Linear Search Algorithm

In this article, we will discuss the Linear Search Algorithm. Searching is the process of finding some particular element in the list. If the element is present in the list, then the process is called successful, and the process returns the location of that element; otherwise, the search is called unsuccessful.

Two popular search methods are Linear Search and Binary Search. So, here we will discuss the popular searching technique, i.e., Linear Search Algorithm.

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).**

The steps used in the implementation of Linear Search are listed as follows -

- First, we have to traverse the array elements using a **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.**

Now, let's see the algorithm of linear search.

Algorithm

1. Linear_Search(a, n, val) // 'a' is the given array, 'n' is the size of given array, 'val' is the value to search
2. Step 1: set pos = -1
3. Step 2: set i = 1
4. Step 3: repeat step 4 while i <= n
5. Step 4: if a[i] == val
6. set pos = i

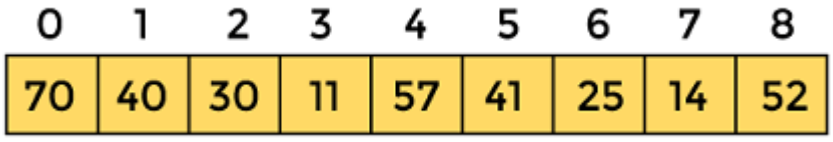
7. print pos
8. go to step 6
9. [end of if]
10. set ii = i + 1
11. [end of loop]
12. Step 5: if pos = -1
13. print "value is not present in the array "
14. [end of if]
15. Step 6: exit

Working of Linear search

Now, let's see the working of the linear search Algorithm.

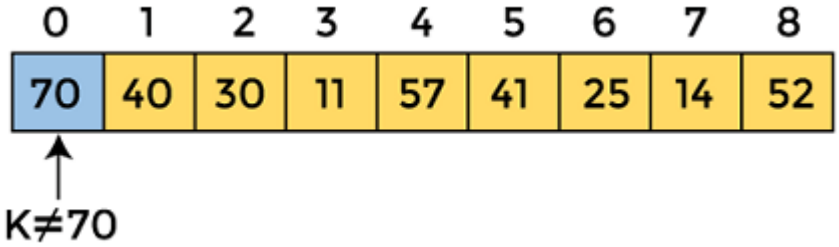
To understand the working of linear search algorithm, let's take an unsorted array. It will be easy to understand the working of linear search with an example.

