Randomized Localization Search Algorithm

Bishwa Ranjan Roy[1], Purnendu Das[2], Sanju Das[3]

[1,2,3]Faculty

[1,2,3]Department of Computer Science,

[1,2,3]Assam University, Silchar, India

Abstract - This paper proposes a new searching algorithm along with the critical analysis of some efficient and popular traditional algorithm with some updated searching algorithm. The report also highlights the merits, demerits and working principles of some well-known searching algorithms along with the new proposed searching algorithm. Several searching techniques have been invented to perform efficient search on all the sorted or unsorted data set, but still more improvement in searching technique is expected. A comparison has been made among certain well-known and newly developed searching techniques with respect to their time complexity.

Index Terms – Algorithm; Searching technique; Data structure; Complexity Analysis.

I. Introduction

A searching algorithm is a technique to find a specific data from a pool of large data. It is very important operation to deal with any data structure[7]. Searching can be done linearly or non linearly over specific collection of data. Based on the structure of data, a suitable searching algorithm is selected[1,2,3]. Basic operation of any search algorithm is to compare all the data with the key value. If the key value matches with any of the data, it returns a Boolean value to show the success or failure status[4]. The efficiency of different searching algorithms depends on the pattern of the data and how the data are feed to the algorithm[5]. The performance of searching algorithms can be analyzed by considering best case, average case and worst case input. In certain algorithms, the three cases might exhibit similar performance, while in others, there could be significant variation among them. In general, average cases are considered to determine the effectiveness of any search algorithm[9,13].

Some common challenges in designing searching algorithms include:

- When the data set is huge, the number of iterations to search an element increases.
- The greater the number of iterations, the more is the time complexity.
- The simplest algorithms are comparatively less efficient and slow in searching.
- Certain calculations need to be performed for determining the pivot values.
- Cannot skip or jump elements accordingly, have to perform search serially.

II. Background

Searching algorithms include both traditional search algorithms which are commonly used in searching problems like linear search, binary search, etc. and improved search algorithms which are developed by different researcher to solve specific searching problems the search algorithms like fast string search, network localization, etc[1,8,17].
Linear Search is termed as a sequential searching algorithm where it compares the key value with all the elements staring from one end of the list to reach the other end until it find a match[6]. In this searching technique, if the the key element is not present in the list the algorithm will search all the elements available in the list. Linear search can be useful in both sorted and unsorted data. In case of *Binary Search* the algorithm can be applied on sorted data only. It bisects the whole list of element to generate two sub-list of data and continues the same procedure for the sub-list until it find a match. The key element is always searched in the middle section of the list. Interpolation Search is similar to the binary search but it operates on the sorted list where the values are uniformly distributed. It selects different data points with the range of different data sets. Unlike binary search, it operate on different locations depending upon the key value being selected [8,14,15].

III. Proposed Methodology

(1)Motivation

The motivation behind the composition of this algorithm lies in several factors. An algorithm is basically a set of rules for solving a problem in a finite number of steps. Here, the proposed algorithm can mimic to some extent the searching technique used by humans. In a sorted record of data people generally turn the first couple of pages at random and accordingly increment or decrement the pages to their required data (destination). The proposed algorithm does the same; it requires a sorted list of data and from that range of data it selects a random element and computes its value with the required data [11, 12, 18]. If it matches, it gives the element's index or else makes a local search upto a threshold value. If still not found, it searches again from a new range excluding the part which it didn't match with by incrementing or decrementing it.

