Tri-Search: A New and Efficient Searching Algorithm: An Extension of Ternary Search Approach with Variable Partitioning

Hriday Kumar Gupta
Shobhit University, Meerut
NH-58, Modipuram, Meerut, Uttar Pradesh 250110

Rajesh Pandey
Shobhit University, Meerut
NH-58, Modipuram,
Meerut, Uttar Pradesh 250110

ABSTRACT
Searching is a traversal technique in a data structure to search a particular element in a given set of particular domain. Sorting Technique is generally used in a huge variety of important applications to search a particular item. There are various Searching Algorithms for different data structure having different time and space complexity. This paper contributes an efficient searching algorithm Tri-Search search which is poisoned on dividing the given elements into three unequal parts. This paper also compare the Tri-Search search algorithm with Linear Search and Binary Search. Python is used for implementation and Analysis of CPU time taken for all the three searching algorithms used. Linear search can be used with any random array elements but for binary search and Tri-Search search element must be in sorted array. Result shows that Tri-Search search algorithm requires less time for search any particular element.

General Terms
Searching, Data Structure, Binary Search, Tri-Search, types of searching, Classification of searching.

Keywords
Binary Search, Complexity, algorithms, data, key. procedure.

1. INTRODUCTION
In computer science, searching is the process of finding an item with specified properties among a collection of items. The items may be stored as records in a database, simple data elements in arrays, text in files, nodes in trees, vertices and edges in graphs or may be elements of other search space. Several Searching/Traversal Algorithms with different time and space complexity for different data structure are exist and used. This paper discussed and compare the previously exist searching algorithms.

This paper contributes a novel searching algorithm Tri-Search which is based on dividing the given elements into three unequal parts. In this paper the comparison of the Tri-Search search algorithm with Linear Search and Binary Search is also has been reviewed.

2. RELATED WORK
There are assertive ways of organizing the data which enhance searching process. That means, if the data is organized in some proper structure then it is accessible to search the required item. Sorting is one of the approaches for making the elements ordered.

3. SEQUENTIAL SEARCH
3.1 Unsorted Sequential Search
Suppose given an array in which order of element is unknown. That means the elements of the array are not sorted. In this case if searching for an element then array list has to be scan completely and t will be cleared if the element is there in the given array or not.

```
//Serch procedure for a matching data in the array
int UnsorteddSequentialSearch (int ARR [], int N, int KEY)
{
//loop overall items in the array
for (int i = 0; i < N; i++)
{if (ARR [i] == KEY)
//if data found return index
return i;}
//Scanning done but data not found then return -1
return -1;
}
```

3.2 Sorted Sequential Search
If the array elements are already sorted then in many cases the there is no need to scan the complete array to see if the element is there in the given array or not. In the below algorithm, it can be seen that, at any point if the value at Arr[i] is greater than the to be searched data then just return −1 without searching the remaining array elements.

```
//Serch procedure for a matching data in the array
int SortedSequentialSearch(int ARR [], int N, int KEY){
//loop overall items in the array
for (int i = 0; i < N; i++){
if (ARR [i] == KEY)
//if data found return index
return i;
else if(ARR [i] > KEY)
//if data value exceeds ,return -1
return -1;}
//Scanning done but data not found then return -1
return -1;}
```

3.3 Complexity Analysis of Sequential Search

For unsorted and sorted sequential search Time complexity O(n) .This is because in the worst case we need to scan the complete array, but the average case of sorted sequential search reduces the complexity even though the growth rate is same.

4. BINARY SEARCH

Suppose the aim is to search a word in a dictionary, generally approach is to go directly on some page and start searching from there. If the word that are searching is same then suppose to stop the search. If the page is before the selected pages then we generally apply the same process recur for the first half otherwise apply the same process recur for the second half. Binary search also works in the same way. The algorithm implement such a approach is introduced as binary search.

4.1 Procedure

In this procedure basically ignore half of the elements just after one comparison.

Compare x with the middle element.

If x matches with middle element, return the mid index.

Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So recur for right half.Else(x is smaller)recur for the left half.

```
// A recursive binary search function, It returns location of x in given array    arr[0..n-1] is present, otherwise -1

int BinarySearch(int ARR[],int START,int END,int KEY){

if(START== END){

if(ARR [START]== KEY)

return START;

else

return -1;}

else{

// The below original middle point condition is done to avoid overflow that may occure in adding two very big integers range where the addition result may become greater than INT_MAX limit and yield unexpected results.

int MIDDLE= START +( END - START)/2;

// If the element is present at the middle itself

if(ARR [MIDDLE]== KEY)

return middle;

//If element is smaller than middle, then

if(ARR [MIDDLE]> KEY)

BinarySearch(ARR, START, MIDDLE -1, KEY);

else

BinarySearch(ARR, MIDDLE +1, END, KEY);

}}
```

4.2 Complexity Analysis of Binary Search

After every passes , the array list is divided into 2 equal parts. Therefore N items can be divided into two parts almost $\log_2 n$ times. So execution time of binary search is O($\log_2 N$).

After 1st iteration, N/2 items remain ($N/2^1$)

After 2nd iteration, N/4 items remain ($N/2^2$)

Worst case: Number of iterations is $\log_2(N)$

It is said that Binary Search is a logarithmic algorithm and executes in O($\log_2 N$) time.

5. PROPOSED ALGORITHMS TRI-SEARCH

In the proposed algorithm element are stored in array in either increasing or non-increasing order. As in binary search we divide the array in two equal part(half-half) after finding middle point index but here we are dividing array list in three unequal parts after calculating two locations(P1 and P2) . The first part of the first pass contains 25% elements, second part contains 50% element and third part contains 25% element. The element to be search can be lying in any of these three parts. Similarly in second pass array list is divided into unequal part as 35% ,30% and 35% respectively and in third pass array list divided into 45%,10% and 45% respectively. After performing these passes partially finished one iteration of the proposed algorithms. If till this iteration if searched element not found then suppose to recursively perform this procedure

5.1 Implementation of Proposed Algorithm

//Tri-Search Search PROGRAM implemented recursively.

Output: hold the index of searched item if found and -1 value if not found

//A is array passed by main function also TS received START and END index value of array list along with value to be searched and I is a global variable having initial value -10 .

```
int TS(int ARR [],int START ,int END,int KEY)

{

int P1,P2;//contain the index of first part and second part of array list

IF(START<=END){

I=I+10;//I is used for updating the index

IF(I==30) I=0;

P1=START+(25+I)*(END-START)/100;

P2=START+END-P1;

IF(ARR [P1]==KEY) return P1;

ELSE IF(ARR [P2]==KEY) return P2;

ELSE IF(KEY> ARR [P1] &&KEY< ARR [P2])

{ START=P1+1;

END=P2-1;

return TS(ARR,START,END,KEY);}

ELSE IF(ARR [P1]>KEY)

{END=P1-1;return TS(ARR,START,END,KEY);}

ELSE IF(ARR [P2]<KEY)
{ START=P2+1; return TS(ARR,START,END,KEY);}
}//END IF
return -1;
}//end of function
```

If searched element is not found then this function will return -1 otherwise return the index of the key if searched element is found at P1 or P2.The proposed algorithms will check flowing three conditions after finding P1 and P2 when the key is not found at the P1 and P2.

Condition 1: IF SEARCHKEY is less than P2 and also greater than P1.Then (P1 + 1) will become the FIRST and (P2 - 1) will become the LAST and the procedure will be repeated for sub list.

Condition 2 : IF SEARCHKEY is greater than P2. Then (P2+1) will become the FIRST and then the procedure will be repeated for sub list.

Condition 3: IF SEARCHKEY is less than P1. Then (P1 -1) will become the LAST procedure will be repeated for sub list.

6. EMPERICAL RESULTS

6.1 Improvement of Tri-Search over Binary Search and Ternary search: Time Complexity

After every passes, the array list is divided into three unequal parts. In worst case after completion of all three iteration of kth pass the total time taken is

