,479,492,501,504,510,518,536,542,555,568,581,596,599,609,614,620,626,642,644,681,700,713,763,765,775,790,791,799,800,814,831,840,842,867,872,877,882,888,914	90	867	4	0.043	0.013	4	0.029	0.017
12,13,15,23,27,48,55,76,90,94,97,135,147,167,168,185,192,194,195,207,223,227,231,233,235,238,278,284,290,297,304,315,326,332,334,341,353,368,369,372,386,393,407,414,416,421,435,445,472,475,479,492,501,504,510,518,536,542,555,568,581,596,599,609,614,620,626,642,644,681,700,713,763,765,775,790,791,799,800,814,831,840,842,867,872,877,882,888,914,921,924,930,931,935,941,974,979,985,988	100	620	6	0.102	0.013	4	0.040	0.012
		Average	4.5	0.0538	0.0114	3.1	0.0296	0.012

Figure 3. Performance of binary search and quaternary search

3.1. Comparison of Binaray Search Iterative and Recursive Approach

Figure 4 illustrates the comparison between the iterative and recursive approaches of binary search. The graph shows the relationship between the array size and the time taken to find the key in milliseconds (ms). The graph clearly indicates that the binary search iterative approach outperforms the binary search recursive approach.

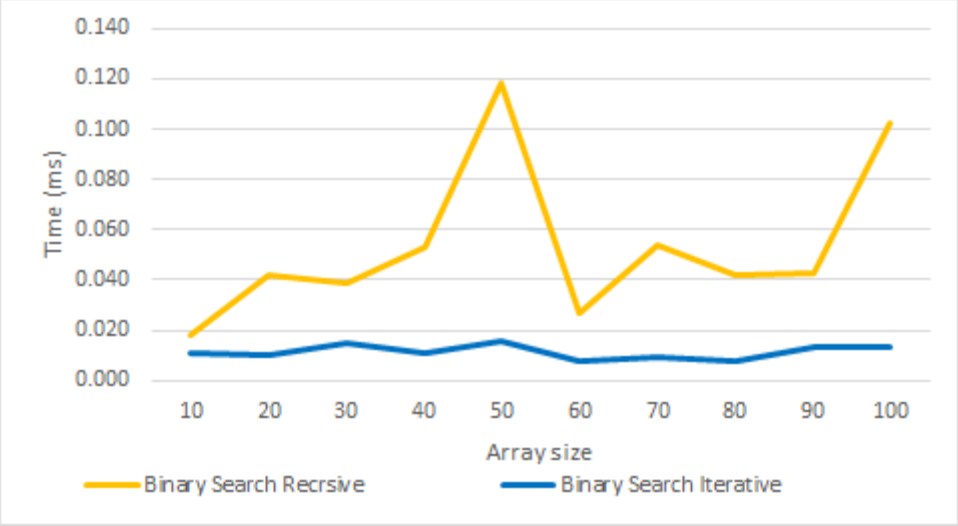

Figure 4. Comparison of binary search iterative and recursive approach

3.2. Comparison of Quaternary Search Iterative and Recursive Approach

Figure 5 depicts a comparison between the iterative and recursive methods of quaternary search. The graph presents the relationship between the size of the array and the time required to locate the key, measured in milliseconds (ms). The results demonstrate that the iterative approach in quaternary search performs better than the recursive approach.

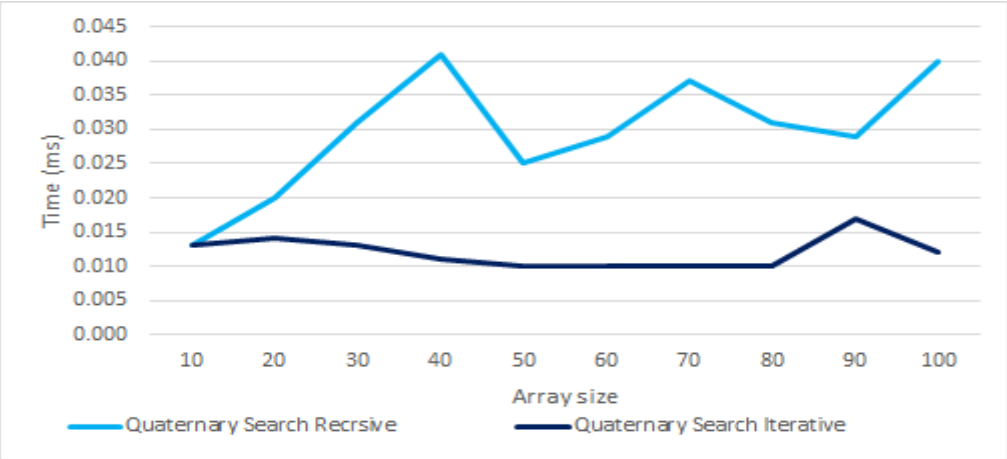

Figure 5. Comparison of quaternary search iterative and recursive approach

3.3. Performance comparison of Quaternary Search and Binaray Search

Figure 6 provides a detailed comparison between the binary search and quaternary search algorithms. The analysis is conducted on an array ranging in size from 10 to 100, comprising randomly sorted values. The graph illustrates the relationship between the array size and the time taken to locate the key, measured in milliseconds (ms). The findings highlight the considerable efficiency superiority of quaternary search over binary search, suggesting its capacity to enhance search performance across diverse applications.

