Equipartition Search a New Algorithm for Searching

Arindam Agarwal
BVCOE
A-18 Pundrik Vihar,
Pitampura, New Delhi-34

Apoorv Gakhar
BVCOE
c8/10 sector 8
New Delhi-85

Narina Thakur
BVCOE
A4 Paschim vihar
New Delhi-63

ABSTRACT
Searching finds wide application in computer systems and till date, it remains one of the most fundamental operation. The need for evolving searching algorithm is never ending. This paper focuses on proposing a new algorithm namely Equipartition search algorithm and compares this method to existing methods by searching in various sequences. Results have been compiled by taking running time as a major parameter. As evident from the results, the Equipartition search method performs better than compared algorithms for several distributions. Hence the proposed method helps to reduce the running time in searching operation.

Keywords
Searching, binary search, interpolation search, linear search, jump search, binary search tree, complexity.

1. INTRODUCTION
Searching is the process of finding an element or item in a given collection or element set. Searching, till date, remains one of the most fundamental operations in any computer system or program, and it has been extensively used in numerous areas like in database management, personalized searching and, even in the currently used Operating Systems. Searching also finds its application in social Networks and search on the web [1] and information retrieval as Backwards Search Algorithm [2]. To a great sets or sequences, the cost of searching for a given key can be extremely high. Hence, there is a need for efficient searching algorithms [3]. The most widely used techniques include binary search algorithm and linear search algorithm. Both of these do not use the distribution of the elements.

It has been observed that sorting the list helps the process of searching especially for lists of large sizes [4]. Interpolation search was developed which assumes a linear distribution. These algorithms have been discussed in detail in the upcoming sections.

In this paper, a new search algorithm, namely the Equipartition search algorithm has been proposed which aims at reducing the execution time and number of probes (key comparisons) while searching for a specified key in the given collection. This algorithm treats the sequence as a linear collection of equidistant elements and uses this property to reach the destination of the specified key. The following sections discuss how searching has evolved and major characteristics of common searching methods. Section 2, 3 and 4 review and compare the existing search algorithms. Section 5 describes the newly developed algorithm and explains its working. Section 7 discusses detailed experiments and results, and Section 8 concludes this work.

2. SEARCHING ALGORITHMS
Many forms of searching algorithms are present that find the existence of an element in the given set. Some of the major searching algorithms are Linear Search, Binary Search, and Interpolation Search. These algorithms have been discussed and compared in detail. Sequential search or more commonly known as linear search is an algorithm that finds the specified value by comparing each element of the sequence till the end of the set and stops whenever a match is found [5]. This type of method does not require an ordered or sorted list. The worst case complexity comes out to be O(n), where n denotes the number of elements in the sequence, i.e., the running time is directly proportional to the number of elements in the data set. The sequential approach viable when the number of elements in the data set is considerably low. It starts to deteriorate as the number of elements increase.

2.1 Binary Search
Binary search or Half-interval search algorithm helps to determine the existence of particular value in a sequence and to locate its position in the sorted list by using a divide-and-conquer approach [6][7]. The searching process starts by comparing the key value to the middle element, which helps in discarding half of the items in every iteration. This iterative process stops when the required value is found, or no more elements are left to consider. The average case of Binary Search complexity is O(log(n)) [5]. The binary search algorithm requires a sorted sequence, but it does not take into consideration, the distribution of elements. Binary search does not take any advantage of a possible uniformity in distribution. Equation (1) below is used to find the next partitioning index in the binary search algorithm. Binary search is a dichotomic algorithm uses a three-way comparison system.

$$\text{pos} = \frac{\text{left} + \text{right}}{2} \quad (1)$$

Where pos is the next location. The value at pos and the key value is compared. Left and right represent the two extreme indices of the given set.

2.2 Interpolation Search
Unlike Binary Search, Interpolation search uses the distribution of the elements to search. Interpolation search is also a dichotomic algorithm which aims at rejecting or discarding a part of the sequence by exploiting the sorted nature of the sequence. This algorithm assumes a linear distribution and calculates a most probable index using interpolation. This assumption helps in reducing data accesses [8] while searching. W.W. Peterson first introduced interpolation search [9] in 1957 which was further studied in detail by Gonnet [10], Yao and Yao [11]. The average case complexity of this algorithm is O (log (log n)) [10] [11] [12] assuming a linear scale distribution, and the average number of accesses is equal to (log (log n)) + O(1)[13]. Interpolation search faces the limitation that as the distribution varies from the assumption of a linear scale, the algorithm starts to fail and reaches a worst-case complexity of O (n), which is same as linear search. Interpolation search tree [14] data structure has been introduced with an average cost of O (log (log n)) for

search operations and a worst case cost of O (log ~ n) to overcome the above drawback. The augmented sampled forest, or ASF [15] was introduced by Arne Andersson and Christer Mattsson to support a large class of distributions using a dynamic approach. Another drawback is that the number of comparisons in every iteration is quite high for interpolation search, limiting its use and wide acceptance. Also, the additional comparison needed to avoid a divide-by-0 situation slows the process.