$N(1/2)^k(7/20)^k(9/20)^k$

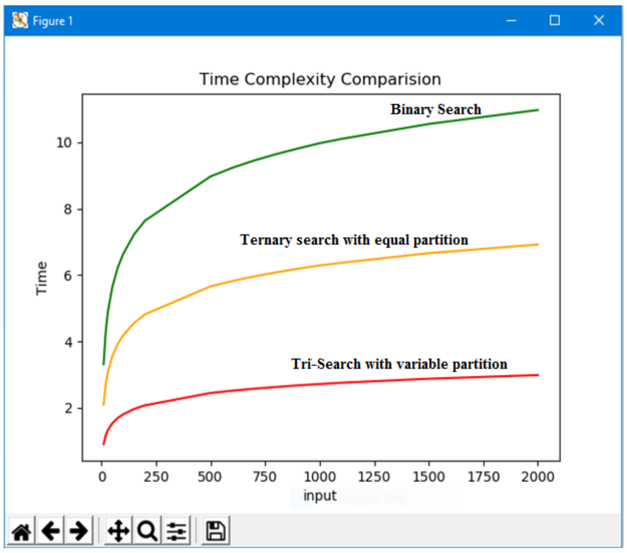
To terminate the recursive tree for this recursive procedure we have relation

$N(1/2)^k(7/20)^k(9/20)^k = 1$

In worst case and to terminate recursive tree we have condition as

$N(63/800)^k=1$

After computation the value of k is $\log_B(n)$ where B=12.6



7. CONCLUSION AND FUTURE SCOPE

This paper, present a new efficient approach for searching data and had been achieved to search within the sorted linear order of items with worst-case complexity as $O(\log_{12.69}(n))$.The approach that had been used to implement this algorithm involves variable partitioning of array list . The performance graph for binary, ternary and tri-search is also present along with their comparison graph. Thus it can observed that how efficient is to use this algorithm it has minimum worst-case complexity. Ternary search are efficient for problem like "Given a word, find the next word in dictionary" or "Find all telephone numbers starting with 9758 or "typing few starting characters in a web browser displays all website names with this prefix". Used in spell checks.

8. ACKNOWLEDGMENTS

Our thanks to the experts who have contributed towards development of the template.

9. REFERENCES

[1] D. E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching. Reading, MA: Addiso Wesley,1973 Structures..

[2]Quadratic Research Paper International Journal of Computer Applications (975– 8887) Volume 65– No.14, March 2013

[3] Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C.(2003) Introduction to Algorithms MIT Press, Cambridge, MA, 2nd edition.

[4] Knight, W.: Search in an ordered array having variable probe cost. SIAM J. Comput. 17(6), 1203–1214 (1988).

[5] D. Coppersmith, "Fast evaluation of logarithms in finite fields of characteristic two," IEEE Trans. Inform. Theory, vol. IT-30, pp. 587-594, 1984.

[6] Ben-Asher, Y., Farchi, E., Newman, I.: Optimal search in trees. SIAM J. Comput. 28(6), 2090–2102 (199Carmo, R., Donadelli, J., Kohayakawa, Y., Laber, E.: Searching in random partially ordered sets.Theor. Comput. Sci. 321(1), 41–57 (2004)

[7] Knight, W.: Search in an ordered array having variable probe cost. SIAM J. Comput. 17(6), 1203–1214 (1988)

[8] Navarro, G., Baeza-Yates, R., Barbosa, E., Ziviani, N., Cunto, W.: Binary searching with nonuniform costs and its application to text retrieval. Algorithmica 27(2), 145 169(2000)

[9] The -Version of Binary Search Trees: An Average Case Analysis, Hindawi Publishing CorporationISRN Combinatorics,Volume 2013, Article ID 450627, 8 pages,http://dx.doi.org/10.1155/2013/450627

[10] Machine Vision and Applications,DOI 10.1007/s00138-013-0483-3A 3-degree of freedom binary search pose estimation technique,Robert Ross · Andrew Martchenko John Devlin,Received: 18 September 2011 / Revised: 29 October 2012 / Accepted: 11 january 2013,© Springer-Verlag Berlin Heidelberg 2013

[11] IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-5, NO. 4, JULY 1979,Multidimensional Binary Search Trees in Database Applications,JON L. BENTLEY, MEMBER, IEEE

[12] Non-blocking Binary Search Trees, PODC'10, July 25–28,2010, Zurich, Switzerland.Copyright 2010 ACM 978-1-60558-888-9/10/07 ...$10.00.

[13] A Non-Blocking Internal Binary Search Tree,SPAA’12, June 25–27, 2012, Pittsburgh, Pennsylvania, USA.Copyright 2012 ACM 978-1-4503-1213-4/12/06

[14] Noisy binary search and its applications,Computer ScienceDivision, University of California, Berkeley. Supported I part by NSF grant CCF-0515259 karp@icsi.berkeley.edu, rdk@cs.cornell.edu