BINARY SEARCH

INTRODUCTION TO SEARCHING

- Searching in data structure refers to the process of finding the required information from a collection of items stored as elements in the computer memory.
- These sets of items are in different forms, such as an array, linked list, graph, or tree.
- Another way to define searching in the data structures is by locating the desired element of specific characteristics in a collection of items.

Searching Methods

- Searching in the data structure can be done by applying searching algorithms to check for or extract an element from any form of stored data structure. These algorithms are classified according to the type of search operation they perform, such as:
- **Sequential search** - The list or array of elements is traversed sequentially while checking every component of the set. For example – Linear Search.
- **Interval Search** - The interval search includes algorithms that are explicitly designed for searching in sorted data structures. In terms of efficiency, these algorithms are far better than linear search algorithms. Example- Logarithmic Search, Binary search.

These methods are evaluated based on the time taken by an algorithm to search an element matching the search item in the data collections and are given by,

- The best possible time
- The average time
- The worst-case time

The primary concerns are with worst-case times, which provide guaranteed predictions of the algorithm's performance and are also easier to calculate than average times.

LINEAR SEARCH

- Linear search is also called as sequential search algorithm.

- It is the simplest searching algorithm.
- In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found.
- If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.
- It is widely used to search an element from the unordered list, i.e., the list in which items are not sorted.
- The worst-case time complexity of linear search is O (n).

Steps used in the implementation of Linear Search

- First, we have to traverse the array elements using for loop.
- In each iteration of for loop, compare the search element with the current array element, and
  - If the element matches, then return the index of the corresponding array element.
  - If the element does not match, then move to the next element.
- If there is no match or the search element is not present in the given array, return -1.

Algorithm

Linear_Search(a, n, val) // 'a' is the given array, 'n' is the size of given array, 'val' is the value to search

Step 1: set pos = -1

Step 2: set i = 1

Step 3: repeat step 4 while i <= n

Step 4: if a[i] == val

set pos = i

print pos

go to step 6

[end of if]

set ii = i + 1

[end of loop]

Step 5: if pos = -1

print "value is not present in the array "

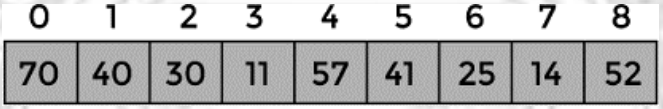
[end of if]

Step 6: exit

Working of Linear search

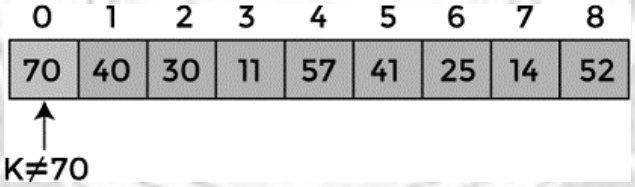
Consider an array of elements 70, 40, 30, 11, 57, 41, 25, 14, 52

Let the elements of array are

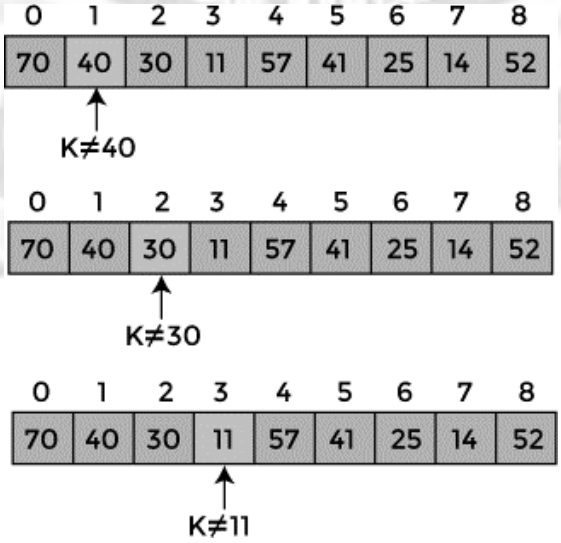


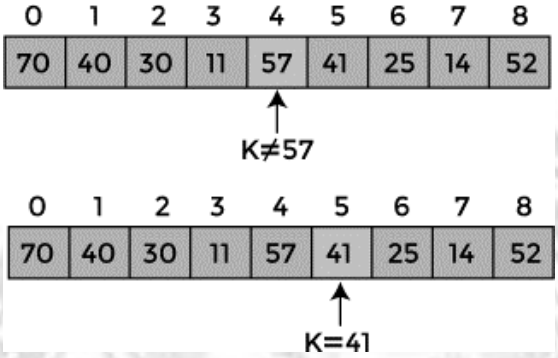
Let the element to be searched is **K = 41**

Now, start from the first element and compare **K** with each element of the array.