(2)Proposed Searching Algorithm

The proposed searching algorithm works by taking a uniform sorted array of data as input and then making a range between the first element (0) and the last element (n-1). It generates a random number from the range and compares it with the searched element (key). If it matches it gives the index number as output. If the value generated is not matched it checks if the value is larger or smaller than the key element. If it is larger, it sequentially searches the left sublist upto threshold limit Array [mid-1] and checks if it matches with it, if not the random value that is generated becomes the last element. If the value is smaller than the key element, it sequentially searches the right sublist upto threshold limit Array [mid=1] and checks if it matches with it, if not the generated element becomes the first element and hence gives us a new set of range by deducting the indexes which do not include a key value [4, 5, 17]. From the new range, again a new random number is generated and compared with the key element. This process is repeated till the key element is found in the sorted list. The flow diagram is shown in the figure 1. The basic steps are given below:

```
Step 1 – Read the sorted N elements and the element to be searched P (key element).
Step 2 ¬ Select a pivot element randomly.
Step 3 – Search the element sequentially in the locality of the pivot element.
Step 4 – If the element is found GOTO Step 5
         else if
              the pivot value is smaller than searched element take left sub-list containing the search element
         else
              take the right sub-list containing the searched element.
         GOTO Step 2
Step 5 – STOP
```

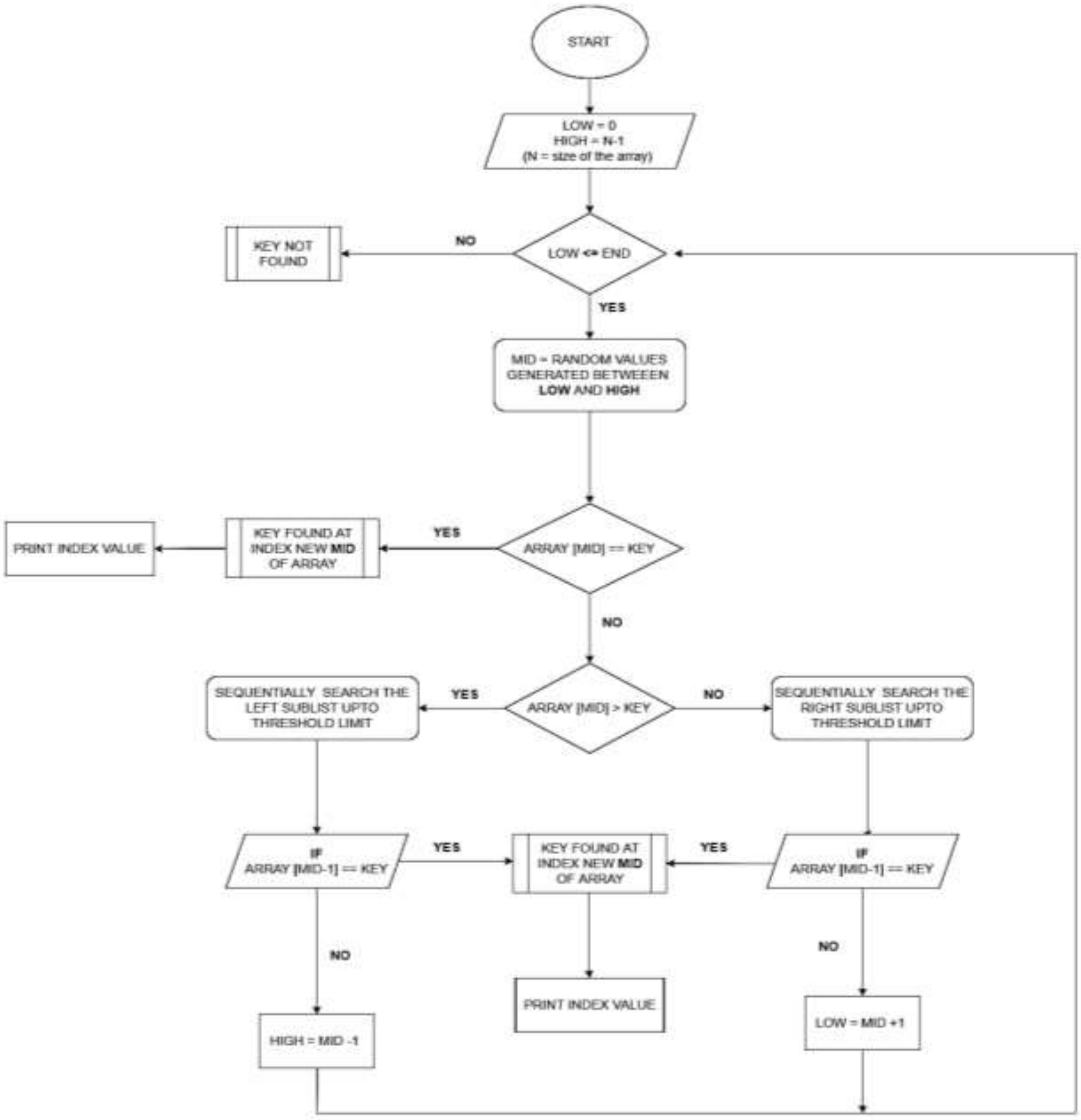
START
LOW = 0
HIGH = N-1
(N = size of the array)
NO
KEY NOT FOUND
LOW <= END
YES
MID = RANDOM VALUES GENERATED BETWEEEN LOW AND HIGH
PRINT INDEX VALUE
KEY FOUND AT INDEX NEW MID OF ARRAY
YES
ARRAY [MID] == KEY
NO
SEQUENTIALLY SEARCH THE LEFT SUBLIST UPTO THRESHOLD LIMIT
YES
ARRAY [MID] > KEY
NO
SEQUENTIALLY SEARCH THE RIGHT SUBLIST UPTO THRESHOLD LIMIT
IF ARRAY [MID-1] == KEY
YES
KEY FOUND AT INDEX NEW MID OF ARRAY
YES
IF ARRAY [MID-1] == KEY
NO
PRINT INDEX VALUE
NO
HIGH = MID -1
LOW = MID +1