Let the elements of array are -

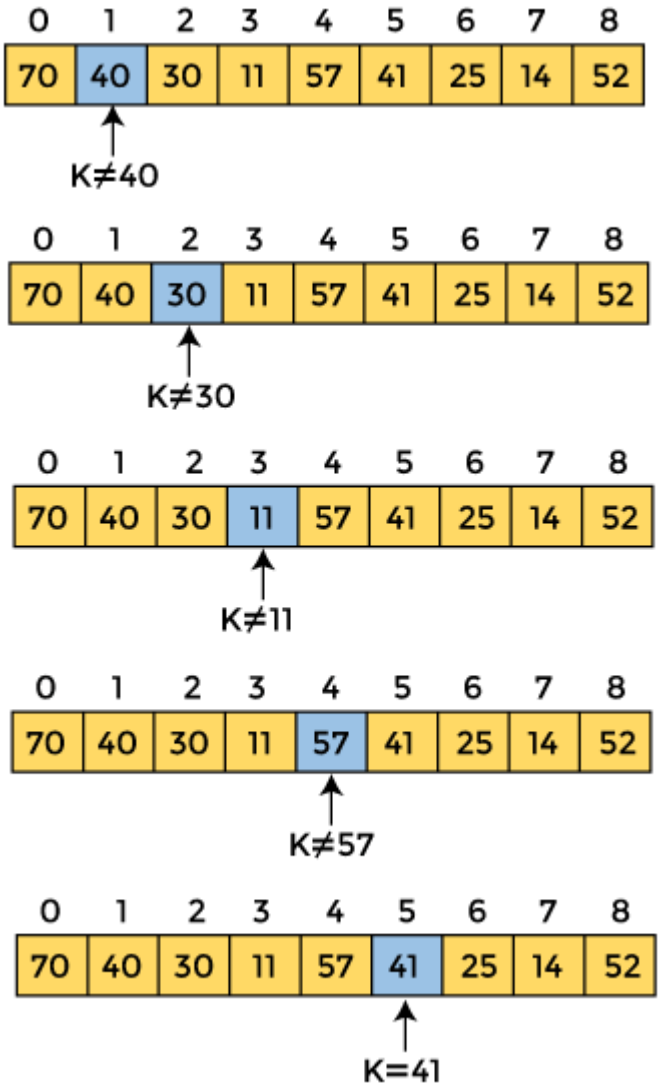


Let the element to be searched is **K = 41**

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

Now, let's see the time complexity of linear search in the best case, average case, and worst case. We will also see the space complexity of linear search.

1. Time Complexity

Case	Time Complexity
Best Case	O(1)
Average Case	O(n)

Worst Case	O(n)

- **Best Case Complexity -** In Linear search, best case occurs when the element we are finding is at the first position of the array. The best-case time complexity of linear search is **O(1).**
- **Average Case Complexity -** The average case time complexity of linear search is **O(n).**
- **Worst Case Complexity -** In Linear search, the worst case occurs when the element we are looking is present at the end of the array. The worst-case in linear search could be when the target element is not present in the given array, and we have to traverse the entire array. The worst-case time complexity of linear search is **O(n).**

The time complexity of linear search is **O(n)** because every element in the array is compared only once.

2. Space Complexity

Space Complexity	O(1)

- The space complexity of linear search is O(1).

Implementation of Linear Search

Now, let's see the programs of linear search in different programming languages.

Program: Write a program to implement linear search in C language.

```
#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() {
```

```
12.  int a[] = {70, 40, 30, 11, 57, 41, 25, 14, 52}; // given array
13.  int val = 41; // value to be searched
14.  int n = sizeof(a) / sizeof(a[0]); // size of array
15.  int res = linearSearch(a, n, val); // Store result
16.  printf("The elements of the array are - ");
17.  for (int i = 0; i < n; i++)
18.  printf("%d ", a[i]);
19.  printf("\nElement to be searched is - %d", val);
20.  if (res == -1)
21.  printf("\nElement is not present in the array");
22.  else
23.  printf("\nElement is present at %d position of array", res);
24.  return 0;
25. }
```

Output

```
The elements of the array are - 70 40 30 11 57 41 25 14 52
Element to be searched is - 41
Element is present at 6 position of array
```

Program: Write a program to implement linear search in C++.

```
1.  #include <iostream>
2.  using namespace std;
3.  int linearSearch(int a[], int n, int val) {
4.    // Going through array linearly
5.    for (int i = 0; i < n; i++)
6.      {
7.          if (a[i] == val)
8.          return i+1;
9.      }
10.   return -1;
11. }
12. int main() {
13.   int a[] = {69, 39, 29, 10, 56, 40, 24, 13, 51}; // given array
14.   int val = 56; // value to be searched
15.   int n = sizeof(a) / sizeof(a[0]); // size of array
16.   int res = linearSearch(a, n, val); // Store result
```

```
17.  cout<<"The elements of the array are - ";
18.  for (int i = 0; i < n; i++)
19.  cout<<a[i]<<" ";
20.  cout<<"\nElement to be searched is - "<<val;
21.  if (res == -1)
22.  cout<<"\nElement is not present in the array";
23.  else
24.  cout<<"\nElement is present at "<<res<<" position of array";
25.  return 0;
26. }
```

Output

```
The elements of the array are - 69 39 29 10 56 40 24 13 51
Element to be searched is - 56
Element is present at 5 position of array
```

