T-Search: An advancement of binary search

Arihant Jain, Hriday Kumar Gupta

***Abstract*— Searching refers to the process of finding a data value within some given set of data values in the form of list or array or any other form. Searching has obtained so much concern on a global scale the reason being the increasing amount of data day by day so there is a need to reduce the searching time and thus we require efficient searching methodology. Taking the binary search as a base for the searching technique goal is to find a required element within an ordered list by making comparisons. Size of list reduces after each iteration. This paper proposes a new algorithm for searching. The main point of difference is that it uses variable partitioning within the list for the first three iterations and again start repeating the same procedure. Binary Search can be improved by increasing the number of sublists but doing so will increase the number of comparisons thus not a good idea. The proposed algorithm overcomes this drawback of breaking into more sublists by variable partitioning.**

***Index Terms*— Binary search, Better variable partition**

I. INTRODUCTION

In computer science, searching within an ordered data structure is a very common problem that has various variants but we need to have one that can give best results with minimum time and space complexity and the number of sublists should also be less. Here we are taking binary search as the base but not referring to quadratic search as it will increase the number of sublists.

In binary search after placing the items in an array, they are sorted either in ascending or descending order and then after that compare key with the middle element if a match is found then return immediately otherwise check whether the key is lesser or greater than a middle element of the array. If the key is less then repeat the same procedure on the first half of list else on the second half of list.

A. *Analysis Of Binary Search*

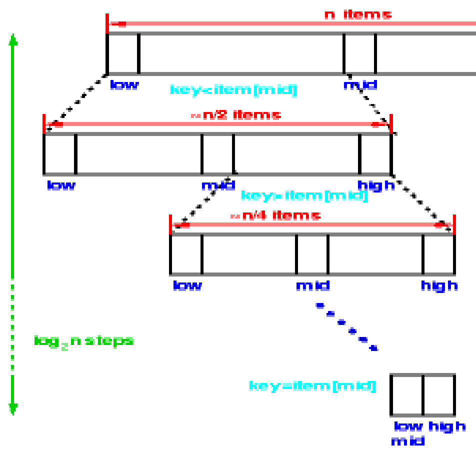


Fig 1. Analysis of binary search

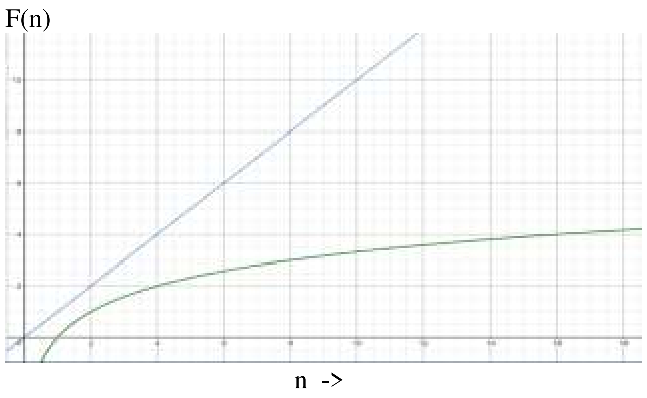
Arihant Jain, B.Tech, in Krishna Institute of Engineering and Technology, Ghaziabad
Hriday Kumar Gupta, Asst. Professor in Krishna Institute of Engineering and Technology, Ghaziabad

The complexity of linear search is **O(n)**.
After every iteration, the list is divided into 2 equal parts. Therefore n items can be divided into two parts almost **$\log_2 n$** times. Hence, running time of binary search is **O ($\log_2 n$)**.

Binary Search: If we calculate the size of the obtained list after 3 iterations from binary search:
0.5*0.5*0.5=0.125
12.5% (**13% approximately**)

Hence, in every three iterations the list size is reduced to the 13% of the original.

