A New Modified Linear Search Algorithm

Subir Saha[#1], Mridul K. Bhaumik[*2], Supratim Das[#3]

[#] *Department of Computer Science, Belda College, Paschim Medinipur-721424, India.*
**Department of Computer Science, K.K. Das College, Kolkata-700084, India.*
[#]*Department of Mathematics and Humanities, CET, Bhubaneswar-751003, India.*

Abstract *- A new modified linear search technique is proposed for the search of an element from an unsorted array elements to refine search techniques of unsorted array elements such as linear search technique and two way linear search technique. Algorithm and efficiency analysis of the proposed search technique is given. A comparison with linear search technique and two-way linear search technique is also given.*

Keywords - *Searching, Algorithm, Linear search, Modified Linear Search, Binary search, Empirical, Efficiency, Complexity.*

I. Introduction

Different search techniques such as linear search, binary search, ternary search etc. [1, 2, 3, 4, 5, 6] play a major role in various scientific and engineering fields. Searching techniques has a considerable attention in Data Structure through last few decades. Now a days, design an algorithm to refine searching techniques is an important task since searching an element in a significant lesser time from a lot of data stored in a system has immense importance in Computer Application. A linear search is the most basic and simple search algorithm. Linear search also known as sequential search which is made over all items one by one until every item is checked and if a match found then that particular item is returned otherwise search continues till the end of the data collection. The worst case performance scenario for a linear search is that it needs to loop through the entire collection, either because the item is the last one or because the item is not found. In a collection of N items, the worst case scenario to find an item is $\Theta(N)$, where Θ is the big theta function [7]. Despite its less performance, linear search is extremely common. Binary search is a more specialized algorithm than a sequential search. In this search data must be in sorted form. The basic idea of the binary search is to divide the sorted data into two halves and to examine the data at the point of split. In worst case, the binary search produces a high time complexity $\Theta(\log_2 N)$ than linear search in an N- items data. Binary search fails if data are not in sorted form. Searching an element from an unsorted collection of data has a
wide range of application than from a sorted collection of data. So the searching techniques which searches element from an unsorted collection of data has to be modified.

In the worst case, when there is no match element or the first match element happens to be at the last iteration of searching, the algorithm makes the largest number of iterations among all possible inputs of size N. The worst case efficiency of an algorithm is its efficiency for the worst case input (or inputs) of size N for which the algorithm runs the longest among all possible inputs of the size. The best case analysis of an algorithm is its efficiency for the best case input of size N, which is an input (or inputs) of size N for which the algorithm runs the fastest among all possible inputs of that size. Accordingly, one can analyze the best case efficiency as follows. First, one can determine the kind of inputs for which the count $C(N)$ will be the smallest among all possible inputs of size N. Then the value of $C(N)$ should be ascertained on these most convenient inputs. For a sequential search from an unsorted list of $N-$ items ($A[0, 1, 2, \ldots N-1]$),

$$C_{worst}(N) = N \text{ and } C_{best}(N) = 1. \qquad (1)$$

For the assumptions, (*i*) the probability of a successful search is equal to p $(0 \le p \le 1)$ and
(*ii*) the probability of the first match occurring in the i^{th} position of the list is the same for every i. The average number of key comparisons is [7]

$$C_{avg} = \frac{N+1}{2}p + (1-p)N \qquad (2)$$

Arora et al [4] got a refinement of linear search and presented two way linear search technique. In this paper, our aim is to refine linear search technique since it has a wide range of application as it works even if data are not in sorted form, to do so, we introduce a modified linear search (MLS) technique to reduce the time taken to search a

key item 'K' from an unsorted list of N- items ($A[0, 1, 2, ...N-1]$). We organize the rest of the paper in the following manner. In section-2, we apply MLS with one middle element. MLS with one middle element has no significant difference with the two way linear search [4]. We will put an algorithm and its efficiency for this search technique. We introduce this technique to generalize the idea of entering more middle elements to get a significant refinement of two way linear search technique. In section-3, we apply MLS with two middle elements. We write an algorithm and analyze its efficiency for this refined search technique. In section-4, some concluding remarks will be given.

