ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

A Comparative Analysis of Three Different Types of Searching Algorithms in Data Structure

Debadrita Roy[1], Arnab Kundu[2]

Trainer, Ghani Khan Chowdhury Institute of Engineering & Technology, Malda, West Bengal[1]

Assistant Professor, Birbhum Institute of Engineering & Technology, Suri, Birbhum, West Bengal[2]

Abstract: Searching is a process of checking and finding an element from a list of elements. Although there are huge numbers of searching algorithms are available. But here our work intends to show an overview of comparison between three different types of searching algorithms. We have tried to cover some technical aspects of Linear or Sequential search, Binary Search and Interpolation Search. This research provides a detailed study of how all the three algorithms work & give their performance analysis with respect to time complexity.

Keywords: Complexity**,** Linear Search, Binary Search, Interpolation search, time complexity

I. INTRODUCTION

In the present scenario an algorithm and data structure play a significant role for the implementation and design of any software [1]. An algorithm is a finite sequential set of instructions which, if followed, accomplish a particular task or a set of tasks in a finite time.

Complexity: The complexity of an algorithm is a function g(n) that gives the upper bound of the number of operations performed by an algorithm when the input size is n. Complexity are divided in two ways. *Time complexity* is the amount of time the computer requires to execute the algorithm& Space complexity of an algorithm is the amount of memory space the computer requires, completing the execution of the algorithm. In case of algorithm searching is the process to finding to location of the given data elements in the data structure. The different types of searching techniques are Linear search, Binary search.

Linear (Sequential) Search is the basic and simple method of searching*: It* is a method where the search begins at the end of the list, scans the elements of the list from left to right until the desired record is found.
In *Binary search* the entire sorted list is divided into two parts. We first compare our input item with the mid element of the list & then restrict our attention to only the first or second half of the list depending on whether the input item comes left or right of the mid element. In this way we reduce the length of the list to be searched by half. Less time is taken by binary search to search an element from the sorted list of elements .So we can conclude that binary search method is more efficient than the linear search. Binary search algorithm is efficient because it is based on divide-and-conquer strategy; which divides the list into two parts and searches one part of the list thereby reducing the search time [2].

Interpolation Search (sometimes referred to as extrapolation search) is an *algorithm for searching* for a given key value in an indexed array that has been ordered by the values of the key. In this paper, mainly we have reviewed three searching algorithms .The concept of three searching techniques with examples, algorithm of each searching process, time complexity have discussed here. Finally, conclusions were presented in section 5.

II. CONCEPT BEHIND SEARCHING PROCESS

In linear search, each element of an array is read one by one sequentially and it is compared with the desired element. A search will be unsuccessful if all the elements are read and the desired element is not found. Where Binary search is an extremely efficient algorithm when it is compared to linear search. Binary search technique searches "data" in minimum possible comparisons. Suppose the given array is a sorted one, otherwise first we have to sort the array elements. Then apply the following conditions to search a "data".

1) Find the middle element of the array (*i.e*., *n*/ 2 is the middle element if the array or the sub-array contains *n* elements).
2) Compare the middle element with the data to be searched and then there are following three cases.

a) If it is a desired element, then search is successful.
b) If it is less than desired data, then search only the first half of the array, i.e., the elements which come to the left side of the middle element.
c) If it is greater than the desired data, then search only the second half of the array, i.e., the elements which come to the right side of the middle element.

Repeat the same steps until an element are found or exhaust the search area. Again For searching an ordered array, Interpolation search is used. This method is even more efficient than binary search, if the elements are uniformly distributed (or sorted) in an array A. Interpolation search is a method of retrieving a desired record by key in an ordered file by using the value of the key and the statistical distribution of the Keys [3].

Consider an array A of n elements and the elements are uniformly distributed (or the elements are arranged in a sorted array). Initially, as in binary search, low is set to 0

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

and high is set to n – 1. Now we are searching an element key in an array between A[low] and A[high].

The key would be expected to be at mid, which is an approximately position. mid = low + (high – low) × ((key – A[low])/ (A[high] – A[low])). If key is lower than A[mid], reset high to mid–1; else reset low to mid+1. Repeat the process until the key has found or low > high.

III. PROCEDURE OF SEARCHING

A. Linear or Sequential Search :

Unsorted array

5 4 21 16 25 3	15
5 4 21 16 25 3	15
5 4 21 16 25 3	15
5 4 21 16 25 3	15
5 4 21 16 25 3	15
5 4 21 16 25 3	15
5 4 21 16 25 3	15

Conclusion: The element 15 is not present inside that array.