Program: Write a program to implement linear search in C#.

```
1.  using System;
2.  class LinearSearch {
3.  static int linearSearch(int[] a, int n, int val) {
4.    // Going through array sequencially
5.    for (int i = 0; i < n; i++)
6.      {
7.          if (a[i] == val)
8.          return i+1;
9.      }
10.   return -1;
11. }
12. static void Main() {
13.   int[] a = {56, 30, 20, 41, 67, 31, 22, 14, 52}; // given array
14.   int val = 14; // value to be searched
15.   int n = a.Length; // size of array
16.   int res = linearSearch(a, n, val); // Store result
17.   Console.Write("The elements of the array are - ");
18.   for (int i = 0; i < n; i++)
19.   Console.Write(" " + a[i]);
20.   Console.WriteLine();
```

```
21. Console.WriteLine("Element to be searched is - " + val);
22. if (res == -1)
23. Console.WriteLine("Element is not present in the array");
24. else
25. Console.Write("Element is present at " + res +" position of array");
26. }
27. }
```

Output

```
The elements of the array are -  56 30 20 41 67 31 22 14 52
Element to be searched is - 14
Element is present at 8 position of array
```

Program: Write a program to implement linear search in Java.

```
1. class LinearSearch {
2. static int linearSearch(int a[], int n, int val) {
3.   // Going through array sequencially
4.   for (int i = 0; i < n; i++)
5.    {
6.       if (a[i] == val)
7.       return i+1;
8.    }
9.   return -1;
10. }
11. public static void main(String args[]) {
12.   int a[] = {55, 29, 10, 40, 57, 41, 20, 24, 45}; // given array
13.   int val = 10; // value to be searched
14.   int n = a.length; // size of array
15.   int res = linearSearch(a, n, val); // Store result
16.   System.out.println();
17.   System.out.print("The elements of the array are - ");
18.   for (int i = 0; i < n; i++)
19.   System.out.print(" " + a[i]);
20.   System.out.println();
21.   System.out.println("Element to be searched is - " + val);
22.   if (res == -1)
23.   System.out.println("Element is not present in the array");
```

```
24.  else
25.  System.out.println("Element is present at " + res +" position of array");
26. }
27. }
```

Output

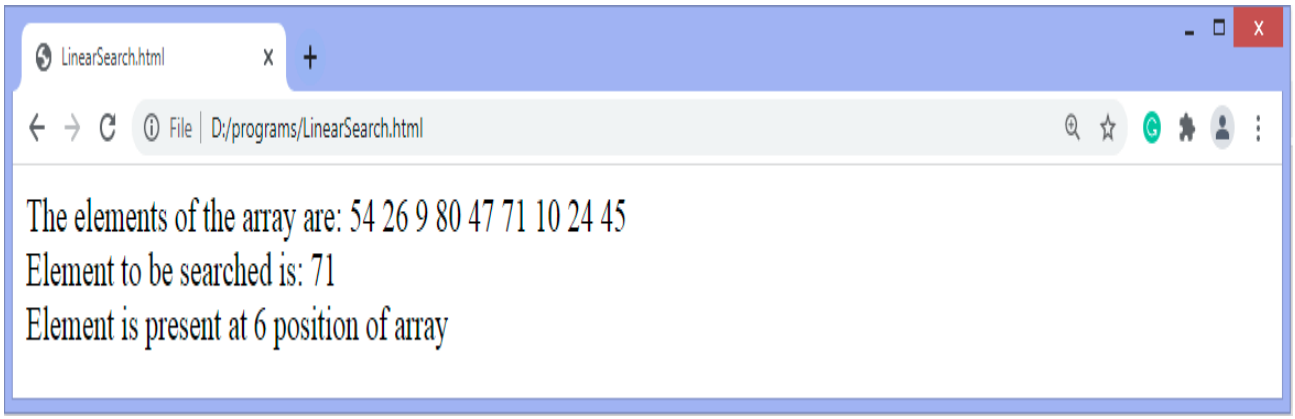
```
D:\JTP>javac LinearSearch.java

D:\JTP>java  LinearSearch

The elements of the array are -  55 29 10 40 57 41 20 24 45
Element to be searched is - 10
Element is present at 3 position of array

D:\JTP>_
```

