Hussein Search Algorithm: A Novel Efficient Searching Algorithm in Constant Time Complexity

Omer H Abu El Haijia[1], Arwa H. F. Zabian[2]
University of Castilla–La Mancha, Toledo-España[1]
Jadara University, Jordan-Irbid[2]

***Abstract*—Hussein search algorithm focuses on the fundamental concept of searching in computer science and aims to enhance the retrieval of data from various data warehouses. The efficiency of cloud systems is substantially influenced by the manner in which data is saved and retrieved, given the vast quantity of data being generated and stored in the cloud. The act of searching entails the systematic endeavor of locating a particular item within a substantial volume of data, and searching algorithms offer methodical strategies for accomplishing this task. There exists a wide array of searching algorithms, each exhibiting variations in terms of the search procedure, time complexity, and space complexity. The choice of the suitable algorithm is contingent upon various aspects, including the magnitude of the dataset, the distribution of the data, and the desired temporal and spatial intricacy. This study presents a novel prediction-based searching algorithm named the Hussein search algorithm. The system is designed to operate in a straightforward manner and makes use of a simple data structure. This study relies on fundamental mathematical computations and incorporates the interpolation search algorithm, an algorithm that introduces a search by-prediction method for uniformly distributed lists, it forecasts the precise position of the queried object. The cost of prediction remains consistent and, in numerous instances, falls under the O(1) range. Hussein search algorithm exhibits enhanced efficiency in comparison to the binary search and ternary search algorithms, both of which are widely regarded as the best methods for searching sorted data.**

Keywords—Binary search; prediction search procedure; prediction cost; constant time complexity

I. Introduction

On a daily basis, a substantial volume of data is generated across many formats, including photographs, videos, and text. This material is subsequently stored in cloud-based repositories, serving as a collective resource for retrieval from any given database. The increasing complexity of this matter can be attributed to the substantial volume of data that is generated and stored on a daily basis. The act of searching is of significant importance in various contexts, regardless of whether the item being sought is stored within a cloud-based infrastructure or a localized database. In both scenarios, the use of a search algorithm is essential for the successful retrieval of the desired item. The process of searching is commonly employed as a means of problem-solving, whereby the problem is provided as input and a solution is generated in the form of a sequential set of activities. Numerous instances in practical contexts can be classified as searching problems, such as the task of determining the shortest route between two nodes. These types of problems can be effectively addressed through the use of graph search algorithms. The act of searching can be categorized into two main types: sequential search and binary search. Various searching algorithms exist, each employing distinct strategies and exhibiting variations in terms of time and space complexity. Certain algorithms employ an informed approach, while others adopt a uniform approach, and a third category utilizes a partial information strategy for the purpose of item retrieval. The uniformity of the sequential search method arises from its lack of concern for any prior knowledge regarding the distribution of items. The binary search algorithm is considered an informed search algorithm due to its reliance on a sorted array. In recent years, there has been limited progress in enhancing the complexity of the search algorithm. This is primarily due to the satisfactory performance of the binary search algorithm in terms of searching complexity. However, it is important to note that the binary search algorithm does encounter challenges related to sorting, as it can only operate on sorted arrays. Additionally, it faces difficulties when dealing with comparison-based input involving searching for candidate items. Hence, the temporal complexity is intricately linked to the duration required for the sorting process, resulting in a trade-off where the time saved when searching is offset by the time invested in sorting. The interpolation search algorithm has a temporal complexity of O(1) when the items in the list are evenly distributed [1]. Therefore, in some scenarios, an interpolation search can provide an accurate estimation of the closest solution to the search problem. The ternary search method is a variant of the binary search algorithm that has a slower temporal complexity [1, 2]. The meta-binary search algorithm is a variant of the binary search method that iteratively creates the index of the desired value within the array. The approach operates in a way akin to the binary search algorithm, exhibiting a time complexity of O(log n) for locating the desired element. The ternary search technique partitions the array into three segments, utilizing the central point of each segment to locate the desired element. The logarithmic complexity of the search algorithm is determined by the number of steps required to locate the desired element, which is logarithmically proportional to the size of the array, denoted as n. However, the search range, which represents the number of elements that need to be examined during the search, is three times the size of the array (3n). Consequently, the worst-case running time of the algorithm can be expressed as the logarithm of three times the size of the array (log3n), while the best-case running time may be approximated as being linearly proportional to the size of the array (O(n)). It is important to note that these running time estimates are valid only when the array is sorted [3]. The jump search algorithm is a searching technique designed for sorted arrays. It involves performing a fixed number of jumps, denoted by k, on a block of data in order to locate the target element. Within each block, linear search operations are conducted to identify the desired element. The algorithm under consideration is superior to the sequential search method, but it falls short of the binary search technique. Specifically, it requires m-1 additional comparisons compared to sequential search, where m represents the size of the block to be traversed [4]. The interpolation search algorithm is a method that generates additional data points within a given range of known data points. Its time complexity is O(log log n) for datasets with uniform distributions, and O(n) in the worst-case scenario. The proposed approach represents an advancement over the binary search technique by employing comparison-based approaches that leverage a mathematical formula to approximate the location of the target element based on its value. Subsequently, the search is conducted in the vicinity of this estimated position. This alternative method has the potential to outperform the binary search algorithm under certain circumstances [5]. The exponential search algorithm operates on a sorted array. It begins by selecting a subarray of size 1, then doubles the size of the subarray in each iteration. The algorithm compares the final element of each subarray until the desired element is found. The algorithm in question is commonly referred to as exponential search, which exhibits a temporal complexity of O(log n) [6]. In the realm of searching for an item within an array of size n, two commonly employed strategies can be identified. The first strategy, known as sequential search, is applicable to unsorted arrays. The second technique, known as binary search, is exclusively applicable to arrays that have been sorted.