Sorted array

4 5 9 11 13	10
4 5 9 11 13	10
4 5 9 11 13	10
4 5 9 11 13	10
4 5 9 11 13	10

Conclusion: The element 10 is not present inside that

B. Binary Search :

Suppose we have an array of 7 elements

9	10	25	30	40	45	70
0	1	2	3	4	5	6

Following steps are generated if we binary search a data = 45 from the above array.

***Step* 1:**

LB UB

9	10	25	30	40	45	70
0	1	2	3	4	5	6

LB = 0;
UB = 6 ;
mid = (0 + 6)/ 2 = 3
A[mid] = A[3] = 30

***Step* 2:**

Since (A[3] < data) - *i.e.*, 30 < 45 – reinitialise the variable LB, UB and mid

LB UB

9	10	25	30	40	45	70
0	1	2	3	4	5	6

LB = 3;
UB = 6;
mid = (3 + 6)/ 2 = 4
A[mid] = A [4] = 40

***Step* 3:**

Since (A [4] < data) - *i.e.*, 40 < 45 - reinitialise the variable LB, UB and mid

LB UB

9	10	25	30	40	45	70
0	1	2	3	4	5	6

LB = 4;
UB = 6;
mid = (4 + 6)/ 2 = 5
A[mid] = A [5] = 45

Step 4:

Since (A [5] == data) - *i.e.*, 45 == 45 - searching is successful.

C. Interpolation Search :

Consider 7 numbers.

2, 25, 35, 39, 40, 47, 50

Step1:

Suppose we are searching 50 from the array.
Here n = 7
Key = 50
low = 0
high = n – 1 = 6
mid = 0+(6–0) × ((50–2)/ (50–2))
= 6 × (48/ 48)
= 6 if (key == A[mid]) ⇒ key == A [6]
⇒ 50 == 50 ⇒ key is found.

Step 2:

Say we are searching 25 from the array
Here n = 7
Key = 25
low = 0
high = n – 1 = 6
mid = 0 + (6–0) × ((25–2)/ (50–2))
= 6 × (23/ 48)
= 2.875
Here we consider only the integer part of the midi.e.,
mid = 2 if (key == A[mid]) ⇒ key == A[2]⇒ 25 == 25 ⇒ key is found.

Step 3:

Say we are searching 34 from the array
Here n = 7
Key = 34
low = 0
high = n – 1 = 6
mid = 0 + (6 – 0) × ((34 – 2)/ (50– 2))
= 6 × (32/ 48)
= 4
if(key < A[mid]) ⇒ key < A[4] ⇒ 34 < 40 so reset
high = mid–1⇒ 3
low = 0
high = 3
Since(low < high)
mid = 0+(3–0) × ((34–2)/ (39–2))
= 3 × (32/ 37)
= 2.59
Here we consider only the integer part of the midi.e.,
mid = 2
if (key < A[mid])

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

⇒key< A[2] ⇒ 34 < 35 so reset
high = mid–1 ⇒ 1
low = 0
high = 1
Since (low < high)
mid = 0+(1–0) × ((34–2)/ (25–2))
= 3 × (32/ 23)
= 1 here (key > A[mid])
⇒key> A[1] ⇒ 34 > 25
so reset low = mid+1
⇒ 2
low = 2
high = 1
Since (low > high)
So" The key is not in the array"

IV. ALGORITHM

A. *Linear or Sequential Search :*

Let A be an array of *n* elements, A[1],A[2],A[3], ... A[*n*]. "*data*" is the element to be searched. Then this algorithm will find the location "*loc*" of data in A. Set *loc* = – 1,if the search is unsuccessful.

1. Input an array A of n elements and "data" to be searched and initialise *loc* = – 1.
2. Initialise *i* = 0; and repeat through step 3 if (*i*<*n*) by incrementing *i* by one .
3. If (data = A[*i*]) (*a*) loc = *i*
4. (*b*) GOTO step 4
5. If (loc> 0)

a. Display "data is found and searching is successful"

6. Else

a. Display "data is not found and searching is unsuccessful"

7. Exit

B. *Binary Search :*

Let A be an array of n elements A [1], A [2], A [3],.....A[*n*]. "Data" is an element to be searched. "mid" denotes the middle location of a segment (or array or sub-array) of the element of A. LB and UB is the lower and upper bound of the array which is under consideration.

1. Input an array A of n elements and "data" to be sorted
2. LB = 0, UB = *n*; mid = int ((LB+UB)/ 2)
3. Repeat step 4 and 5 while (LB <= UB) and (A[mid]! = data)
4. If (data < A[mid]) UB = mid–1
5. Else LB = mid + 1
6. Mid = int ((LB + UB)/ 2)
7. If (A[mid] == data) Display "the data found"
8. Else Display "the data is not found"
9. Exit

C. *Interpolation Search :*

Suppose A be array of sorted elements and key is the elements to be searched and low represents the lower bound of the array and high represents higher bound of the array.