Program: Write a program to implement linear search in JavaScript.

```
1.  <html>
2.  <head>
3.  </head>
4.  <body>
5.  <script>
6.     var a = [54, 26, 9, 80, 47, 71, 10, 24, 45]; // given array
7.     var val = 71; // value to be searched
8.     var n = a.length; // size of array
9.     function linearSearch(a, n, val) {
10.    // Going through array sequencially
11.    for (var i = 0; i < n; i++)
12.    {
13.       if (a[i] == val)
14.       return i+1;
15.    }
16.    return -1
17.    }
18.   var res = linearSearch(a, n, val); // Store result
19.   document.write("The elements of the array are: ");
20.   for (i = 0; i < n; i++)
21.   document.write(" " + a[i]);
22.   document.write("<br>" + "Element to be searched is: " + val);
23.   if (res == -1)
```

```
24.  document.write("<br>" + "Element is not present in the array");
25.  else
26.  document.write("<br>" + "Element is present at " + res +" position of array");
27.   </script>
28.   </body>
29. </html>
```

Output

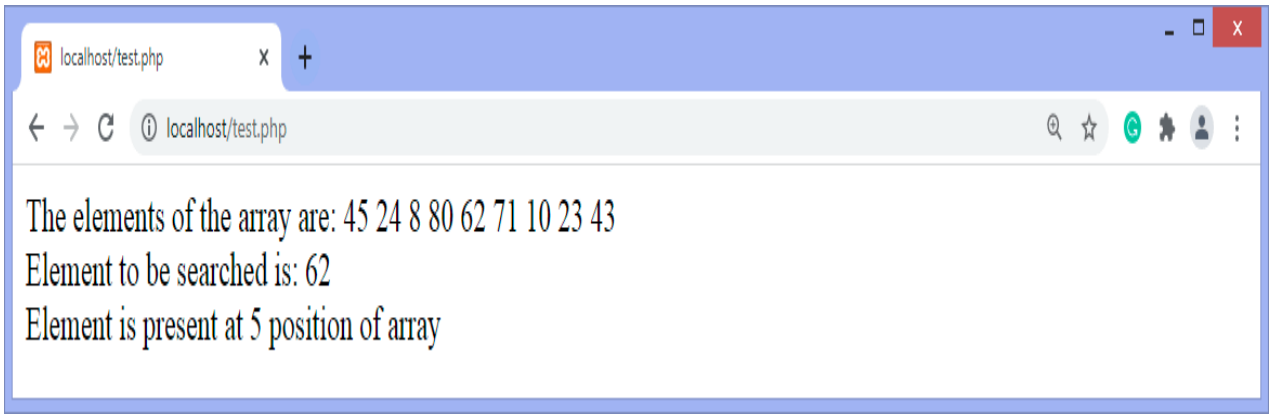


Program: Write a program to implement linear search in PHP.

```
1.  <?php
2.     $a = array(45, 24, 8, 80, 62, 71, 10, 23, 43); // given array
3.     $val = 62; // value to be searched
4.     $n = sizeof($a); //size of array
5.     function linearSearch($a, $n, $val) {
6.     // Going through array sequencially
7.     for ($i = 0; $i < $n; $i++)
8.     {
9.        if ($a[$i] == $val)
10.       return $i+1;
11.    }
12.    return -1;
13.    }
14.   $res = linearSearch($a, $n, $val); // Store result
15.   echo "The elements of the array are: ";
16.   for ($i = 0; $i < $n; $i++)
17.   echo " " , $a[$i];
18.   echo "<br>" , "Element to be searched is: " , $val;
```

```
19.  if ($res == -1)
20.  echo "<br>" , "Element is not present in the array";
21.  else
22.  echo "<br>" , "Element is present at " , $res , " position of array";
23. ?>
```

Output



So, that's all about the article. Hope the article will be helpful and informative to you.

Binary Search Algorithm

In this article, we will discuss the Binary Search Algorithm. Searching is the process of finding some particular element in the list. If the element is present in the list, then the process is called successful, and the process returns the location of that element. Otherwise, the search is called unsuccessful.