The study presented in this paper aims to introduce a novel searching algorithm that operates on a sorted array, relying solely on mathematical operations and a computed prediction approach implemented through a straightforward data structure. The search process in question exhibits a constant time complexity. While the space complexity may exceed that of sequential search, the time complexity can be lowered to O(1) in numerous scenarios and to O(constant) in the worst-case scenario. The algorithm relies on the computation of the array's average, operating under the assumption of a uniform distribution of items within the list. Additionally, it generates supplementary arrays, one of which records the frequency of successful matches, while the other stores the locations where the sought-after items can be located. Part of process is similar to the Knuth-Pratt-Morris algorithm, which is commonly used for pattern matching [7]. The approach under consideration aims to decrease the time complexity by minimizing the number of comparisons and implementing a straightforward prediction system that ensures the absence of collisions through the utilization of error-free lookup tables and basic arithmetic operations.

The primary objectives underlying this research endeavor are to provide a straightforward predictive approach utilizing search techniques and to execute a basic computation with constant time complexity.

The subsequent sections of this study are structured as follows: Section II presents a review of relevant literature pertaining to searching algorithms. It is worth noting that the process of locating sufficient recent works on searching algorithms proved to be challenging. The majority of the literature discovered consisted of dated publications or encompassed broader discussions on binary search algorithms. In Section 3, the proposed algorithm is introduced. In Section IV are shown the results obtained from the proposed study, followed by an examination of the algorithm employed. Finally, the paper concludes with findings and discusses potential avenues for further research in Section V.

II. Related Works

One of the main benefits of searching operations in computer science is their capability to assist in finding particular data from databases. The effectiveness of the search process directly impacts the overall performance of the system. Searching algorithms are widely employed in several computer applications, such as problem-solving [9], data analysis, and information retrieval, due to their ability to efficiently search through extensive datasets within a limited timeframe. Various searching techniques exist, including sequential, binary, hashing, and graph search. The selection of an appropriate algorithm is contingent upon the particular situation at hand and the attributes of the data being queried [10]. The authors of [5] introduce a hybrid search method known as interpolated binary search (IBS), which integrates the interpolation algorithm and the binary search algorithm to accurately determine the precise position of the desired object. The IBS algorithm commences by employing an interpolation algorithm to estimate the approximate location of the item being searched. It subsequently operates as a binary search algorithm to precisely determine the location of the sought-after item. The Inverse Binary Search (IBS) algorithm exhibits a greater computational cost compared to both the binary search algorithm and the interpolation technique. However, when executed on uniformly distributed data, IBS demonstrates a lower temporal complexity cost than the aforementioned algorithms. Specifically, its time complexity is O(log2log2n). On the other hand, when applied to non-uniformly distributed datasets, IBS necessitates O(log2n) operations. In [8], a comparison between different search algorithms is presented, the authors analyze the performance of various search algorithms, including uninformed search algorithms (DFS, uniform cost search) and informed search algorithms (A* and BFS), with a focus on their time complexity and space complexity.

