Searching

Searching is a process of finding the location of a specific data element within an array of size n.

Types of Searching

(1) Linear Search or Sequential Search

(2) Binary Search

(3) Interpolation Search

Linear Search

- 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.

- It is the simplest way for finding an element in a list.
- It searches the element sequentially in a list, no matter whether list is sorted or unsorted.

15	10	12	8	22	20	11	25
0	1	2	3	4	5	6	7

item = 15 :-> loc = 0 (Best case)

item = 25 :-> loc = 7 (Worst case)

item = 22 :-> loc = 4 (Average case)

C program to implement Linear Search

```
#include <stdio.h>
void main()
{
  int a[10], n, i, found=0, item;
  printf(" How many elements: ");
  Scanf("%d", &n);
  printf(" Enter array elements: ");
  for (i=0; i<n; i++)
   {
    scanf("%d", &a[i]);
   }
  printf(" Enter element to be searched: ");
  Scanf("%d", &item);
  for(i=0; i<n ; i++)
   {
       if (a[i] == item)
        {
         printf(" Element Found at loc : %d", i);
         found =1;
         break;
        }
     }
     if ( ! found)
        {
         printf(" Not found");
        }
  }
```

Analysis of Linear Search

Three Cases :

(1) {Best Case} :-> The best case is that in which the element is found during the first comparison.

(2) {Worst Case} :-> The worst case is that in which the element is found only at the end i.e in last comparison.

(3) {Average case} :-> The average case is that in which the element is found in comparison more than the best case but less than worst care.

∴ Time Complexity of Linear Search is

Best case $= \Omega(1)$

Worst case $= O(n)$

Average case $= \dfrac{\sum \text{All cases}}{\text{No. of cases}}$

$$= \frac{1+2+3+\dots+n}{n}$$

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

$$= \frac{n+1}{2}$$

$$\simeq O(n)$$

Drawbacks of Linear Search

(1) It is a very slow process.
(2) It is used only for small amount of data.
(3) It is a very time consuming method.

Binary Search

- Fast Search algorithm with time complexity of $O(\log n)$.
- Based on divide and conquer.
- Data item should be in sorted order.

General Idea

1. Find the middle element of the array.
2. Compare the element to be search with the middle element.

   (a) If it is the desired element, then search is successful.

   (b) If the element to be search is less than the middle element, 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 the element to be search is greater than the middle element, then search only the second half of the array i.e the elements which come to the right side of the middle element.

0	1	2	3	4	5	6	7	8	9	10
2	17	23	34	56	61	69	78	84	91	98

low=0 mid=5 high=10

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

$$= \frac{0+10}{2}$$

$$mid = 5$$

Algorithm :->

```
1. Input an array of n elements having sorted values.
2. Set low = 0 and high = n-1
3. mid = int((low+high)/2)
4. Repeat step 5 and 6 while (low <= high)
5. if (key < a[mid])
           high = mid-1;
   elseif (key > a[mid])
           low = mid + 1;
6. else if (key == a[mid])
           printf("Element found at loc: %d ", mid);
   else
           printf(" Element is not found");
7. Exit.
```

C Implementation of Binary Search

```
#include<stdio.h>
void main()
{
 int a[20], n, i, key, low, high, mid;
 printf(" How many elements : ");
 scanf(" %d ", & n);
 printf( " Enter array elements : "); /* Sorted elements */
 for (i=0; i<n; i++)
   {
    scanf(" %d", & a[i]);
   }
printf (" Enter element to be searched : ");
scanf ("%d", & Key);
 low = 0;
 high = n-1;
 while ( low <= high)
 {
    mid = int((low+high)/2);
    if ( Key < a[mid])
      {
        high = mid-1;
      }
    else if ( Key > a [mid])
      {
        low = mid +1;
      }
    else if ( Key == a [mid])
      {
      printf ("Element found at loc : %d ", mid);
      break;
      }
      else
        printf (" Element not found ");
    }
}
```

Analysis of Binary Search

Best case Time Complexity $= \Omega(1)$

i.e mid is the desired element to be searched. and it takes 1 comparison.

Worst case / Average case Time complexity $= O(\log n)$.

Drawbacks

The drawbacks of the Binary Search are :

(i) Binary Search requires that the items in the array should be sorted.