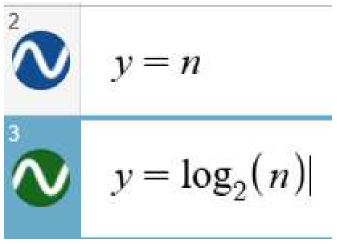
B. *Performance Graph Of Binary Search*



n: number of elements

Fig 2. Performance graph of binary search

2 Linear Search
3 Binary Search



2 $y = n$
3 $y = \log_2(n)$

II. PROPOSED ALGORITHM- T SEARCH

In the proposed algorithm after placing the items in an array, they are sorted either in ascending or descending order and then compare key with P1 and P2 position elements of the array. If there is a match then return it immediately else check whether the key is lesser or greater than P1 and P2 position elements of the array. After this, we will get the sublist in which key will lie and then repeat the same procedure on that sublist. Here, same procedure refers to the process of finding P1 and P2 again and then check in which sublist key will be

present. As in this algorithm, we are using variable partitioning in each and every iteration so basically, after three sub-iterations the process repeats again. Every iteration comprises three sub-iterations.
The first sub-iteration divides the list in the ratio as 25:50:25. Second sub-iteration divides the list in the ratio as 35:30:35. The third sub-iteration divides the list in the ratio as 45:10:45. After performing these three sub-iterations we are done with one iteration of the proposed algorithm. If till this step we are unable to find the key then we will perform this procedure again to get the position of the key.
Certain conditions used in the algorithm are given below:

P1 = first + (25 + I) * (last – first) / 100;
P2 = first + last - p1;

Here,
FIRST: start index
LAST: end index
I: a parameter that varies partition with each iteration

A. *Algorithm*

```
int i=0;

TSearch (int FIRST, int LAST)
{
 if(FIRST <= LAST)
 {
   i=i+10;
       if(i==30)
           i=0;

int p1= FIRST + ( 25 + i )  * ( LAST – FIRST )  /100;
     int p2= FIRST + LAST -p1;

     if(a[p1]==x)
     {
   return p1;
      }

     else if(a[p2]==x)
     {
        return p2;
     }
          else if(x>a[p1] &&x<a[p2])
     {
       FIRST = p1+1;
       LAST=p2-1;
       return TSearch (FIRST,LAST);
     }
      else if(a[p1]>x)
      {
        LAST=p1-1;
        return TSearch (FIRST,LAST);
      }
           else if(a[p2]<x)
       {
        FIRST = p2+1;
        return TSearch (FIRST,LAST);
     }
  }
 return -1;
}
```

The function will return -1 if element not found and if the element is found at P1 or P2 then it will return the index of the key. The given Algorithm will check 3 conditions after calculating P1, P2 when the key is not found at the P1 and P2 :

Condition 1: IF KEY 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 sublist.
Condition 2: IF KEY is greater than P2. Then (P2+1) will become the FIRST and then the procedure will be repeated for sublist.
Condition 3: IF KEY is less than P1.Then (P1 -1) will become the LAST procedure will be repeated for sublist.

B. *T-Search Overview*

T-Search Overview: In this algorithm, we proceed as follows:

Step-1

1	2	3
25%	50%	25%

The only 1/3rd probability of getting 50% part of the list in worst case, while in Binary Search 50% of the list is always obtained after partition.

Step-2

1	2	3
35%	30%	35%

Step-3

1	2	3
45%	10%	45%

Everytime partition % from start and end is increased by 10% to 45% and then repeats from 25% again.

- We are given a list of sorted records.
- Given sorted list is divided into 3 sublists with the variable partition.

T-Search: Considering worst case for obtaining maximum possible list in three iterations so,
50% , 35 % , 45% .
=0.50*0.35*0.45
=0.07875
=7.875 % (**approximately 8 %**)

- So, a total number of possible combinations of dividing list are 3^3=27.
- After three iterations in t-search eight distinct partitions are :-
  25 * 35 * 45
  25 * 35 * 10
  25 * 30 * 45
  25 * 30 * 10
  50 * 35 * 45
  50 * 35 * 10
  50 * 30 * 45
  50 * 30 * 10
- For 27 cases the result is reduced by 6.12 % in comparison to equal ternary search which gives 3.9304% of list after 3 iterations.