The equation (2) below, is used in interpolation search to calculate the next partitioning index. This equation can be compared to the standard equation of the line as shown in equation (3). Equation (4) represents the analogous slope of assumed linear distribution in interpolation search.

$$pos = \frac{\text{left+((key-arr[left])* \{right-left\}}}{\text{arr[right]-arr[left]}} (2)$$

$$y = y1 + \frac{(x - x1) * y2 - y1}{x2 - x1} (3)$$

$$slope = \frac{\text{\{right-left\}}}{\text{arr[right] -arr[left]}} (4)$$

Santoro [16] shows how to combine both interpolation and binary search to achieve a worst case complexity of binary search i.e. O (log n). Along with interpolation and binary search, other methods like Fibonacci search and jump search also exist. Fibonacci search [17] uses the Fibonacci numbers to narrow down the key being searched. It runs in the time complexity of O (log(n)), same as binary search. Jump search [18] is a modification to sequential search where the step size varies according to the number of elements. It takes O (√n) time to complete its search which is much slower than binary search for large sets.

3. METHODOLOGY

This paper introduces an algorithm which tries to take advantage of the distribution of the sequence. This is achieved by comparing it to a series of equidistant elements between the two extreme elements. By this assumption, the sequence can be treated as divided into equally sized partitions and the algorithm can figure out the most probable partition or location for the key (element to be searched) by using the following equation(5)

$$pos = \text{left} + \frac{\text{key*(right-left+1)}}{\text{arr[right] +arr[left]}} (5)$$

Here pos is the next location, and its value is compared to the key that is to be searched. Left and right carry the same meaning as above and arr[left] and arr[right] represent the value at respective ends. Here, the number of elements is divided by the sum of first and last elements, resulting in a partitioning factor, which when multiplied by the key value provides the most probable location (pos) for the key in the given set. pos gives the most likely location under the assumption of equispaced distribution. This allows us to avoid extra comparisons.

The process starts with verifying that the element to be searched lies within the given set by comparing it to extremes. Then it checks if the extremes match the value to be searched. If yes, the respective values are returned. Following this, a most probable location is calculated by the equation mentioned above. Then, similar to binary and interpolation search, a three-way comparison is made. If the most likely location has the key element, its index is returned. If the value at calculated index is less than key value, the left half is rejected, and the calculated index (pos) becomes the left extreme. Else if the value at calculated index is greater than key value, the right half is rejected, and the calculated index (pos) becomes the right extreme. Hence the proposed algorithm is also dichotomous in nature. This process is continued until either the index of the key value is found or it's non-existence is determined. Unlike interpolation search, there is no need to verify that extremes are not equal (to avoid divide-by-0 error). Such a situation can occur when both the extremes are equal to zero, in which case, the function would return a value at an earlier stage. Not only this has fewer comparisons than interpolation search as shown, but it also works better for the slightly higher degree of distributions, cases in which the performance of interpolation search deteriorates rather quickly (discussed further in section 5).

3.1 Algorithm

1. Equipartition_search(left, right, key)
2. if(left>right||arr[left]>key||arr[right]<key)
3. return -1
4. if(arr[left]==key)
5. return left
6. if(arr[right]==key)
7. return right
8. pos=left+(key*(right-left+1))/(arr[right]+arr[left])
9. if(pos<left||pos>right)
10. return -1
11. if(arr[pos]==key)
12. return pos
13. if(arr[pos]>key)
14. return Equipartition_search(left, pos-1, key)
15. if(arr[pos]<key)
16. return Equipartition_search*(pos+1, right, key)*
17. return -1

3.2 Example

To demonstrate the working of the algorithm mentioned above, let's take a randomly generated sequence. Say,

Table 1. Randomly Generated Sequence

loc	0	1	2	3	4	5	6
Arr[loc]	67	158	210	382	499	567	681

Now on applying the algorithm to search for 499 in the above sequence.

I.e. key = 499

1st iteration

Left = 0

Right = 6

Then,

$$Pos = \text{left} + \frac{\{\text{key* left (right-left+1)}}{(\text{arr[right]} + \text{arr[left]})\}}$$

$$Pos = \ 0 \ + \frac{(499 * (6\text{-}0\text{+}1))}{(681 + 67)}$$

$Pos = 4.66 = 4$*(decimal values are neglected)*

Return Pos

Index 4 (the required location) is returned in first iteration itself.