Linear Search and Binary Search are the two popular searching techniques. Here we will discuss the Binary Search Algorithm.

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.

> ***NOTE: Binary search can be implemented on sorted array elements. If the list elements are not arranged in a sorted manner, we have first to sort them.***

Now, let's see the algorithm of Binary Search.

Algorithm

1. 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
2. Step 1: set beg = lower_bound, end = upper_bound, pos = - 1
3. Step 2: repeat steps 3 and 4 while beg <=end
4. Step 3: set mid = (beg + end)/2
5. Step 4: if a[mid] = val
6. set pos = mid
7. print pos
8. go to step 6
9. else if a[mid] > val
10. set end = mid - 1
11. else
12. set beg = mid + 1
13. [end of if]

14. [end of loop]
15. Step 5: if pos = -1
16. print "value is not present in the array"
17. [end of if]
18. Step 6: exit

Working of Binary search

Now, let's see the working of the Binary Search Algorithm.

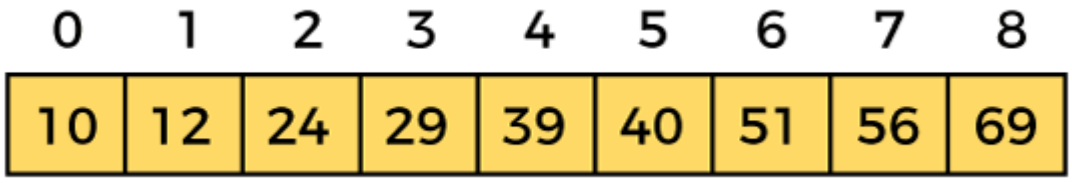
To understand the working of the Binary search algorithm, let's take a sorted array. It will be easy to understand the working of Binary search with an example.

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.

