International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 09, September-2015

An Alternate Binary Search Algorithm based on Variable Checkpoint

Mohammad Zeeshan
Dept. of Computer Science & Engineering
ASET, Amity University
Lucknow, India

Alpika Tripathi
Dept. of Computer Science & Engineering
ASET, Amity University
Lucknow, India

Saba Khan
Dept. of Computer Science & Engineering
ASET, Amity University
Lucknow, India

***Abstract*—This paper describes a new and innovative concept of an alternate binary search algorithm. The proposed algorithm finds the element on the basis of the possible location of that element in the sorted list. A new formula based on VCP is generated for searching the element. The complexity of derived algorithm depends on the difference between elements in the list. Better performance of the algorithm can be seen on large data set.**

***Index Terms*—Average Difference, Binary Search, Variable Check Point.**

I. INTRODUCTION

Searching an element in the list has ever been an important task in Computer Science. There are basically few searching algorithms in the literature. Mainly, Linear search and Binary search algorithms are used widely. Linear Search is the simplest one in which we check for the element linearly in the list. It is easy to implement but its time complexity is high. In Binary Search the necessary condition is that the elements in the list should be in sorted order, and the search is based on finding the mid-point and dividing the array in two sub-arrays. Finding the mid-point is the core part of the Binary Search Algorithm, but if the desired element is not found at one stroke, it further divides the array into a sub array and again repeats the same process. The complexity of Binary Search is $O(\log_2 n)$, which increases with increase in size of the list. This paper presents a new search algorithm which will be based on positional weight of the desired element.

The basic concept behind the proposed algorithm is that, if there are n elements in the list, placed in sorted order then there would be some possibility of occurrence of an element at a particular location in that list.

II. LITERATURE SURVEY

The binary search has a great role in computer science. There are various applications of binary search. Researchers have done various changes in the technique to achieve the goal as per the requirement. In the paper by In the paper by David Dobkin et al. [2], Authors have demonstrated the usefulness and efficiency of Binary search. Binary search is extended to various multidimensional search problems. These new search techniques can efficiently solve several important problems of computer science. In the paper by Eitan Zemel [3], Author analyzes the performance of randomized binary search. Randomized binary search algorithm is used to find the global optimum of a multimodal one dimensional function. In the paper by Farhan A. baqai et al. [4], they integrate a higher order measurement-based model for printer dot interactions within the iterative direct binary search (DBS) halftoning algorithm. They have presented an efficient strategy for evaluating the change in computational cost as the search progresses. Experimental results are used to show the efficacy of the approach. In the paper by Ahmed Tarek [1], auther said that binary search algorithm is fundamental method to the study and analysis of Discrete Computational Structures. This is an efficient searching technique due to its logarithmic time complexity. It is used to identify the position of a key in a sorted list. Often, database applications need searching for two to more different key elements at the same iteration. Author has changed the binary search algorithm for searching the multiple items in one stroke. In the paper, X. L. Shi et al. [5], authors investigated that in RFID system, tag collision is a main problem for fast tag identification. On the basis of binary search of backtracking, an enhanced binary anti-collision search algorithm for radio frequency identification (RFID) system is presented.

III. PROPOSED WORK

The concepts and processes used in the proposed work are discussed below.

A. Methodology

In this paper, the proposed algorithm uses the list division technique of traditional Binary Search algorithm. But the main difference between both of them is that the proposed algorithm finds a Variable Check Point instead of the midpoint as in the traditional Binary Search. One more change is done in the Binary Search Algorithm that the new proposed algorithm checks both the upper and the lower element of the array.

The concept behind the overall algorithm is that if the array elements are sorted, then there would be some possibility of a particular element to be at a specific location. This specific location can be calculated by finding the Average Difference (AD) between the array elements. Average Difference (AD) plays vital role in calculating the location of the desired element. The difference pattern creates

the Best, Average and Worst case for the proposed algorithm. The details will be discussed in the other section.