(ii) Binary Search cannot be used where there are many insertion or deletion.

Interpolation Search

- Interpolation Search is a type of searching algorithm.
- Interpolation Search is an improvement over Binary Search for scenarios where the values in a sorted array are uniformly distributed.
- Binary Search always check the value at the middle index. But Interpolation Search may check at different locations based on the value of element being searched.

NOTE :-> Interpolation search work efficiently when the array elements are sorted and uniformly distributed.

How Interpolation Search works

Given: Sorted array of n uniformly distributed values.

Goal: Search for element x.

Formula: $$Pos = low + \frac{x - a[low]}{a[high] - a[low]} \times (high - low)$$

Uniformly Distributed means 1, 3, 5, 7, 9, 11, 13, 15.

Not Uniformly Distributed means 1, 3, 5, 14, 21, 25, 26, 27.

Example:

0	1	2	3	4	5	6	7
1	3	5	7	9	11	13	15

Find position of 9.

Soln Here, low = 0 $\therefore$ a[low] = 1

high = 7 $\therefore$ a[high] = 15

$$\therefore \text{Pos of } 9 = low + \frac{x - a[low]}{a[high] - a[low]} \times (high - low)$$

$$= 0 + \frac{9-1}{15-1} \times (7-0)$$

$$= 0 + \frac{8}{\cancel{14}_2} \times \cancel{7}$$

$$= 4$$

$\therefore$ In 1 step, we are able to find the position of element 9.

But using Binary Search,

low = 0

high = 7

$$\therefore mid = \left\lfloor \frac{low + high}{2} \right\rfloor = \frac{0+7}{2} = 3$$

here, $9 > a[mid]$

$\therefore$ search in second half of array

So, low = mid + 1 = 4

high = 7

$$\therefore mid = \left\lfloor \frac{4+7}{2} \right\rfloor = \left\lfloor \frac{11}{2} \right\rfloor = 5$$

$9 < a[mid]$

$\therefore low = high = 4.$

$$mid = \left\lfloor \frac{low+high}{2} \right\rfloor = \frac{4+4}{2} = 4$$

$\therefore 9 == a[mid]$

So, to search element 9 using Binary Search, total steps taken = 3.

Hence, Performance of Interpolation Search is much better than Binary Search.

<u>Example</u>: Array is not so uniformly distributed

0	1	2	3	4	5	6	7
1	3	5	14	21	25	26	27

Find position of Element 25.

<u>Soln</u> Here, $low = 0$ $\quad \therefore a[low] = 1$

$high = 7$ $\quad \therefore a[high] = 27$

$$\therefore Pos = low + \frac{x - a[low]}{a[high] - a[low]} \times (high - low)$$

$$= 0 + \frac{25-1}{27-1} \times (7-0)$$

$$= \frac{24}{26} \times 7$$

$$= \frac{84}{13}$$

$$\simeq 6$$

$\because$ 25 < a[6] , So Search left half of array.

$\therefore$ low = 0 and high = 5

a[low] = 1   a[high] = 25

$$\therefore pos = low + \frac{x - a[low]}{a[high] - a[low]} \times (high - low)$$

$$= 0 + \frac{25-1}{25-1} \times (5-0)$$

$$= \frac{24}{24} \times 5$$

$$= 5$$

$\therefore$ 25 == a[5] , Search successful.

$\therefore$ Steps Required = 2.

Let's Run Binary Search on this array

low = 0   high = 7

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor = \frac{0+7}{2} = 3$$

$\therefore$ 25 > a[3] , Search element in second half of array.

So, low = mid + 1 = 4

high = 7

$$\therefore mid = \frac{4+7}{2} = \frac{11}{2} = 5$$

$\therefore$ 25 == a[mid] . Search Successful.

$\therefore$ By applying Binary Search, total steps taken $= 2$.

Hence, for this given array, the performance of Interpolation Search and Binary Search are same.

Analysis of Interpolation Search

(1) If the data is uniformly distributed, then time taken by Interpolation Search $<$ Binary Search.

(2) If the data is not uniformly distributed, then Interpolation Search takes more steps to find the position of element x.

(3) If the uniformity of data decreases, then the number of steps required for Interpolation Search will increase. So, the time complexity will also increase.

NOTE :-> Binary Search is not affected by uniformity of data and it takes $O(\log n)$ time to search element.