Fig.1: Randomized Localization Search to find key in a sorted Array

IV. Performance Evaluation

Linear search algorithm requires *N/2* operations in average to find an element from a list of *N* elements. However, the number of operations increase linearly with the increase in the number of elements. But the constant factor 1/2 is negligible in comparison to the very large *N*. So, the cost of linear search is O(*N*). In best case, the number of operations is constant, so the cost of linear search is O(1). Whereas, in binary search we select the middle point of the sorted list of *N* elements and compare whether the searched element is less than or greater than the middle element. Similarly, in the next step we again take the middle of the half list element either left or right sublist. So, maximum of *log N* comparison is required to be performed to find an element from N elements[10,16]. Hence, the cost of binary search is O(*log N*). The best binary search algorithm also requires O(1) comparisons similar to linear search. In the proposed algorithm, the algorithm works similar to the binary search algorithm. Instead of the middle value, a random element is selected as pivot and we search the element nearby the pivot element. The steps are followed iteratively with sublist containing the searched element. The complexity of the algorithm is O(N) as the randomly generated pivot value may beat one end and the searched element at the other side. But in average case, the cost is as calculated below:-

The probability of pivot element is 1/N, so we have

$$T(n)=T(1)/N+T(2)/N+....+T(n-1)/2+1$$

The pivot element is selected in constant time. So, 1 is added at the end.

Now, multiplying both sides by N,

$$N(T(n)) =T(1)+T(2)+....+T(n-1)+N$$

Similarly,

$$(N-1)*T(N-1) = T (1)+T(2)+....+T(n-2)+(N-1)$$

So,

$$N*T(N)-(N-1)*T(N-1)=T(N-1)+1$$
$$N*T(N)=(N-1)*T(N-1)+T(N-1)+1$$
$$N*T(N)=N*T(N-1)+1$$
$$T(N)=T(N-1)+1/N$$

Now, we have

$$T(N)=1+1/2+1/3+....1/N$$

Hence, average cost of the proposed algorithm is O (*log N*).

Different important performance parameter are considered to evaluate the effectiveness of the proposed method with some popular search algorithms as in Table 1.

Table1: Detailed comparison with the popular search algorithm.