Graph search algorithms typically construct a graph based on the given input data and traverse the nodes of the graph using various strategies in order to locate the desired objects [13]. The algorithms that were examined all exhibited a time complexity of O(mb), where m represents the number of offspring for each node (also known as the branching factor) and b represents the solution depth, which is the length of the path. The binary search algorithm utilizes a value of m equal to 2 and assigns b as the logarithm base 2 of n.

The act of searching is a crucial and widely employed process in several contexts. In order to obtain data of various types, it is necessary to carry out two fundamental procedures: searching and sorting. In order to address this concern, it is worth noting that the majority of searching algorithms operate on an array that has been sorted. However, it is important to acknowledge that the computational expense of sorting the data can vary significantly, ranging from a best-case scenario of O(n log n) to a worst-case scenario of O(n^2). This additional cost must be taken into consideration when evaluating the overall efficiency of the search process. The authors of [11] present a novel technique called the bound sequential search (BSS) algorithm, which uses logical gates to simultaneously search for two items. The primary concept is around the utilization of Binary Search with Sorted Subarrays (BSS) to concurrently search for about two items, hence eliminating the need for executing the search process twice, all without the involvement of parallel processors. In the context of Binary Search Systems (BSS), the process of looking for two items, X and Y, involves the utilization of an additional key, Z. This key, denoted as Z, is determined by the logical operation of the inclusive OR between X and Y. Subsequently, Z serves as the key element for conducting a search operation within an unsorted array, denoted as A, which possesses a size of n. The search procedure commences by evaluating each element in the array A against Z, provided that Z AND A[i] != A[i] indicates that neither of the two elements is present in A[i].

In the event of the most unfavorable scenario, the computational complexity for locating two items in the BSS (Binary Search Structure) is N, as opposed to 2N in sequential search or (n2+ log n) in the binary search algorithm. The Bubble Sort algorithm demonstrates superior performance in terms of comparison count in both the best and worst case scenarios, when compared to the Binary Search algorithm and the Sequential Search method. A limitation of this algorithm is its inability to perform a search for a single item, as it is designed to search for about two objects simultaneously.

Hashing is a technique employed in the search of extensive databases, wherein a distinct key is generated for each record or item. This key serves to designate the specific area within the database where the item is potentially kept. The time complexity for retrieving an item from a big database using hashing can range from constant time (O(1)) to linear time (O(n)) in the worst-case scenario. The variable n represents the size of the linked list, which is utilized for the purpose of storing colliding data within a single slot, denoted as [12]. The objective is to verify the presence of item X inside a given dataset. As the volume of data expands, the intricacy of the situation escalates. The problem of searching has been introduced and elucidated by Levin and Solomonoff throughout the time span of 1973 to 1984 [14,15, 16, and 17]. In reference [18], a novel quantum technique is presented for the search problem, demonstrating a polynomial time complexity. This algorithm utilizes XOR logical gates to convert the data into a polynomial form, following the amplification process provided in reference [19]. Consequently, the search operation may be executed within a polynomial time frame relative to the input size, denoted as n.