The Variable Check Point (VCP) is calculated with the help of Average Difference (AD). To find the Average difference (AD), firstly the Range (R) is calculated. Range (R) is the difference between the upper most and the lower most element of the array.

$$Range\ (R) = Upper\ element - Lower\ element \quad (1)$$

Range is divided by total number of gaps between the array elements that is one less than total number of array elements (N).

$$Number\ of\ Gaps = N - 1 \quad (2)$$

Now the Range is divided by the number of gaps that is the Average difference.

$$Average\ Difference\ (AD) = Range / N\text{-}1 \quad (3)$$

After calculating the Average Difference (AD), the Variable Check Point (VCP) is calculated. To do this the Lower element is subtracted from the desired element that we are searching for. And after that the resultant of this subtraction is divided by the Average Difference. Now it is the Variable Check Point (VCP).

$$VCP = (Desired\ element - Lower\ element) / AD \quad (4)$$

The lower and upper element checking increases the possibility of finding the key element at one stroke.

B. Proposed Algorithm

The above mention equation (1) for calculating the Variable Check Point (VCP) is used in the new proposed algorithm.
Following notations are used in the algorithm:
key: The number searched for.
arr: Name of the array of list elements.
low: Lower-most index in the list.
up: Upper-most index in the list.
tag: Used as flag.
chk: index of Variable Check Point
gap: No. of Gaps between the list elements.
The proposed algorithm is as follows:

```
Algorithm alternate_binary_search
while key ≥ arr[low] and arr[up] ≥ key and low ≤ up do
    if arr[low] = key then
        print low+1
        tag ← 1
        break
    else if arr[up] = key then
        print up+1
        tag ← 1
        break
    else
        chk ← (key-arr[low])*gap/(arr[up]-arr[low]
        chk ← chk+low
        if arr[chk] = key then
            tag ← 1
            print chk+1
            break
        else if key < arr[chk] then
            up ← chk-1
            low ← low+1
        else
            low ← chk+1
            up ← up-1
    end if
    gap ← up-low
end while
if tag ≠ 1 then
print not found
end if
```

The first line of the algorithm is for the while loop which checks three conditions; first one is that the *key* is greater than or equal to the lower most element (*arr[low]*) of the array, second one is that the upper most element (*arr[up]*)is greater than or equal to the *key* and the third one is that the upper index (u*p)* is greater than the lower index (*low)* of the array. The first two conditions stand to check whether the *key* element is inside the list or not and the third one checks that the *low* and *up* have crossed each other or not. If all the three conditions are true, the control will enter inside the loop. The second line of the algorithm checks if the lower most element (*arr[low]*)is the *key* element then it will show the location of the lowest element and breaks the loop. If the condition is not true then it will skip the above lines and jump to the 6th line and checks whether the upper most element (*arr[up]*) is equal to the *key* element.

If the *key* could not be found at lower most and the upper most index location of the array list then the control comes to the 11th line of the array where the *Variable Check Point (chk)* is calculated. Line 12 sets the *chk* according to the lower index (*low)* of the array. In line 13, control checks if the *arr[chk]* is equal to the *key.*If not, line 17 checks whether the *key* is less than the *arr[chk]*. If it is true then the upper most index is shifted to one less than the *chk*, and the lower is also shifted upward with one location.

If the condition in line 17 does not hold, then in line 21 and 22, the lower is shifted to *chk+1* and *up* is shifted to one less than itself. This shifting is done to set the lower and upper of the sub array. The loop continues until any of the condition is false. In line 4, 8 and 14; the *tag* sets itself 1 if the *key* is found. Line 26 checks the value of *tag*. If *tag* is not equal to 1 then not found is shown.

IV. Experimental Result

On a specified number of elements it is found that the proposed algorithm, most of the times, is better than the binary search algorithm. An array of size 100 has been taken to analyze the output and the performance of the proposed algorithm. We have taken 100 number starting from 0 to 4950. When the input sequence with equal difference is given to the array then the algorithms finds its each element at one stroke.
at the location 50 if we give a sudden hike then the algorithm starts giving poor results. When we give the difference between the elements according to the real time data then on an average the algorithm takes 2.35 iterations to find an element, where binary algorithm takes 5.8 iterations to find an element.
Similar results have been found for the 1000 elements.

V. Performance Analysis

As the proposed algorithm is based on the Variable Check Point, the location of the check point *chk* could be anywhere in the array list. The algorithm finds the most possible location of the *key* on the basis of the *Average difference* between the list elements. It has been found that if the differences between the list elements are more random or there is a sudden high difference between two numbers, it disturbs the Average difference at all and makes the worst case for the algorithm.