Search Technique	Linear Search	Binary Search	Interpolation Search	Randomized Localization Search
Searching principle	Sequentially search the key element in the list until it finds a match or it reaches the end of the list.	It divides the list into half and searches the key value with one of the half which may contain the element. Repeat the procedure until it find the desired element	It is an improvement on Binary search where it works on the different position of list depending on the value to be searched.	Generates a random number and compares throughout the list searching for the key element with changing range in every iteration.
Time Complexity Analysis	Best case = O (1), Average case = O (*n*), Worst case = O (*n*)	Best case = O (1), Average case = O (*log n*), Worst case = O (*log n*)	Best case = O (1), Average case = O (*log* (*log n*)), Worst case = O(*n*)	Best case = O (1), Average case = O (*log n*), Worst case = O (*n*)
Merit	It is a straightforward approach and easy to implement with limited resource. It is applicable on any types of data set.	In case of worst case scenario, it is better than linear search and interpolation search.	In terms of average case, the performance of interpolation search is better than linear and binary search.	Usually faster than linear search.
Demerit	It compares with all the elements regardless the types of data set	Not applicable to un-sorted list.	Like binary search, it is specific to sorted and also, it's execution time is equivalent to linear search in worst case scenario.	Works only on sorted elements.

V. CONCLUSIONS

The proposed algorithm works better than linear search in average case. It has the potential of being better than binary search but there exists probability of binary search functioning more efficiently. The main take away from the report is the introduction of a new searching algorithm with *log n* average behavior along with the fact that the introduction of random localization will help us to better understand the use of random generation in searching. The algorithm can be further improved by incorporating dynamic threshold values. We can include machine learning and AI to advance the searching algorithm to generate more appropriate random numbers. Thus, the upcoming opportunities in the implications of the proposed algorithm are numerous and further advancements can be done to achieve more efficient search(s) and technological developments.

VI. REFERENCES

[1] H. Chen, Design of the fast supermarket shopping system based on the Internet. J. Fujian Commercial Coll. 1, 203–208 (2014)
[2] L. Luo, C. Peng, Small warehouse management system design and implementation based on the Android system. Comput. Knowl. Technol. 16, 132–137 (2015)
[3] W. Gao, S. Li, F. Gao, et al., Design and implementation of small and medium intelligent warehouse management system based on HTML5. Electron. Technol. 97(7), 4115–4119 (2014)
[4] C. Li, H.L. Xiong, Improved binary tree search anti-collision algorithm. Appl. Mech. Mater. 2013, 397–400 (2012-2018)
[5] M. Qatawneh, International Journal of Modern Education and Computer Science, Study of Performance Evaluation of Binary Search on Merge Sorted Array Using Different Strategies, 2017.

[6] W. Sarada, Dr. P. V. Kumar, "A Comparative Study and Analysis of Searching and Sorting algorithms" , International Journal of Advanced Research in Computer Engineering & Technology (IJARCET). Volume 5, Issue 5
[7] Systematic Approach to Data Structures using C- A.M Padma Reddy
[8] A Comparative Analysis of Three Different Types of Searching Algorithms in Data Structure.(Debadrita Roy)volume 3, 5may2014.
[9] Analysis and Comparative Study of Searching Techniques (Ayush Pathak) Acropolis Institute of Technology & Research, Indore ,India.
[10] Comparing Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented Through Static Array, Dynamic Array & Linked List(Vimal P Parmar, CK Kumbhakarna) Department of computer science, Saurashtra University, Rajkot, Gujrat,India.
[11] Javed A. Aslam. A simple bound on the expected height of a randomly built binary search tree. Technical Report TR2001-387, Dartmouth College Department of Computer Science, 2001.
[12] Arne Andersson. Faster deterministic sorting and searching in linear space. In Proceedings of the 37th Annual Symposium on Foundations of Computer Science, pages 135–141, 1996.
[13] Donald E. Knuth. Sorting and Searching, volume 3 of The Art of Computer Programming. Addison-Wesley, 1973. Second edition, 1998.
[14] Conrado Mart´ınez and Salvador Roura. Randomized binary search trees. Journal of the ACM, 45(2):288–323, 1998.
[15] Kurt Mehlhorn. Sorting and Searching, volume 1 of Data Structures and Algorithms. Springer, 1984.
[16] J. Nievergelt and E. M. Reingold. Binary search trees of bounded balance. SIAM Journal on Computing, 2(1):33–43, 1973.
[17] Daniel D. Sleator and Robert E. Tarjan. Self-adjusting binary search trees. Journal of the ACM, 32(3):652–686, 1985.
[18] Raimund Seidel and C. R. Aragon. Randomized search trees. Algorithmica, 16(4–5):464– 497, 1996.