Let the elements of array are -



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

We have to use the below formula to calculate the **mid** of the array -

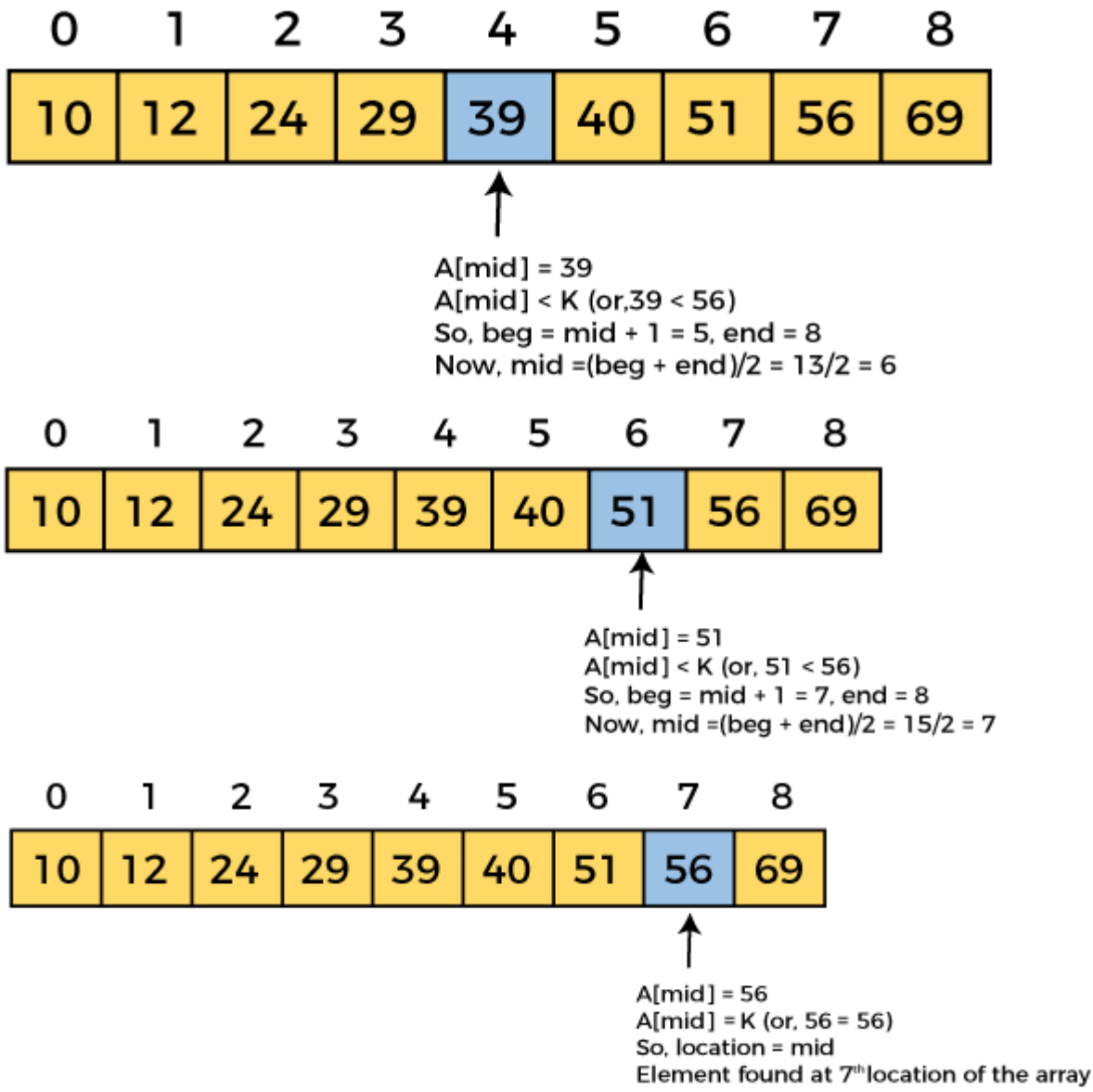
1. mid = (beg + end)/2

So, in the given array -

beg = 0

end = 8

mid = (0 + 8)/2 = 4. So, 4 is the mid of the array.



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

Binary Search complexity

Now, let's see the time complexity of Binary search in the best case, average case, and worst case. We will also see the space complexity of Binary search.

1. Time Complexity

Case	Time Complexity

Best Case	O(1)
Average Case	O(logn)
Worst Case	O(logn)

- **Best Case Complexity -** In Binary search, best case occurs when the element to search is found in first comparison, i.e., when the first middle element itself is the element to be searched. The best-case time complexity of Binary search is **O(1).**
- **Average Case Complexity -** The average case time complexity of Binary search is **O(logn).**
- **Worst Case Complexity -** In Binary search, the worst case occurs, when we have to keep reducing the search space till it has only one element. The worst-case time complexity of Binary search is **O(logn).**

2. Space Complexity

Space Complexity	O(1)

- The space complexity of binary search is O(1).

Implementation of Binary Search

Now, let's see the programs of Binary search in different programming languages.

Program: Write a program to implement Binary search in C language.

```
#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 at middle */
      if(a[mid] == val)
      {
         return mid+1;
      }
```

```
12.          /* if the item to be searched is smaller than middle, then it can only be in left subarra
    y */
13.       else if(a[mid] < val)
14.       {
15.           return binarySearch(a, mid+1, end, val);
16.       }
17.           /* if the item to be searched is greater than middle, then it can only be in righ
    t subarray */
18.       else
19.       {
20.          return binarySearch(a, beg, mid-1, val);
21.       }
22.    }
23.     return -1;
24. }
25. int main() {
26.   int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70}; // given array
27.   int val = 40; // value to be searched
28.   int n = sizeof(a) / sizeof(a[0]); // size of array
29.   int res = binarySearch(a, 0, n-1, val); // Store result
30.   printf("The elements of the array are - ");
31.   for (int i = 0; i < n; i++)
32.   printf("%d ", a[i]);
33.   printf("\nElement to be searched is - %d", val);
34.   if (res == -1)
35.   printf("\nElement is not present in the array");
36.   else
37.   printf("\nElement is present at %d position of array", res);
38.   return 0;
39. }
```

Output

```
The elements of the array are - 11 14 25 30 40 41 52 57 70
Element to be searched is - 40
Element is present at 5 position of array
```