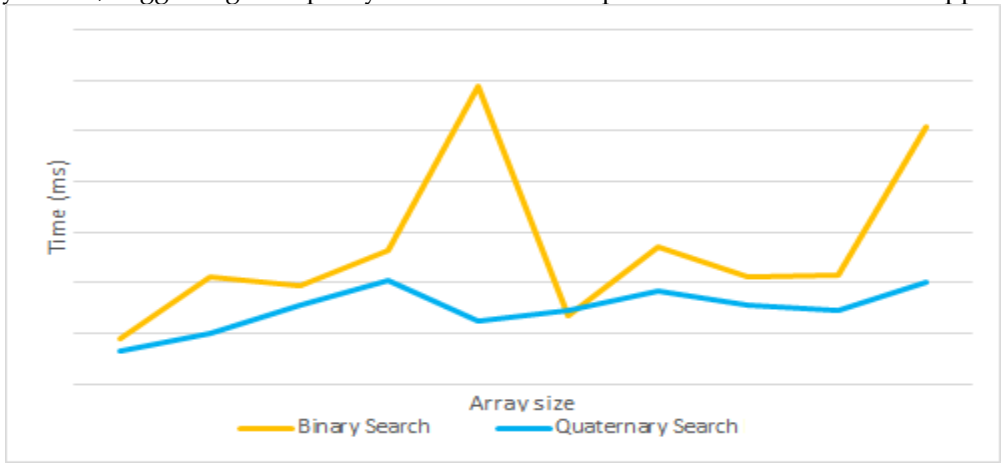

Figure 6. Performance comparison of quaternary search and binary search

4. Discussion

The study compared the performance of binary search and a proposed quaternary search algorithm for searching in sorted arrays. Both algorithms were applied to a sorted array with uniformly distributed random numbers. The worst-case time for each algorithm to find elements was recorded, along with the number of steps taken. The experimental results indicate that the binary search iterative approach outperforms the binary search approach [15]. Similarly, the quaternary search performs better than the binary search. Overall, the study suggests that the quaternary search algorithm may offer improved performance compared to the traditional binary search algorithm, particularly in scenarios involving large datasets. Further research could explore the efficiency of the quaternary search algorithm in various search contexts and dataset sizes to validate its potential advantages over binary search.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Laber, E. and M. Molinaro, An approximation algorithm for binary searching in trees. Algorithmica, 2011. 59(4): p. 601-620.
2. Morshtein, S., R. Ettinger, and S. Tyszberowicz, Verifying Time Complexity of Binary Search using Dafny. arXiv preprint arXiv:2108.02966, 2021.
3. McClellan, M.T. and J. Minker, The art of computer programming, vol. 3: sorting and searching. 1974, JSTOR.
4. Ohya, T., M. Iri, and K. Murota, A fast Voronoi-diagram algorithm with quaternary tree bucketing. Information processing letters, 1984. 18(4): p. 227-231.
5. Li, M., et al., Minimum total-squared-correlation quaternary signature sets: new bounds and optimal designs. IEEE transactions on communications, 2009. 57(12): p. 3662-3671.
6. Dobkin, D. and R.J. Lipton. On some generalizations of binary search. in Proceedings of the sixth annual ACM symposium on Theory of computing. 1974.
7. Zemel, E., Random Binary Search: A Randomizing Algorithm for Global Optimization in R^1. Mathematics of operations research, 1986. 11(4): p. 651-662.
8. Baqai, F.A. and J.P. Allebach. Printer models and the direct binary search algorithm. in Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181). 1998. IEEE.
9. Tarek, A., A new approach for multiple element Binary Search in Database Applications. International Journal of Computers, 2007. 1: p. 263-268.
10. Li, X., et al., A compact folded printed dipole antenna for UHF RFID reader. Progress In Electromagnetics Research Letters, 2009. 6: p. 47-54.
11. Ford Jr, L.R. and D.R. Fulkerson, Constructing maximal dynamic flows from static flows. Operations research, 1958. 6(3): p. 419-433.
12. Wilkinson, W.L., An algorithm for universal maximal dynamic flows in a network. Operations Research, 1971. 19(7): p. 1602-1612.
13. Kotnyek, B., An annotated overview of dynamic network flows. 2003, INRIA.
14. Hoppe, B. and É. Tardos. Polynomial Time Algorithms for Some Evacuation Problems. in SODA. 1994.
15. Dhikhi, T. T. (2019). Measuring size of an object using computer vision. International Journal of Innovative Technology and Exploring Engineering, 8(4), 424-426.