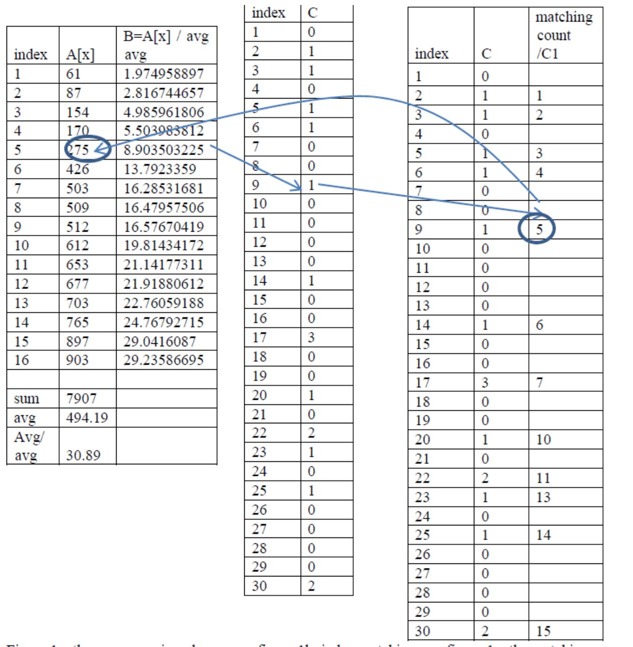
In the cited work [20, 21], the authors introduce a fractional cascade algorithm, which is a method aimed at enhancing the efficiency of binary search algorithms. This methodology achieves a reduction in the time complexity of binary search algorithms to O(k + log n) while searching for k elements within a sorted array of size n.

III. Hussein Search Algorithm

The Hussein search algorithm is designed to efficiently locate an element in a sorted array. It achieves this by utilizing a prediction table structure and a sequential search algorithm. This approach reduces the time complexity from O(n) in sequential search or O(log n) in binary search to O(1) [7]. The algorithm achieves this improvement by employing only arithmetic operations and a technique inspired by the interpolation method for searching sorted lists [1]. It is important to note that the algorithm assumes a uniform distribution of elements in the list. The algorithm operates in two distinct phases: the preprocessing phase and the searching phase. During the preparation step, the entirety of the array undergoes arithmetic operations in order to ready the data for the subsequent prediction finding procedure. During the preprocessing phase, an array of integers denoted as A, which has a size of n is examined. The data within this array is created in a random manner. During this phase, the operations conducted involve the calculation of the average value of variable A. The average can be determined by calculating the mean of a set of values. To obtain the mean, each element in array A should be divided by a given value t, and the resulting values should be stored in a new array B. The size of array B is denoted as n. Generate a novel array C, whose size is determined by the floor function applied to the value of t. To iterate through a set of counters starting from 0 to size (c), the ceiling of each element B[i] is compared with the indices of C. The number of matching values is determined, and the resulting matching score is stored in a new list C1 at the corresponding index of the matching item. The elements contained within the array C1 will represent the respective indices of the desired item within the original array. During the searching phase, the following operations are performed to determine whether an item X is present in an array: X is divided by t, and the resulting value is compared with the indices of C1 using the ceiling function. The value found in the corresponding cell of C1 represents the index of the searched item X in the original array. If the corresponding cell is empty, it indicates that the item X is not present in the array.

The following example provides a comprehensive elucidation of the functioning of the Hussein search method. In this scenario, array A of size 16 is randomly produced by [7], and the data within the array is uniformly dispersed. The elements in set A are arranged in ascending order, while the outcomes of dividing each element in set A by the average of the average are recorded in set B (refer to Fig. 1(a)). A new array, denoted as C, is to be created with a size of 30. This array will contain the floor value of the average values obtained from Fig. 1(b). The quantity of corresponding elements in the array is stored in Fig. 1(c). Now, let us explore the scenario where aim to search for item X, which has a value of 310, within the table. Initially, the value of X is divided by 30.89. The resulting quotient is then rounded up to the nearest whole number, denoted as 11. Subsequently, proceed to locate the element within the array by referring to the index position

11. Upon examination, it is determined that the value at this index is 0, indicating that the desired element is not present inside the array. Let X be equal to 275. Next, perform a division operation on X by t, and subsequently apply the ceiling function. The resulting value, 9, is assigned to the index 9 in array C. Upon further examination, we finally, observe that the element at index 9 in array C is 1, indicating that X is located at index 5 in the original array (as depicted in Fig. 2).