1) Input a sorted array of *n* elements and the key to be searched
2) Initialise low = 0 and high = n – 1
3) Repeat the steps 4 through 7 until if(low < high)
4) Mid = low + (high – low) × ((key – A[low]) / (A[high] – A[low]))
5) If(key < A[mid])
   high = mid–1
6) Elseif (key > A[mid]) low = mid + 1
7) ElseDisplay " The key is not in the array"
8) STOP

V. OBSERVATIONS

A. *Linear or Sequential Search :*

Time Complexity of the linear search is found by number of comparisons made in searching a record.

Suppose there are n elements in the list. The following expression, gives average number of comparisons
„ 1+2+ …..+n / n
„ 1+2+ …..+n = n(n+1)/2
„ Thus the following expression gives the average number of comparisons made by sequential search in successful case :

$$\frac{1+2+\ldots\ldots+n}{n} = \frac{1n(n+1)}{2} = \frac{n+1}{2}$$

In the best case, the desired element is present in the first position of the array, *i.e*., only one comparison is made. So *f* (*n*) = O(1). In the Average case, the desired element is found in the half position of the array, then *f* (*n*) = O[(*n* + 1)/ 2]. But in the worst case the desired element is present in the nth (or last) position of the array, so n comparisons are made. So *f* (*n*) = O(*n* + 1)

B. *Binary Search :*

Time Complexity is measured by the number *f* (*n*) of comparisons to locate "data" in A, which contain *n* elements. Each step of the algorithm divides the block of items being searched in half. We can divide a set of ***n*** items in half at most $\log_2 n$ times. Thus the running time of a binary search is proportional to **log *n*** and we say this is a ***O*(log *n*)** algorithm. Observe that in each comparison the size of the search area is reduced by half. Hence in the worst case, at most $\log_2 n$ comparisons required. So *f* (*n*) = O([$\log_2 n$]+1). Time Complexity in the average case is almost approximately equal to the running time of the worst case. Binary search requires a more complex program than our original search and thus for *small* ***n*** it may run slower than the simple linear search. However, for large ***n***,

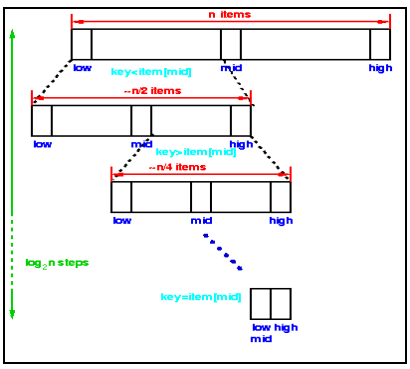

Fig. 1. : Pictorial representation of Binary Search Technique

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 5, May 2014

$$\lim_{n\to\infty} \log n / n$$

At large ***n***, **log *n*** is *much* smaller than ***n***, consequently an ***O*(log *n*)** algorithm is *much* faster than an ***O(n)*** one.

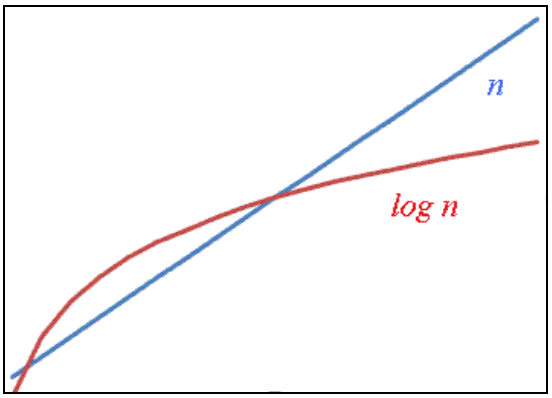

Fig. 2. : Plot of *n* and log *n v/s n*

C. Interpolation Search :

On average the interpolation search makes about log(log(*n*)) comparisons(if the elements are uniformly distributed), where *n* is the number of elements to be searched. In the worst case (for instance where the numerical values of the keys increase exponentially) it can make up to O(*n*) comparisons.

Using big-O notation, the performance of the interpolation algorithm on a data set of size *N* is *O*(*N*); however under the assumption of a uniform distribution of the data on the linear scale used for interpolation, the performance can be shown to be *O*(log log *N*).

The expected running time of interpolation search on random files (generated according to the uniform distribution) of size n is 0(log log n). This was shown by Yao and Yao [4], Pearl et al. [5]. A very intuitive explanation of the behaviour of interpolation search can be found in Pearl and Reingold [6].