II. MLS with one middle element

We apply MLS technique with one middle element of the array. It works by comparing the search item K by one increasing index from first element to the middle element and the another increasing index from the middle to the last element simultaneously. This check will continue until either a match with the search key K found or the list $A[0, 1, 2, ..., N-1]$ is exhausted. First we put an algorithm for this search and then we analyze the efficiency of this search algorithm. **Algorithm**

MLS ($A[0, 1, 2, ..., N-1]$, K)

//Implements Modified linear search for a given value in a given Array

//Input: An unsorted Array $A[0, 1, 2, ..., N-1]$ and a search key K

//Output: Return the index of the Array's element that is equal to K or 1 if there is no such element

lb ← 0
ub ← $N - 1$
mid ← (lb + ub)/2
$i \leftarrow 0$
$j \leftarrow$ mid
while $i <$ mid and $j < N$
    if $K = A[i]$ or $A[j]$
        return 1
        $i = i + 1$
        $j = j + 1$
    else
    return − 1

Efficiency Analysis: The run time of this algorithm can be quite lesser than linear search algorithm for the same number of elements of size N. One can analyze the algorithm to see what kind of inputs yields the largest value of the basic operation's count $C(N)$ among all possible inputs of size N, then worst case value will be

$$C_{worst}(N) = \frac{N}{2} . \quad (3)$$

For this search technique, best case inputs will be the lists of the data of size N with the first element or their middle element equal to a search key;

$$C_{best}(N) = 1. \quad (4)$$

But, neither the worst case analysis nor its best case counterpart yields the necessary infor- mation about an

algorithm's behavior on a 'typical' or 'random' input. So, the average case efficiency i.e., $C_{\text{Avg}}(N)$ has to be calculated.

Assume that

i. The probability of a successful search is equal to p ($0 \leq p \leq 1$).

ii. The probability of the first match occurring in the i^{th} or j^{th} position of the list is the same for every i or j.

In case of a successful search the probability of the first match occurring in the i^{th} or j^{th} position is $2p/N$ for every i or j and the number of comparisons made by the algorithm in such a situation is i. In case of an unsuccessful search, the number of comparison is $N/2$ with the probability of such a search is $(1-p)$. Therefore,

$$C_{avg}(N) = \left(1 + 2 + \cdots + \frac{N}{2}\right)\frac{2p}{N} + \frac{N}{2}(1-p)$$

$$= \frac{N+2}{4}p + \frac{N}{2}(1-p) \qquad (5)$$

The result (5) yields some noticeable information. For p = 1 (i.e. for a successful search), the average case efficiency (5) reduces to (N+2)/4 i.e. the algorithm will inspect, on average, about 1/4 th of the list's elements. If p = 0 (for unsuccessful search), the average number of iteration (5) can be put as N/2 which is same as $C_{\text{worst}}(N)$. For N = 2 i.e. for the first element and the middle element $A[M]$ and p = 1, (5) becomes 1 which is equal to $C_{\text{best}}(N)$. The time complexity of this search is $\Theta(N)$ which is similar as linear search [7] and two way linear search [4]. Arora et al [4] compared Linear search algorithm with two way linear search algorithm using Matlab 8.0 for implementation and Analysis of CPU time taken by both the algorithms and they observed a significant refinement. We compare this MLS technique with one middle element with linear search technique and also with two way linear search technique.

The time complexity of both the MLS algorithm with one middle element and two way linear search algorithm is $\Theta(N)$. Also there is no significant refinement of (5) of efficiency two way linear search algorithm. Though we are not getting refinement, MLS with one middle element is proposed and analyzed to generalize this technique and get a refined technique which we propose in the next section.

III. MLS with two middle elements

In this section, we apply MLS technique with two middle elements by dividing the array list in three equal halves and the search key K compare with list elements by increasing index from first element to the first middle element $A[M1]$, second one by increasing index from first middle element $A[M1]$ to the second middle element $A[M2]$ and the last one by increasing index from second middle element $A[M2]$ to the last element of the list $A[0, 1, 2, ..., N-1]$ and this check will continue until a match with the search key K found from $A[0, 1, 2, ..., N-1]$ or the list become exhausted. We write an algorithm for this search technique and then we put an efficiency analysis of the algorithm.