Program: Write a program to implement Binary search in C++.

```
#include <iostream>
using namespace std;
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 at middle */
      if(a[mid] == val)
      {
         return mid+1;
      }
         /* if the item to be searched is smaller than middle, then it can only be in left subarra
y */
      else if(a[mid] < val)
      {
          return binarySearch(a, mid+1, end, val);
      }
       /* if the item to be searched is greater than middle, then it can only be in right s
ubarray */
   else
      {
         return binarySearch(a, beg, mid-1, val);
      }
   }
    return -1;
}
int main() {
  int a[] = {10, 12, 24, 29, 39, 40, 51, 56, 70}; // given array
  int val = 51; // value to be searched
  int n = sizeof(a) / sizeof(a[0]); // size of array
  int res = binarySearch(a, 0, n-1, val); // Store result
  cout<<"The elements of the array are - ";
  for (int i = 0; i < n; i++)
  cout<<a[i]<<" ";
  cout<<"\nElement to be searched is - "<<val;
```

```
36.  if (res == -1)
37.  cout<<"\nElement is not present in the array";
38.  else
39.  cout<<"\nElement is present at "<<res<<" position of array";
40.  return 0;
41. }
```

Output

```
The elements of the array are - 10 12 24 29 39 40 51 56 70
Element to be searched is - 51
Element is present at 7 position of array
```

Program: Write a program to implement Binary search in C#.

```
1.  using System;
2.  class BinarySearch {
3.     static int binarySearch(int[] a, int beg, int end, int val)
4.     {
5.         int mid;
6.         if(end >= beg)
7.         {
8.             mid = (beg + end)/2;
9.             if(a[mid] == val)
10.            {
11.                return mid+1;  /* if the item to be searched is present at middle */
12.            }
13.            /* if the item to be searched is smaller than middle, then it can only be in left
    subarray */
14.            else if(a[mid] < val)
15.            {
16.               return binarySearch(a, mid+1, end, val);
17.            }
18.            /* if the item to be searched is greater than middle, then it can only be in right subarr
    ay */
19.            else
20.            {
21.                return binarySearch(a, beg, mid-1, val);
22.            }
```

```
        }
        return -1;
    }
    static void Main() {
        int[] a = {9, 11, 23, 28, 38, 45, 50, 56, 70}; // given array
        int val = 70; // value to be searched
        int n = a.Length; // size of array
        int res = binarySearch(a, 0, n-1, val); // Store result
        Console.Write("The elements of the array are - ");
        for (int i = 0; i < n; i++)
        {
            Console.Write(a[i] + " ");
        }
        Console.WriteLine();
        Console.WriteLine("Element to be searched is - " + val);
        if (res == -1)
        Console.WriteLine("Element is not present in the array");
        else
        Console.WriteLine("Element is present at " + res + " position of array");
    }
    }
```

Output

```
The elements of the array are - 9 11 23 28 38 45 50 56 70
Element to be searched is - 70
Element is present at 9 position of array
```

Program: Write a program to implement Binary search in Java.

```
class BinarySearch {
    static int binarySearch(int a[], int beg, int end, int val)
    {
        int mid;
        if(end >= beg)
        {
            mid = (beg + end)/2;
            if(a[mid] == val)
```

```
9.              {
10.                return mid+1;  /* if the item to be searched is present at middle
11. */
12.             }
13.             /* if the item to be searched is smaller than middle, then it can only
14. be in left subarray */
15.             else if(a[mid] < val)
16.             {
17.                 return binarySearch(a, mid+1, end, val);
18.             }
19.             /* if the item to be searched is greater than middle, then it can only be
20. in right subarray */
21.             else
22.             {
23.                 return binarySearch(a, beg, mid-1, val);
24.             }
25.         }
26.         return -1;
27.     }
28.     public static void main(String args[]) {
29.         int a[] = {8, 10, 22, 27, 37, 44, 49, 55, 69}; // given array
30.         int val = 37; // value to be searched
31.         int n = a.length; // size of array
32.         int res = binarySearch(a, 0, n-1, val); // Store result
33.         System.out.print("The elements of the array are: ");
34.         for (int i = 0; i < n; i++)
35.         {
36.             System.out.print(a[i] + " ");
37.         }
38.         System.out.println();
39.         System.out.println("Element to be searched is: " + val);
40.         if (res == -1)
41.         System.out.println("Element is not present in the array");
42.         else
43.         System.out.println("Element is present at " + res + " position of array");
44.     }
45.     }
```