4. EXPERIMENTAL RESULTS

4.1 Experimental Setup

The searching performance of the proposed algorithm is tested on eight synthetic sequences. Sequences [1, 2, 5, 6, 7, and 8] have been generated using simple polynomial functions while sequences [3, 4] are randomly generated sequences. Also, all of these sequences are 10^6 elements long. All the algorithms were made to search for every element present in the sequence and running times are calculated by using CPU clock cycles of the same machine. The running times have been calculated individually for a different sequence, and since the number of elements in every sequence is same, the calculated time has been used as a direct comparison for efficiency.

4.2 Results and Discussion

We can observe from the below-mentioned Table.2 that, Equipartition search algorithm outperforms Interpolation in all the considered test sequences. Since Binary search remains unaffected by the distribution of elements within the sequence, the time taken by it remains almost constant (though it is affected by redundancy), running time for both Interpolation search and Equipartition search vary noticeably for different distributions.

Table 2. Execution time in milliseconds

Sno.	Distribution or list	Equipartition Search	Interpolation Seach	Binary Search
1	y=x {1,2,3,4,5....}	**17.466**	20.649	191.308
2	y=2*x {2,4,6,8,10....}	**16.942**	19.642	186.944
3	y=random(10)* {3,17,21,32,49...}	**21.857**	25.955	187.077
4	y=random(100)* {5,170,237,323....}	**21.904**	27.39	185.996
5	y=x^0.25 {1,1,1,1,1....}	**140.57**	163.842	22.313
6	y=x^0.5 {1,1,1,2,2....}	**183.035**	205.895	62.781
7	y=x^1.85 {1,3,7,12,19....}	**392.386**	539.03	187.359
8	y=x^2 {1,4,9,16,25....}	**452.169**	728.398	186.268

*random(X) denotes a sequence of randomly generated values, where a number is generated for every X numbers.

For example, random (10) is generated in following manner in C.

For i goes from 0 to 10^6
arr[i] =rand()%10+10*i;

As evident from Table 1 (above), Equipartition search is almost ten times as fast as binary search or even more quickly for sequences[1,2,3,4]. Generalizing these results, we can say that Equipartition search shows a reduction of more than 80% in the time required to complete the search, where the sequence follows a linear and uniform scale distribution

Equipartition search takes less time than interpolation search due less number of comparisons and more converging partition approach. Also as the scale of distribution increases, the differences in the principal assumptions of interpolation (linear scale) and Equipartition search (equidistant elements) come into action, making Equipartition search much faster than Interpolation search.