index	A[x]	B=A[x] / avg avg
1	61	1.974958897
2	87	2.816744657
3	154	4.985961806
4	170	5.503983812
5	275	8.903503225
6	426	13.7923359
7	503	16.28531681
8	509	16.47957506
9	512	16.57670419
10	612	19.81434172
11	653	21.14177311
12	677	21.91880612
13	703	22.76059188
14	765	24.76792715
15	897	29.0416087
16	903	29.23586695
sum	7907	
avg	494.19	
Avg/ avg	30.89	

Figure.1a: the preprocessing phase

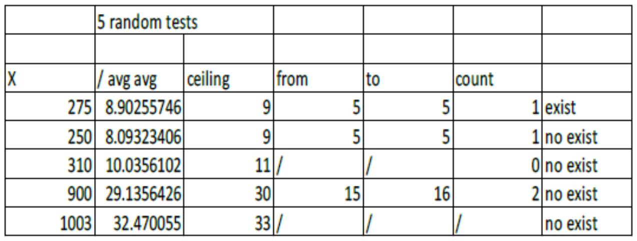
index	C
1	0
2	1
3	1
4	0
5	1
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	1
15	0
16	0
17	3
18	0
19	0
20	1
21	0
22	2
23	1
24	0
25	1
26	0
27	0
28	0
29	0
30	2

figure.1b: index matching

index	C	matching count /C1
1	0	
2	1	1
3	1	2
4	0	
5	1	3
6	1	4
7	0	
8	0	
9	1	5
10	0	
11	0	
12	0	
13	0	
14	1	6
15	0	
16	0	
17	3	7
18	0	
19	0	
20	1	10
21	0	
22	2	11
23	1	13
24	0	
25	1	14
26	0	
27	0	
28	0	
29	0	
30	2	15

figure.1c: the matching score

Fig. 1. Hussein search algorithm simulation on random data generation.



	5 random tests					
X	/ avg avg	ceiling	from	to	count	
275	8.90255746	9	5	5	1	exist
250	8.09323406	9	5	5	1	no exist
310	10.0356102	11	/	/	0	no exist
900	29.1356426	30	15	16	2	no exist
1003	32.470055	33	/	/	/	no exist

Fig. 2. Testing the search operation using Hussein search algorithm.

A. *Algorithm Description:*

1) Preprocessing phase: Consider we have an array of integers A of size n, where the data is randomly generated, in this phase the following operations are performed:

- Calculate the average of A. average $=\frac{sum(A)}{n}$
- Calculate the average of average: $t=\frac{average}{n}$
- Divide all the elements of A by t and store it in a new array B, the size of B is n
- Create a new prediction array C with size floor (t)
- For I counter that start from 0 to size (c), compare the ceiling of B[i] with the indices of C, and count the matching values and store the matching score in a new list C1 in the corresponding index to the matching item. The values stored in the array C1 will be the corresponding index of the searched item in the original array.

2) Searching phase: For searching an item X, if in the array or not is performed the following operations:

- Divide X by t, $S=\frac{X}{t}$
- Compare the ceiling of (S) with the indices of C1; the value found in the corresponding cell will be the index of the searched item (X) in the original array, if the corresponding cell is empty that means the item is not found in the array.

IV. Results and Analysis

The proposed algorithm is simple and easy to understand and implement. It is implemented and tested in MacBook air i5 processor 1.3 GHz speed, 8 GB ram, using visual studio, C# and in Python, and is tested for large input size list up to 16 mg . Furthermore, we have successfully implemented both the binary search method and the ternary algorithm using the C# programming language. These implementations were carried out in an identical setting, with the input size being consistent with that of the Hussein search algorithm. The objective of the experiment was to conduct N iterations in order to seek a randomly produced list with a size of N. The findings indicate that the speed of the prediction searching method in all evaluated algorithms exhibits a linear relationship with the amount of input. However, it is noteworthy that as the input size increases, the performance of the Hussein search surpasses that of the other algorithms. The Hussein search method exhibits a constant time complexity of O(1) for finding an individual item.