C. *Analysis Of T-Search*

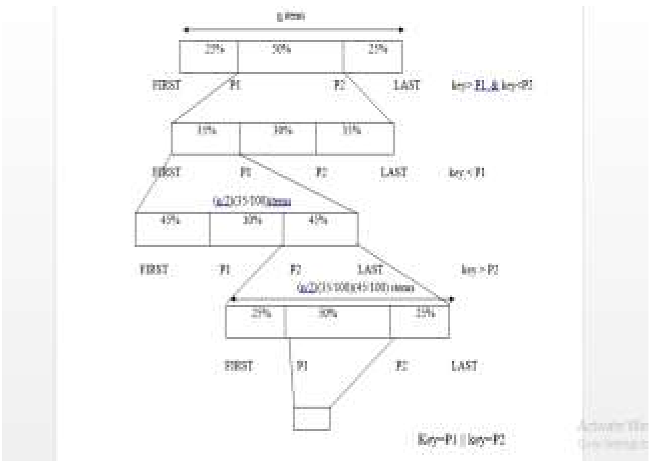
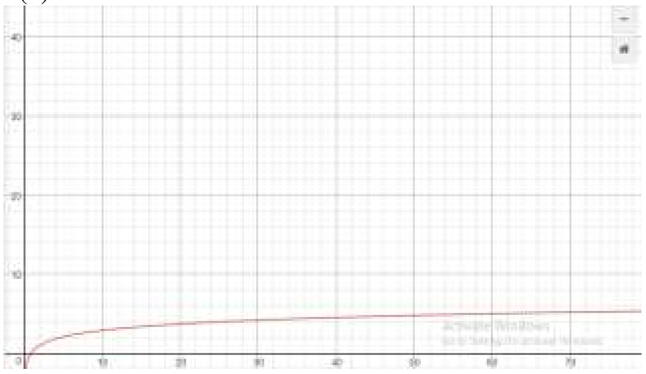


Fig 3. Analysis of T-search

D. *Performance Graph Of T-Search*

<u>In worst case:</u>

F(n)



n →

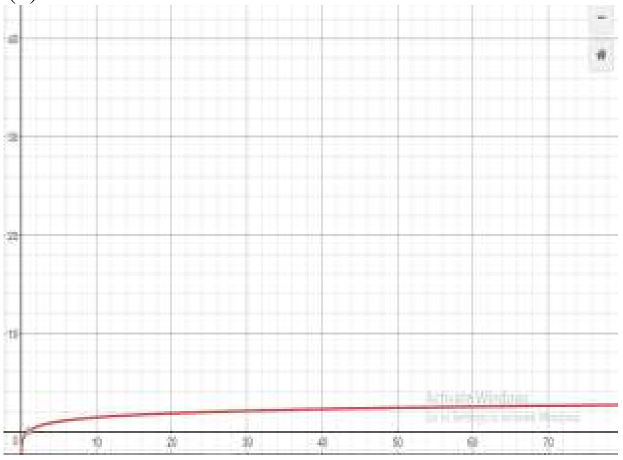
n: number of elements

Fig 4. Performance graph of T search

$$Y = 3 \log_{12.69} (400\, n \;/\; 63) - 2$$

<u>In best case:</u>

F(n)



n →

n: number of elements

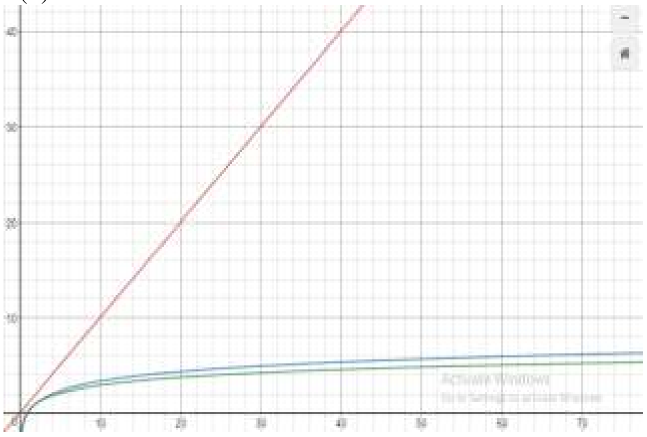
Fig 5. Performance graph of T search

$$Y = 3 \log_{133.3} (n)$$

III. Comparison Graph Of Linear, Binary, And T-Search

This section will compare the performance of the linear, binary, quadratic and t-search. The division of the list is shown diagrammatically and the performance graph of all the three approaches is also present. The performance is plotted between Time " f(n)" and a number of elements. As the number of elements is increased the time will decrease and it will be clear to distinguish the difference of performance among the Linear Search, Binary Search, the Quadratic Search, and T-Search technique.