The value of **K**, i.e., **41**, is not matched with the first element of the array. So, move to the next element. And follow the same process until the respective element is found.





Now, the element to be searched is found. So algorithm will return the index of the element matched.

Linear Search complexity

Time Complexity	
Best	O(1)
Worst	O(n)
Average	O(n)
Space Complexity	O(1)

Applications of Linear Search Algorithm

- Linear search can be applied to both single-dimensional and multi-dimensional arrays.
- Linear search is easy to implement and effective when the array contains only a few elements.
- Linear Search is also efficient when the search is performed to fetch a single search in an unordered-List.

Advantages and Disadvantages

Sl. No.	Advantages	Disadvantages
1.	Will perform fast searches of small to medium lists	Time consuming for the enormous arrays.

2.	The list does not need to sorted	Slow searching of big lists
3.	Not affected by insertions and deletions	A key element doesn't matches any element then Linear search algorithm is a worst case

Example Program 5.5: Program for Implementation of Linear Search

```
#include <stdio.h>
int linearSearch(int a[], int n, int val) {
  // Going through array sequencially
  for (int i = 0; i < n; i++)
    {
      if (a[i] == val)
      return i+1;
    }
  return -1;
}
int main() {
  int a[] = {59, 40, 33, 11, 57, 41, 27, 18, 53}; // given
  arrayint val = 41; // value to be searched
  int n = sizeof(a) / sizeof(a[0]); // size of array
  int res = linearSearch(a, n, val); // Store
  result printf("The elements of the array are -
  ");
  for (int i = 0; i < n; i++)
  printf("%d ", a[i]);
  printf("\nElement to be searched is - %d", val);
  if (res == -1)
  printf("\nElement is not present in the array");
  else
  printf("\nElement is present at %d position of array", res);
```

```
  return 0;
}
```

Output

The elements of the array are - 59, 40, 33, 11, 57, 41, 27, 18, 53

Element to be searched is – 41

Element is present at 6 position of array

BINARY SEARCH

- Binary search is the search technique that works efficiently on sorted lists.
- Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.
- Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list.
- If the match is found then, the location of the middle element is returned.
- Otherwise, we search into either of the halves depending upon the result produced through the match.

Algorithm

Binary_Search(a, lower_bound, upper_bound, val) // 'a' is the given array, 'lower_bound' is the index of the first array element, 'upper_bound' is the index of the last array element, 'val' is the value to search

Step 1: set beg = lower_bound, end = upper_bound, pos = - 1

Step 2: repeat steps 3 and 4 while beg <=end

Step 3: set mid = (beg + end)/2

Step 4: if a[mid] = val

set pos = mid

print pos

go to step 6

else if a[mid] > val

set end = mid - 1

else

set beg = mid + 1

[end of if]

[end of loop]

Step 5: if pos = -1

print "value is not present in the array"

[end of if]

Step 6: exit

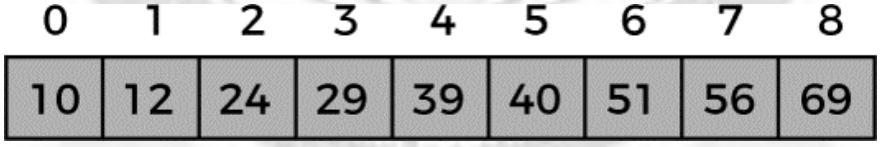
Working of Binary search

- To understand the working of the Binary search algorithm, let's take a sorted array. It will be easy to understand the working of Binary search with an example.
- There are two methods to implement the binary search algorithm -
  - Iterative method
  - Recursive method