Output

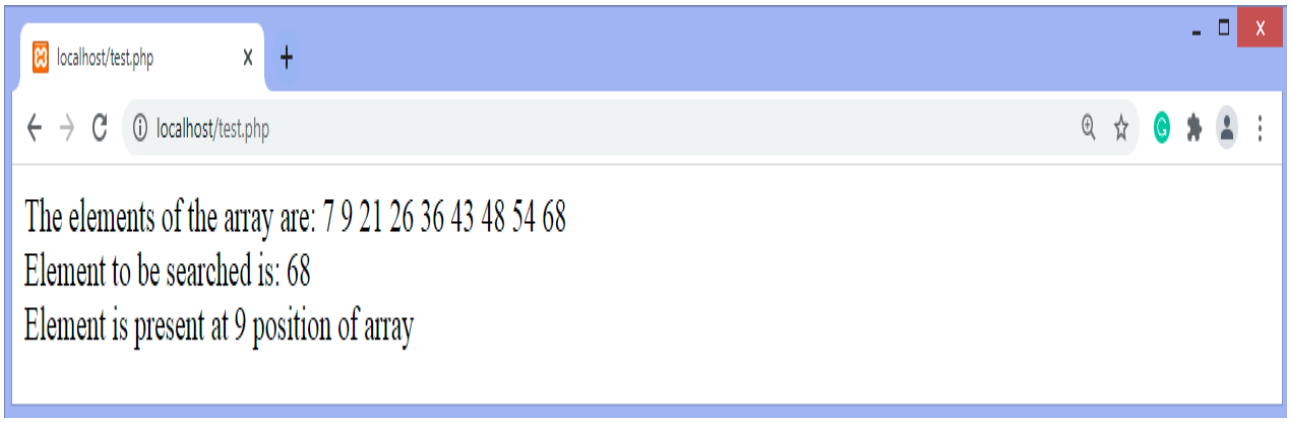
```
D:\JTP>javac BinarySearch.java

D:\JTP>java  BinarySearch
The elements of the array are: 8 10 22 27 37 44 49 55 69
Element to be searched is: 37
Element is present at 5 position of array

D:\JTP>
```

Program: Write a program to implement Binary search in PHP.

```
<?php
function binarySearch($a, $beg, $end, $val)
{
if($end >= $beg)
{
    $mid = floor(($beg + $end)/2);
    if($a[$mid] == $val)
    {
        return $mid+1;  /* if the item to be searched is present at middle */
    }
    /* if the item to be searched is smaller than middle, then it can only  be in left subarray */
    else if($a[$mid] < $val)
    {
       return binarySearch($a, $mid+1, $end, $val);
    }
    /* if the item to be searched is greater than middle, then it can only be in right subarray */

    else
    {
        return binarySearch($a, $beg, $mid-1, $val);
    }
}
return -1;
}
    $a = array(7, 9, 21, 26, 36, 43, 48, 54, 68); // given array
    $val = 68; // value to be searched
    $n = sizeof($a); // size of array
```

```
27.    $res = binarySearch($a, 0, $n-1, $val); // Store result
28.    echo "The elements of the array are: ";
29.    for ($i = 0; $i < $n; $i++)
30.       echo " " , $a[$i];
31.    echo "<br>" , "Element to be searched is: " , $val;
32.    if ($res == -1)
33.       echo "<br>" , "Element is not present in the array";
34.    else
35.       echo "<br>" , "Element is present at " , $res , " position of array";
36. ?>
```

Output



So, that's all about the article. Hope the article will be helpful and informative to you.