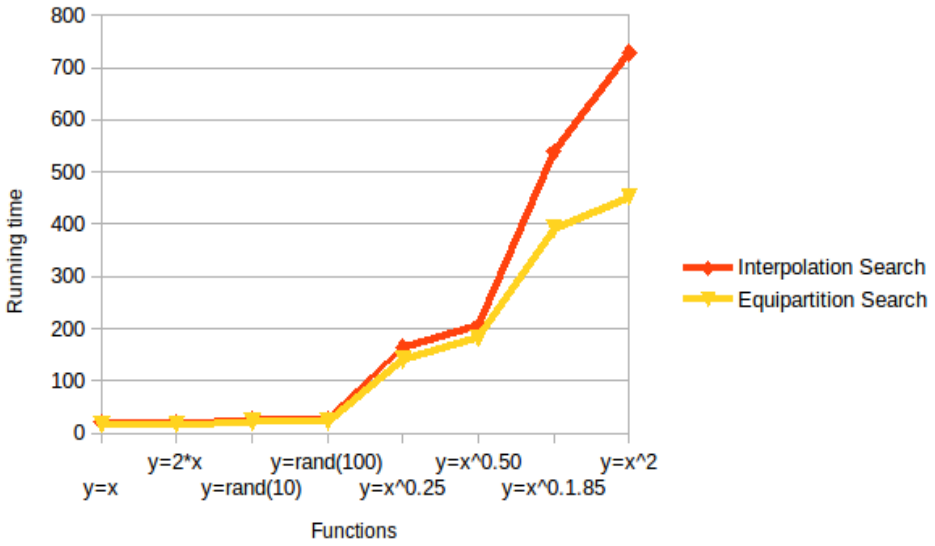

Fig 1: Comparison between Equipartition and interpolation Search

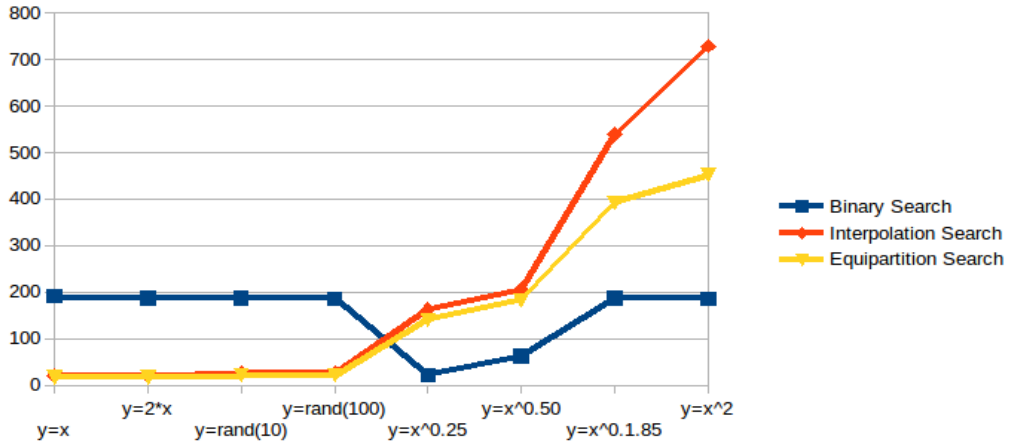

Fig 2: Comparison between Equipartition, interpolation Search and Binary Search.

Figure 2. (Above) shows how binary search remains almost unaffected by distribution while the running time is reduced as redundancy is introduced (y=x^0.25 has redundant elements). Also as the degree of distribution increases, the performance of both, Equipartition search and Interpolation search deteriorates.

5. CONCLUSION AND FUTURE SCOPE

In this paper, a new search technique, namely the equipartition search has been proposed. Its performance has been compared to existing search algorithms like Binary and Interpolation search.

Experimental Results for different data sets demonstrate that Equipartition Algorithm provides substantial speedups up to 80% time reduction over Binary Search and up to 50% reduction over Interpolation Search. Results also indicate that Equipartition search performs very fast for searching in sequences with a linear distribution or distributions similar to any arithmetic progressions but slows down as the degree of distribution increases. Nevertheless, the proposed algorithm can narrow down the vast differences between Interpolation and Binary search and can cover for flaws in interpolation search that lead to its much less application in real world.

Indeed, much remains to be done. Extending Equipartition search for distribution with higher degrees is one direction. Also, the Time complexity of the algorithm is yet to be determined. Exploring its usefulness in Artificial Intelligence, Graphic Processing, Data Handling and Machine learning tasks is another interesting avenue for future work. Tree-structured implementation of the algorithm is another promising direction for broader scope in searching objects.

6. REFERENCES

[1] Jiang Z, Li J. A tag feedback based sorting algorithm for social search. InSystems and Informatics (ICSAI), 2012 International Conference on 2012 May 19 (pp. 1482-1485). IEEE.

[2] Li Z, Zhang C, Hu Y. Backwards Search Algorithm of Double-Sorted Inter-relevant Successive Trees. In2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery 2008 Oct 18.

[3] Horowitz E, Sahni S. Fundamentals of computer algorithms. Computer Science Press; 1978.

[4] Asagba PO, Osaghae EO, Ogheneovo EE. IS BINARY SEARCH TECHNIQUE FASTER THAN LINEAR SEARCH. Scientia Africana. 2010 Dec;9(2):83-92.

[5] Knuth DE. The art of computer programming: sorting and searching. Pearson Education; 1998.

[6] Cormen TH. Introduction to algorithms. MIT press; 2009 Jul 31.

[7] Weisstein EW. Binary Search. From MathWorld—A Wolfram Web Resource.

[8] Knuth DE. TheArt ofComputerProgramming, vol. 3.

[9] Peterson WW. Addressing for random-access storage. IBM journal of Research and Development. 1957 Apr 1;1(2):130-46.

[10] Gonnet GH. Interpolation and interpolation hash searching. Canada: University of Waterloo; 1977 Feb.

[11] Yao AC, Yao FF. The complexity of searching an ordered random table. InFoundations of Computer Science, 1976., 17th Annual Symposium on 1976 Oct 25 (pp. 173-177). IEEE.

[12] Perl Y, Itai A, Avni H. Interpolation search—a log log N search. Communications of the ACM. 1978 Jul 1;21(7):550-3.

[13] Gonnet GH, Rogers LD, George JA. An algorithmic and complexity analysis of interpolation search. Acta Informatica. 1980 Jan 1;13(1):39-52.

[14] Mehlhorn K, Tsakalidis A. Dynamic interpolation search. Journal of the ACM (JACM). 1993 Jul 1;40(3):621-34.

[15] Andersson A, Mattsson C. Dynamic Interpolation Search in o (log log n) Time. InInternational Colloquium on Automata, Languages, and Programming 1993 Jul 5 (pp. 15-27). Springer Berlin Heidelberg.

[16] Santoro N, Sidney JB. Interpolation-binary search. Information processing letters. 1985 May 10;20(4):179-81.

[17] Ferguson DE. Fibonaccian searching. Communications of the ACM. 1960 Dec 1;3(12):648.

[18] Shneiderman B. Jump searching: a fast sequential search technique. Communications of the ACM. 1978 Oct 1;21(10):831-4.