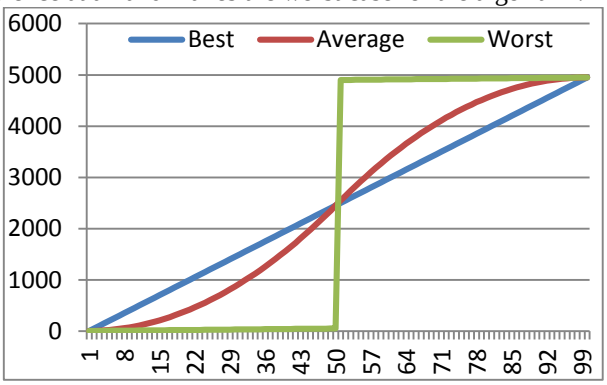

Fig. 1: Best, Average and Worst cases based on the differences between the list elements

The proposed algorithm is different from the traditional binary search algorithm, because the complexity of traditional binary search depends on the probability that the key element is at the midpoint or anywhere else.

The proposed algorithm can perform better in the worst case of the traditional binary search if the differences between the elements are equal or in a defined and limited range.

The performance of the proposed algorithm changes with the change in differences between the elements. Differences, that are approximately close and distributed throughout the array makes the performance better.

The proposed algorithm performs its best when the differences are equal between the array elements, but the condition is impractical. The pathetic condition occurs for the algorithm in its worst case, when the elements are increasing with the difference 1,

VI. Comparative Analysis

In Comparative analysis of the proposed algorithm with the traditional binary search algorithm, it can be seen that the proposed algorithm works better than the traditional binary search in its Best and Average case. Graph for each Best, Average and Worst cases are given below in comparison with binary search for the 100 elements for a specific range of values.

A. *Best case analysis*

Best case for the proposed algorithm does not depend on the position of the key element in the list, but on the differences between the elements. Proposed algorithm can search each element in one stroke if the differences between elements are equal.

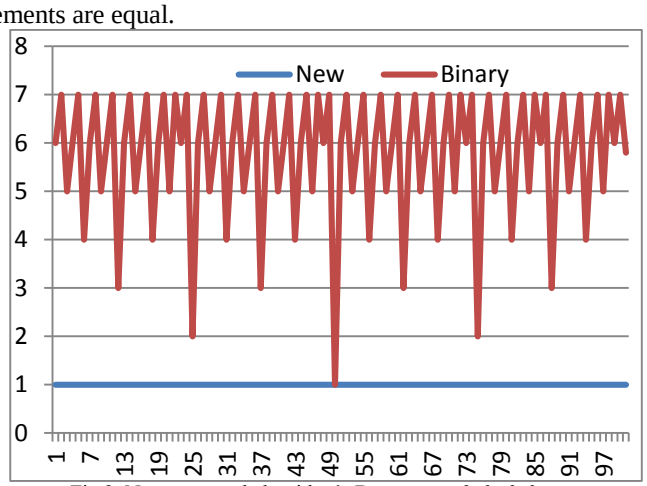

Fig 2: New proposed algorithm's Best case and the behavior of Binary search

In the fig 2; X-axis shows the no. of elements in the list and the Y-axis contains the no. of iterations to find an element at a specific location. The behaviour of binary search remains same for the same no. of elements, i.e. for the fixed no. of elements it will always firstly search for the midpoint and then move to left or right sub-list accordingly, but the proposed algorithm behaves according to the difference between its elements i.e. if the differences between the elements are equal then it will strike on the exact location of key element.

It has been found that the average number of iterations required to search an element in the array using the proposed algorithm is 1, whereas for the binary search algorithm it is more than 1 and increases logarithmically.

B. *Average case analysis*

For a specified number of elements, Binary search algorithm works same, no matter where a specified element is located and how much the difference between the elements is.

On the contrary, in the proposed algorithm, the differences between elements plays vital role.

The average case for the proposed algorithm occurs when the difference between the elements are not equal, but in a specified range and distributed throughout the array.