Consequently, the search operation for n items may be accomplished in linear time complexity of O(n). This stands in contrast to the binary search strategy, which necessitates a time complexity of O(n log n) for searching n items. Fig. 3 presents the outcomes achieved by the Hussein search algorithm in contrast to the other algorithms when searching for N items across varying input sizes. while considering input sizes of 8 MB and 16 MB, it has been observed that the Hussein search method exhibits a time requirement that is 20% lower than that of the binary search strategy, and 17.3% lower than that of the ternary algorithm, while searching for all items. Hussein's search method demonstrates a search speed that is approximately 494% greater than that of binary search when applied to a dataset of 16 MB. Table 1 illustrates the speedup, which quantifies the extent to which the Hussein search algorithm outperforms the binary search algorithm across various input sizes. Table I illustrates the observed increase in search speed across various input sizes.

In the Hussein search algorithm the searching process about an item requires O (1), which means searching n items requires only O(n) in comparison with the binary search algorithm that requires O(n log n). Fig. 3, show the results obtained by the Hussein search algorithm in comparison to the other algorithms for different input size. For input size (8 M, 16 M) searching all the items using the Hussein search algorithm requires time that is 20% smaller than the binary search algorithm and 17.3 % smaller than the Ternary algorithm. The search speed in the Hussein search algorithm is increased by about 494% than the binary search for 16 M of data. Table I, and Fig. 3 show the speed up that represents how much Hussein's search is faster than the binary search algorithm in searching about all the input sizes.

TABLE I. THE SPEED UP IN SEARCHING DIFFERENT INPUT SIZE

	Binary	Trenary	Hussein	Speedup
4 k	2	2	4	50%
8k	5	6	2	250%
16 k	4	6	2	200%
32 k	12	13	3	400%
64 k	27	24	5	540%
128 k	43	22	10	430%
256 k	89	109	20	445%
512 k	189	222	39	485%
1 MB	349	419	69	506%
2MB	703	825	140	502%
4MB	1398	1606	259	540%
8MB	3343	3568	629	531%
16MB	5813	6784	1176	494%

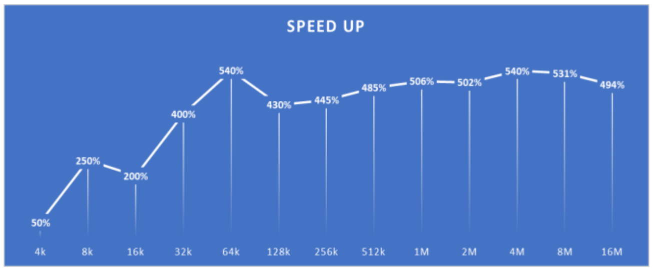
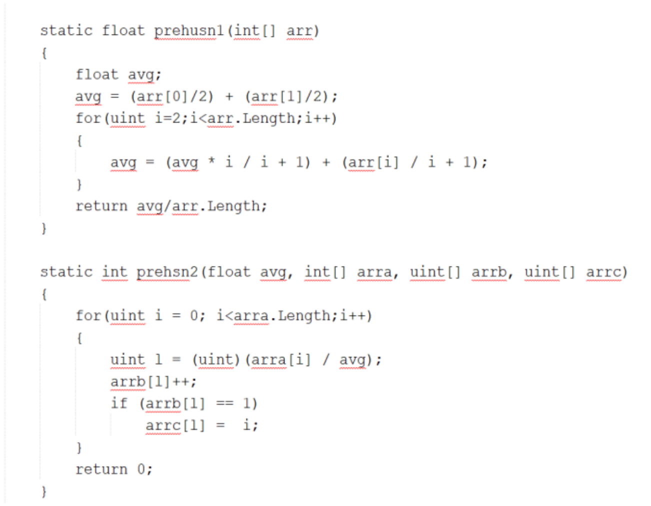


Fig. 3. Hussein search algorithm speed up.