F(n)

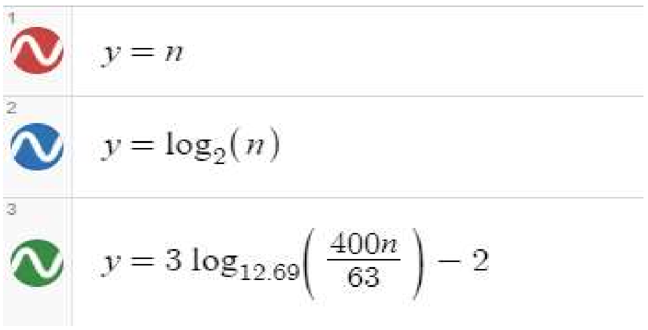


n →

n: number of elements

Fig 6. Comparison graph

1. Linear Search
2. Binary Search
3. T-search



1 $y = n$

2 $y = \log_2(n)$

3 $y = 3\log_{12.69}\left(\dfrac{400n}{63}\right) - 2$

IV. Results And Discussion

We are using a sorted array of 5000 elements distributed uniformly. Binary search and proposed algorithm are applied to find the same element and result of approaches is shown below along with a number of steps in comparison.

ARRAY :

INDEX:

0 1 2 ….. …… 4998 4999

VALUE:

1	**2**	**3**	**…….**	**….**	**4999**	**5000**

Table 1: Search key = 2

Steps	**Binary**	**T-Search**
Initial	Low=0 mid=2499 high=4999	Low=0 p1=1249 p2=3750 high=4999
1.	Low=0 mid=1249 high=2498	Low=0 p1=436 p2=812 high=1248
2	Low=0 mid=624 high=1248	Low=0 p1=195 p2=240 high=435
3	Low=0 mid=311 high=623	Low=0 p1=48 p2=146 high=194
4	Low=0 mid=155 high=310	Low=0 p1=16 p2=31 high=47
5	Low=0 mid=77 high=154	Low=0 p1=6 p2=9 high=15
6	Low=0 mid=38 high=76	Low=0 p1=1 p2=4 high=5
7	Low=0 mid=18 high=37	
8	Low=0 mid=8 high=17	
9	Low=0 mid=3 high=7	
10	Low=0 mid=1 high=2	
Total:	**10**	**6**

Table 2 : Search key = 4999

Steps	**Binary**	**T-search**
Initial	Low=0 mid= 2499 high= 4999	Low= 0 p1= 1249 p2= 3750 high= 4999
1	Low=2500 mid=3749 high=4999	Low=3751 p1=4187 p2=4563 high=4999
2	Low=3750 mid=4374 high=4999	Low=4564 p1=4759 p2=4804 high=4999
3	Low=4375 mid=4687 high=4999	Low= 4805 p1=4853 p2=4951 high=4999
4	Low=4688 mid=4843 high=4999	Low=4952 p1=4968 p2=4983 high=4999
5	Low=4844 mid=4921 high=4999	Low= 4984 p1=4990 p2=4993 high=4999
6	Low=4922 mid=4960 high=4999	Low= 4994 p1=4995 p2=4998 high=4999
7	Low=4961 mid=4980 high=4999	
8	Low=4981 mid=4990 high=4999	
9	Low=4991 mid=4995 high=4999	
10	Low=4996 mid=4997 high=4999	
11	Low= 4998 mid=4999 high=4999	
Total:	**11**	**6**

V. Conclusion

Here, in this paper, we present a new algorithm for searching and had been implemented to search within the sorted linear arrangement of items with worst-case complexity as **$O(3\log_{12.69}(400n/63)–2)$.** The technique that had been used to implement this algorithm involves multiple splitting along with variable partitioning. The performance graph for linear, binary and t-search is also present along with their comparison graph. Thus we see how efficient is to use this algorithm it has minimum worst-case complexity. As we see it has reduces the list to 7.8% in comparison to 12.5% in binary search. Though it is true that in quadratic list reduces to 6.25% but the number of sublists is large which is a drawback in the quadratic search algorithm. Hence, we are reducing list keeping a number of sublists also minimum.

REFERENCES

[1] Quadratic Research Paper
International Journal of Computer Applications (0975 – 8887) Volume 65– No.14, March 2013

[2] Binary Search Algorithm
https://en.wikipedia.org/wiki/Binary_search_algorithm

[3] Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein (2009).
Introduction To Algorithms, Third Edition. MIT Press

Arihant Jain is a student in B.Tech, in Krishna Institute of Engineering and Technology, Ghaziabad. He has done a lot of project in Computer Science & Engineering field. He is a dedicated and compassionate student committed to the field of research. He is the author of this Research Paper.

Hriday Kumar Gupta is an Asst. Professor in Krishna Institute of Engineering and Technology, Ghaziabad. He is working in the field of Computer Science & Applications.