Algorithm MLS ($A[0, 1, 2, ..., N\ 1], K$)

//Implements Modified linear search for searching a given value in a given Array

//Input: An unsorted Array $A[0, 1, 2, ..., N-1]$ and a search key K

//Output: Return the index of the Array's element that is equal to K or 1 if there is no such element

lb ← 0;
ub ← $N - 1$;
mid1 ← (lb + ub)/3; mid2 ← 2
× mid1;
i ← 0

j ← mid1

$l \leftarrow$ mid2

while $i <$ mid1 and $j <$ mid2 and $l < N$

if $K = A[i]$ or $A[j]$ or $A[l]$ return 1

$i = i + 1$ $j = j + 1$
$l = l + 1$

else
return -1

Efficiency Analysis The run time of this algorithm will be a refinement than both the two way linear search algorithm and MLS algorithm with one middle element for the same list of size N . The worst case value of this algorithm will be

$$C_{\text{worst}}(N) = N/3 \tag{6}$$

which gives us a refinement than the worst case value of both the algorithms of MLS with one middle element and two way linear search.

For this algorithm the best case analysis is

$$C_{\text{best}}(N) = 1. \tag{7}$$

Now we discuss about average case efficiency of the algorithm i.e. of $C_{avg}(N)$. To do so, we assume that

i. The probability of a successful search is equal to p $(0 \leq p \leq 1)$.

ii. The probability of the first match occurring in the i^{th}, j^{th} or k^{th} position of the list is the same for every i, j or k.

The probability of the first match occurring in the i^{th} or j^{th} or k^{th} position is $3p/N$ for every i, j or k for the successful search and the number of comparisons made by the algorithm in such a situation is i. In case of an unsuccessful search, the number of comparison is $N/3$ with the probability $1 - p$. The average case efficiency for this algorithm can be put as [7]

$$C_{avg}(N) = \left(1 + 2 + \cdots + \frac{N}{3}\right)\frac{3p}{N} + \frac{N}{3}(1-p)$$
$$= \frac{N+3}{6}p + \frac{N}{3}(1-p) \tag{8}$$

Analyzing the formula (8), the following observations can be made: For a successful search, $p = 1$ and the average case efficiency (8) becomes $\frac{N+3}{6}$ i.e. the algorithmic inspection, on average, about $1/6$ th of the list's elements. In case of an unsuccessful search, $p = 0$ and the average case efficiency (8) is given by $\frac{N}{3}$ which is equal to $C_{\text{worst}}(N)$. For $N = 3$ i.e. for the first element and the middle elements $A[M1]$ & $A[M2]$ and $p = 1$, (8) becomes 1 which is equal to $C_{\text{best}}(N)$.

IV. Concluding Remarks

In this work we proposed new MLS techniques with one middle element and two middle elements to search an element from an unsorted array elements of size N. We compare these proposed techniques with linear search technique and two way linear search technique and observed that though MLS technique with one middle element is not a refinement of two way linear search technique, it gives the idea of MLS technique with two middle elements. We have shown that the MLS technique with two middle elements is a refinement of all the previous search techniques of an unsorted array elements such as linear search technique, two way linear search technique and MLS technique with one middle element. One may generalize the idea and can think of MLS with three, four or more middle elements and gets more and more refined techniques with increasing middle elements. The average case efficiency of MLS algorithm with m middle elements will be $C_{avg}(N) = \frac{N+m+1}{2m}p + \frac{N}{m+1}(1-p)$ on an unsorted array of N elements.

V. References

[1] D. Knuth 'The Art of Computer Programming Sorting and Searching', Addison- Wesley 3 (1998)
[2] A. Oommen and C. Pal Int. Jour. of Innov. Res. in Tech. 1 800-803 (2014)
[3] S. Korteweg and H. Nieuwenhuÿzen The jour. of Forth Appl. and Res. 2 43-45 (1984)
[4] N. Arora, G. Bhasin and N. Sharma Int. Jour. of Comp. Appl. 107 (2014)
[5] S. Manchanda 'Ternary search algorithm: Improvement of Binary search' Research- gate (2015)
[6] A. R. Chandha, R. Misal and T. Mokashi Int. Jour. of Appl. Inf. Syst. 7 37-40 (2014)
[7] A. Levitin 'Introduction to The Design and Analysis of Algorithms' Pearson Educa- tion (2004)