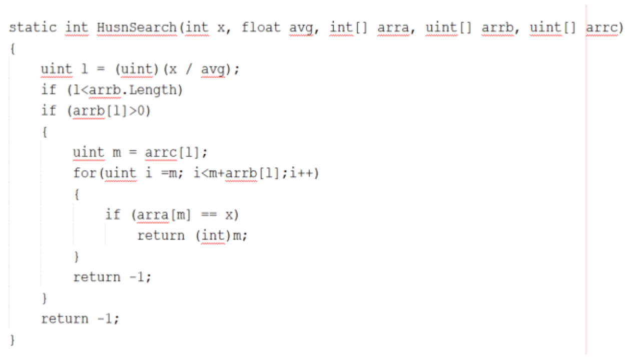


```
static float prehusn1(int[] arr)
{
    float avg;
    avg = (arr[0]/2) + (arr[1]/2);
    for(uint i=2;i<arr.Length;i++)
    {
        avg = (avg * i / i + 1) + (arr[i] / i + 1);
    }
    return avg/arr.Length;
}

static int prehsn2(float avg, int[] arra, uint[] arrb, uint[] arrc)
{
    for(uint i = 0; i<arra.Length;i++)
    {
        uint l = (uint)(arra[i] / avg);
        arrb[l]++;
        if (arrb[l] == 1)
            arrc[l] =  i;
    }
    return 0;
}
```

Fig. 4. The implementation of mathematical operations in Hussein search algorithm.

Implementation of Hussein Search Algorithm: as mentioned previously Hussein Search algorithm is implemented in C#, with binary search algorithm and ternary algorithm on the same environment and the same data. Fig. 4 and 5 represent the implementation of the main functions of the Hussein search algorithm.



```
static int HusnSearch(int x, float avg, int[] arra, uint[] arrb, uint[] arrc)
{
    uint l = (uint)(x / avg);
    if (l<arrb.Length)
    if (arrb[l]>0)
    {
        uint m = arrc[l];
        for(uint i =m; i<m+arrb[l];i++)
        {
            if (arra[m] == x)
                return (int)m;
        }
        return -1;
    }
    return -1;
}
```

Fig. 5. The implementation of the Hussein search algorithm.

V. CONCLUSION AND FUTURE WORKS

In this study, we introduce the Hussein search algorithm, a novel informed search approach that leverages a straightforward prediction method, basic arithmetic operations, and a simple data structure. The findings demonstrate that the Hussein search algorithm outperforms previous search algorithms in terms of time complexity, particularly when dealing with substantial data sets. Moving forward, there are several avenues for future research. Firstly, it would be valuable to explore the algorithm's performance under different search scenarios and input distributions. Additionally, investigating potential optimizations and further enhancements to the algorithm could yield even more efficient search capabilities. Finally, conducting comparative studies with other state-of-the-art search algorithms would provide a comprehensive evaluation of the Hussein search algorithm's effectiveness. The procedure operates on an array that has been sorted, with the underlying assumption that the data is spread equally. The Hussein search algorithm has a time complexity of O(n) for finding n items. In contrast, the binary search algorithm has a time complexity of O(n log n), while the sequential search technique requires O(nn) in the worst case. Given the assumption of a sorted array, the proposed technique offers a notable advantage in terms of computational simplicity and a reduction in the number of comparisons required for item search. Specifically, the algorithm achieves a time complexity of O(1) when searching for an item by index rather than by value. Fig. 6 presents a comparison of the running times for various input sizes. The future objective is to enhance the algorithm based on simple prediction methods to operate with the same time complexity for an unsorted array. Additionally, we aim to offer a sorting algorithm that utilizes the same mathematical processes and achieves linear time complexity in the worst-case scenario.

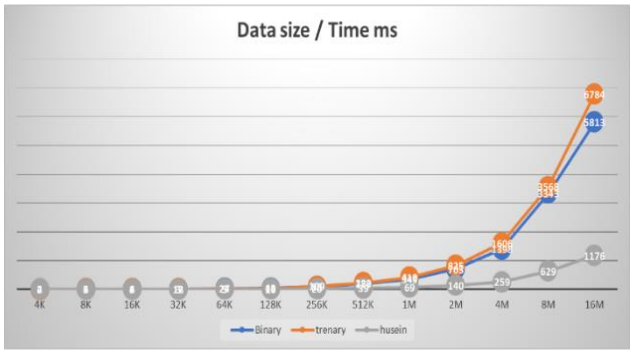


Fig. 6. Running time comparison for different input size.

REFERENCES

[1] C. Blum, A. Roli: Meta Heuristic in combinatorial Optimization: overview and Conceptual Comparison. ACM Computing Survey, vol 35, No.3, PP: 268-308.2003.