VI. CODES

A. Linear or Sequential Search :

```
#include <stdio.h>
 int main()
{
int array[100], search, c, n;
printf("Enter the number of elements in array\n");
scanf("%d",&n);
printf("Enter %d integer(s)\n", n);
for (c = 0; c < n; c++)
scanf("%d", &array[c]);
 printf("Enter the number to search\n");
scanf("%d", &search);
for (c = 0; c < n; c++)
   {
if (array[c] == search)     /* if required element found */
     {
printf("%d is present at location %d.\n", search, c+1);
break;
     }
 }
if (c == n)
printf("%d is not present in array.\n", search);
 return 0;
}
```

B. Binary Search :

```
#include <stdio.h>
int main()
{
int c, first, last, middle, n, search, array[100];
printf("Enter number of elements\n");
scanf("%d",&n);
printf("Enter %d integers\n", n);
for ( c = 0 ; c < n ; c++ )
scanf("%d",&array[c]);
printf("Enter value to find\n");
scanf("%d",&search);
first = 0;
last = n - 1;
middle = (first+last)/2;
while( first <= last )
{
if ( array[middle] < search )
first = middle + 1;
else if ( array[middle] == search )
{
printf("%d found at location %d.\n", search, middle+1);
break;
}
else
last = middle - 1;
middle = (first + last)/2;
}
if ( first > last )
printf("Not found! %d is not present in the list.\n",
search);
return 0;
}
```

C. Interpolation Search :

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[25],n,mid,low,high,f=0,item,i;
printf("Enter the size of the array");
scanf("%d",&n);
printf("Enter the elements in sorted order");
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
printf("Enter the item to be searched for");
scanf("%d",&item);
low=0;
high=n-1;
while(low<=high)
{
mid=(low+(high-low)*((item-a[low])/(a[high]-a[low])));
if(a[mid]==item)
{
printf("\n\nItem found at position %d",mid);
```

```
f=1;
break;
}
else if(a[mid]>item)
{
high=mid-1;
}
else
{
low=mid+1;
}}
if(f==0)
printf("\n\nItem not found in the array");
getch();
}
```

VII. CONCLUSION

Searching is very important for huge application such as database management system. In this work, we have tried to summarize three searching algorithms and also included searching concept, algorithm, coding & time complexity. Coding part is implemented in C language. Rather we have also discussed about the advantages of binary search algorithm with respect to others for a given problem.

ACKNOWLEDGMENT

The authors would like to thank the authorities of Ghani Khan Chowdhury Institute of Engineering & Technology, Malda, West Bengal & Birbhum Institute of Engineering & Technology, Suri, Birbhum-731101, West Bengal for providing every kind of supports and encouragement during the working process.

REFERENCES

[1] Ms.NidhiChhajed Assistant Professor C.S.E Dept. PIES, Mr. Imran Uddin Assistant Professor, C.S.E Dept. PIES, Mr.Simarjeet Singh Bhatia Assistant Professor, C.S.E Dept. PIES Indore (M.P), India. A Comparison Based Analysis of Four Different Types of Sorting Algorithms in Data Structures with Their Performances.

[2] Asagba P. O, Osaghae E. O. and Ogheneovo, E. E.Department of Computer Science, University of Port Harcourt, Choba, Port Harcourt, Rivers State. Is Binary search technique faster than Linear search technique?

[3] Yehoshua Perl Bar-Ilan University and The Weizmann Institute of Science,AlonItaiTechnion--Israel Institute of Technology, HaimAvni The Weizmann Institute of Science. Interpolation Search A Log LogNSearch.

[4] YAO, A. C., AND YAO, F. F. The complexity of searching an ordered random table. InP~ocadmgs of the 17th .4 WILLC11 Symposium OH the Fowz&ztzotzs of Computer Science. 1EEE, NewYork, 1976, pp. 173-175.

[5] PEARL, Y., ITAI, A., ANDAVNI, H. Interpolation search—A log log N search. Commun. ACM 21, 7, (1978), 550–554

[6] PEARL. Y., AND REINGOLD, E. M. Understanding the complexity of interpolation search. lnf.Proc. Left, 6.6 (1977), 219–222

BIOGRAPHIES

Debadrita Roy, born in India, obtained her M.Tech degree from Heritage Institute of Technology, Kolkata, W.B. Now she is Trainer of Information Technology Department in Ghani Khan Chowdhury Institute of Engineering & Technology, Malda-732101, West Bengal, India. She is Life member of International Association of Engineers India.

Arnab Kundu, born in India, obtained his M.Tech degree from West Bengal University of Technology, W.B. Now he is Asst. Professor of E.C.E. Department in Birbhum Institute of Engineering & Technology, P.O.:Suri, Dist.:Birbhum, PIN:731101 and also doing his research work from CSIR-CMERI, Durgapur-713209, West Bengal, India. He is member of IEEE & also Life member of Indian Society for Technical Education, Indian Society of Remote Sensing, International Association of Engineers, The Institution of Electronics & Telecommunication Engineers & Society for EMC Engineers (India).