The recursive method of binary search follows the divide and conquer approach

Consider an array of elements 10, 12, 24, 29, 39, 40, 51, 56, 69

Let the elements of array are

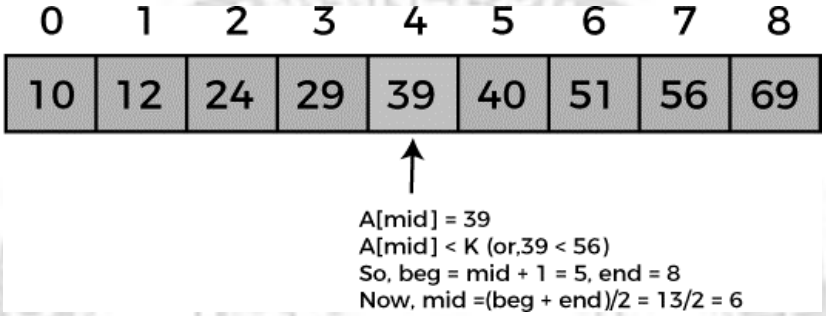


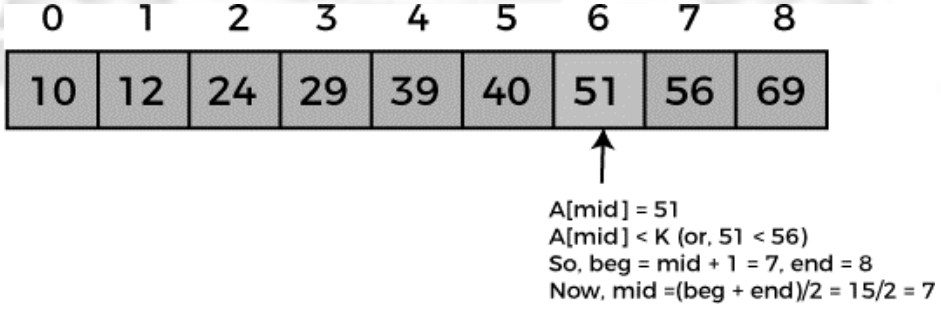
Let the element to search is, **K = 56**

Use the below formula to calculate the mid of the array

mid = (beg + end)/2

In the given array beg = 0, end = 8. So mid = (0+8)/2 = 4


0 1 2 3 4 5 6 7 8
10 12 24 29 39 40 51 56 69
A[mid] = 39
A[mid] < K (or,39 < 56)
So, beg = mid + 1 = 5, end = 8
Now, mid =(beg + end)/2 = 13/2 = 6


0 1 2 3 4 5 6 7 8
10 12 24 29 39 40 51 56 69
A[mid] = 51
A[mid] < K (or, 51 < 56)
So, beg = mid + 1 = 7, end = 8
Now, mid =(beg + end)/2 = 15/2 = 7

plexity:

Time Complexity	
Best	O(1)
Worst	O(logn)
Average	O(logn)
Space Complexity	O(1)

N

ow, the eleme nt to search is found. So algorit hm will return the index of the eleme nt match ed.

Binar y S ea rc h co m

Advantages and Disadvantages

Sl. No.	Advantages	Disadvantages
1.	It is a much faster algorithm	It can be used only when data is sorted
2.	It works on the divide and conquers principle	It is more complicated
3.	It is efficient	If random access is not supported ther efficiency might be lost
4.	It is a simple algorithm to understand	It can be implemented only for two-wa transversal data structures

Example Program 5.6: Program for implementation of Binary Search#include <stdio.h>

```
int binarySearch(int a[], int beg, int end, int val)

{
   int mid;
   if(end >= beg)
   {      mid = (beg + end)/2;
/* if the item to be searched is present
```

```
a           )
t           {
                return mid+1;
m           }
i               /* if the item to be searched is smaller than middle, then it can
                only be in
d
d       left subarray*/
l           else if(a[mid] < val)
e           {
                return binarySearch(a, mid+1, end, val);
*           }
/               /* if the item to be searched is greater than middle, then it can
                only be in
        right
i           s
f           u
(           b
a           a
[           r
m           r
i           a
d           y
]           *
            /
=
=           e
            l
v           s
a           e
l           {
```

```
    }
    return -1;
}
int main() {
  int a[] = {21, 14, 35, 30, 40, 51, 55, 57, 70}; //
  given arrayint val = 40; // value to be searched
  int n = sizeof(a) / sizeof(a[0]); // size of array
  int res = binarySearch(a, 0, n-1, val); //
  Store resultprintf("The elements of the
  array are - ");
for (int i = 0; i < n; i++)
printf("%d ", a[i]);
printf("\nElement to be searched is - %d", val);
if (res == -1)
printf("\nElement is not present in the array");
else
printf("\nElement is present at %d position of array", res);
return 0;
}
```

Output

```
The elements of the array are - 21, 14, 35, 30, 40, 51, 55, 57, 70
Element to be searched is – 40
Element is present at 5 position of array
```

Linear Search vs Binary Search

Sl. No.	Linear Search	Binary Search
1.	In linear search input data need not to be in sorted.	In binary search input data need to be in sorted order.
2.	It is also called sequential search.	It is also called half-interval search.
3.	It is preferable for the small-sized data sets.	It is preferable for the large-size data sets.
4.	The time complexity of linear search O(n).	The time complexity of binary search O(log n).
5.	Multidimensional array can be used.	Only single dimensional array is used.
6.	Linear search performs equality comparisons	Binary search performs ordering comparisons
7.	It is less complex.	It is more complex.
8.	It is very slow process.	It is very fast process