[2] Brodrick Crwford, Ricardo So et al: Putting Continuous Meta Heuristic to Work in Binary Search Spaces. Hindawi complexity. Volume 2017. https://doi.org/10.1155/2017/8404231. PP:1-19.

[3] Manpreet Singh Bajwa, Arun Prakash Agarwal, Sumati Manchanda. Tenary Search Algorithm: Improvement of Binary Search. 2nd

[4] International conference on Computing for Sustainable Global Development (INDIACOM)11-13 March 2015. New Delhi. India. Ieeexplore.org/document/7100542.

[5] Adnan Saher Mohammed, Sahin Emrah Amrahou, Fatin V. Celebi. Interpolated Binary Search : An Efficient Hybrid Search Algorithm on Ordered Datasets. Engineering Science and Technology, An International Journal. Volume 24, Issue.5, October 2021, PP:1072-1079. https://doi.org/10.1016/j.jestch.2021.02.009.

[6] Milos Simic. Exponential Search. Last modified 26 June 2022 available at: https://ww.baeldung.com/cs/exponential-search.

[7] Knuth, Donald (1998). Sorting and Searching: The art of Computer Programming. (2ned).MA. Addison- Wesley. ISBN:978-0-201-89685-5.

[8] Maharshi, J. Pthak, Romit L. Patel, Sonal P. Rmi. Comparative Analysis of Search Algorithm. International Journal of Computer Applications. Volume 179. No.50. June 2018. PP:40-43.

[9] Dhanya Thailappan. Introduction to Problem Solving Using Search Algorithms for Beginners. Last modified 26th July 2022. Analytics Vidhya. Available at: https://www.AnalyticsVidhya.com/blog/2021/an-introduction-to-problem-solving-using-search-algorithms-for-beginners.

[10] Stuart Russell, Peter Norvig. Artificial Intelligence: A Modern Approach. Pearson Education International. 4th Edition 2021. ISBN-13: 978-1292401133.

[11] Omer H. Abu El Hija, Azmi Alazzam. Bound Sequential Search (BSS). Proceedings of the Worlds Congress on engineering and Computer Science. 2012. Vol.1, WCES 2012. October 24-26/2012. San Francisco, USA.

[12] George T. Heineman, Gary Pllice, Stanley Sekow. Algorithms in a Nuthshell: A Practical Guide. 2nd Edition O'reilly Media (April, 12, 2016). ISBN-10: 1491948922.

[13] Thomas Cormen, Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. 4th Edition (2022). The MIT press. ISBN-10: 026204630X.

[14] M. R Gavey and D. S. Johnson. Computer and Intractability: A Guide to the Theory of NP-Completeness. W. H. freeman, 1979.

[15] L. A. Levin. Universal Sequential Search Problems. Problems of Information Transmission. 9(3): 265-266-1973.

[16] L. A. Levin. Randomness Conversation Inequalities. Information and Independence in Mathematical Theories. Information and Control. 61: 15-37. 1984

[17] R. J. Salmonoff. Optimum Sequential Search. Memorandum, Oxbridge Research Cambridge, Mass. June 1984.

[18] S. Iriyama, M. Ohya, and V. Velovich. On Quantum Algorithm for Binary Search and its Computational Complexity. arXiv:1306.5039v1 [quant-ph] 21 June 2013.

[19] M. ohya and I. V. Vovich. Quantum Computing and Chaotic amplifier. J.OPT.B, 5, No.6. 639-642.2003.

[20] Chazelle, Bernard, Liu, Ding. Lower Bounds for Intersection Searching and Fractional Cascading in Higher Dimension.33rd ACM Symposium on Theory of Computing. ACM. PP 322-329. Doi: 10:1145/380752.380818. ISBN:978-1-58113-349-3. Retrieved 30 June 2018.

[21] Chazelle Bernard, Liu, Ding (1 March 2004). Lower Bound for Intersection Searching and Fractional Cascading in Higher Dimension. Journal of Computer and System Sciences. 68(2): 269-284. Doi:10.1016/j-jcs.2003.07.003. ISSN 0022-0000. Retrieved 30 June 2018.