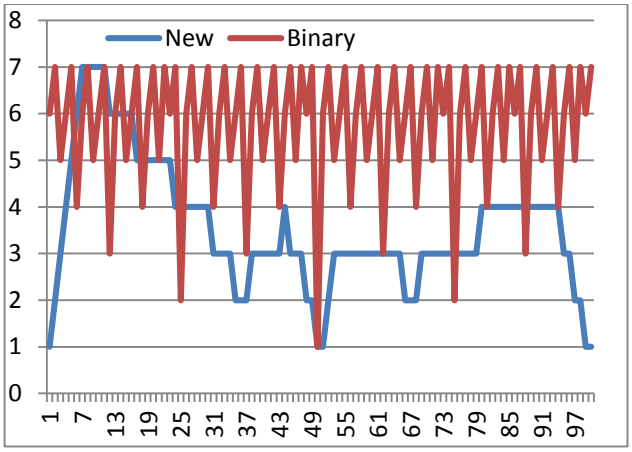

Fig 3: Average case of new proposed algorithm and the Binary

In fig 3, the graph shows the comparison between Binary search algorithm and the proposed algorithm. The results tested on 100 specified value elements shows that the proposed algorithm takes equal iteration to the binary search element at few locations, and it can also take more iteration in some cases but mostly it is taking less iteration than the Binary search algorithm at most of the locations. If we compute the average of total no. of iterations, then proposed algorithm takes less iterations than the Binary search algorithm.

On an average the proposed algorithm works better than the Binary search algorithm in its average case.

C. Worst case analysis

The proposed algorithm performs badly in its worst case. Worst case for the proposed algorithm occurs when the numbers are increasing with the difference 1 from the starting but at the middle location of the array there is an abrupt extreme high increase in the difference, and after that sudden difference the elements again start increasing with the difference 1.

This condition becomes worst case for the proposed algorithm.

Fig. 4 shows the graph for the performance of proposed and binary search algorithm in the worst case.

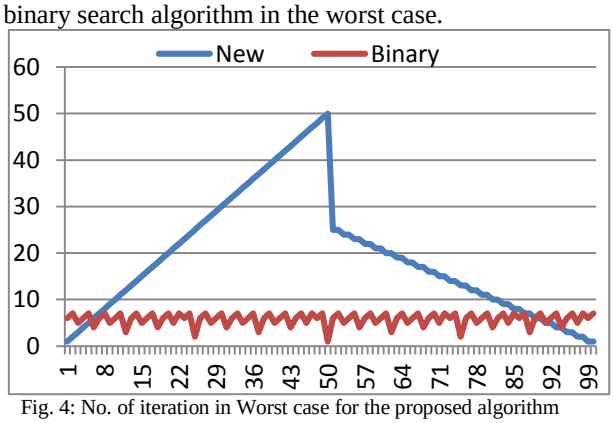

Fig. 4: No. of iteration in Worst case for the proposed algorithm

In this condition the algorithm starts changing the check point linearly up to n/2 in half of the array. Just after the middle location the no. of iterations reduces to n/4 and starts decreasing by 1 after each two elements. This wretched behaviour makes the algorithm useless for the searching problems. But the situation, in which the algorithm works badly, is impractical in use.

VII. Conclusion and Future Scope

The proposed algorithm works better in case that the differences between the array elements are not abruptly high. So the proposed algorithm can be used where the database values are in a approximately equal or in a limited range. The real time data is normally matches this requirement of the proposed algorithm.In future the work on the proposed algorithm can be furnished more to match to the requirement of the technology.

Acknowledgment

We would like to thank respected Mr. Aseem Chauhan, Chancellor, Amity University, Lucknow and Maj. Gen. K.K. Ohri AVSM (Retd.), Pro.Vice-Chancellor, Amity University, Lucknow for providing excellent facilities in university campus and their encouragement and advice. We would also like to pay regards to Prof. S.T.H. Abidi, Director and Brig. U. K. Chopra, Deputy Director, Amity University, Lucknow for their valuable feedback.

References

[1] Ahmed Tarek-A New Approach for Multiple Element Binary Search in Database, INTERNATIONAL JOURNAL OF COMPUTERS,Issue 4, Volume 1, 2007

[2] David Dobkin and R.J. Lipton, On some generalization of binary serach, Sixth Annual ACM Symposium on theory of computing, Washington, April 1974.

[3] Eitan Zemel-Random Binary Search-A Randomizing algorithm for global optimization in R1*, Bonn Workshop on Combinatorial optimization, June 1984.

[4] Farhan A. Baqai and Jan P. Allebach-PRINTER MODELS AND THE DIRECT BINARY SEARCH ALGORITHM, 0-7803-4428-6198, 1998 IEEE.

[5] X. L. Shi, F. Wei, Q. L. Huang, L. Wang, and X. W. Shi-Novel Binary Search, Progress In Electromagnetics Research B, Vol. 9, 97–104, 2008.