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.

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]
```

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 -

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

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

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

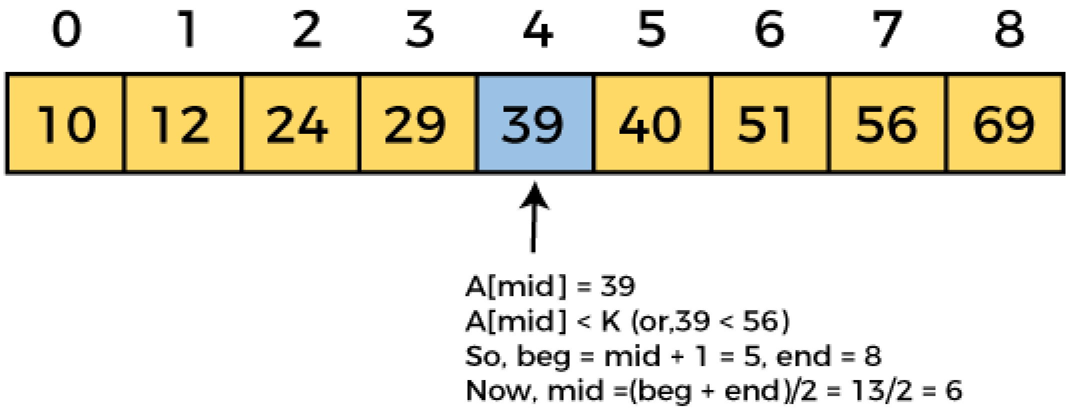
1. mid = (beg + end)/2

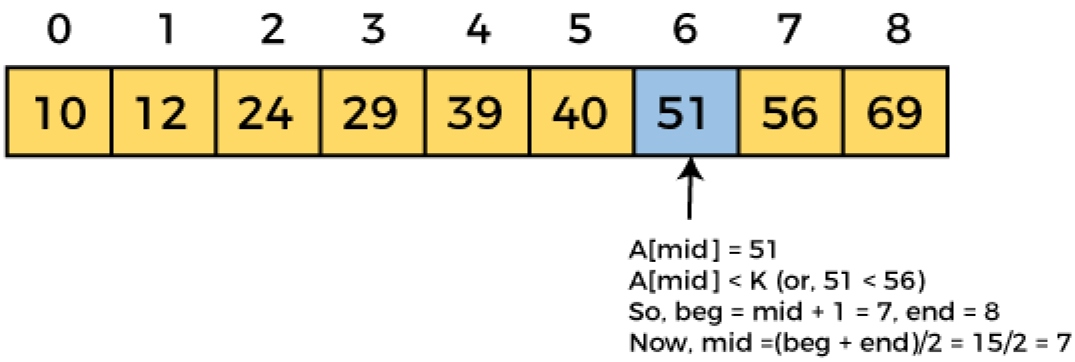
So, in the given array -

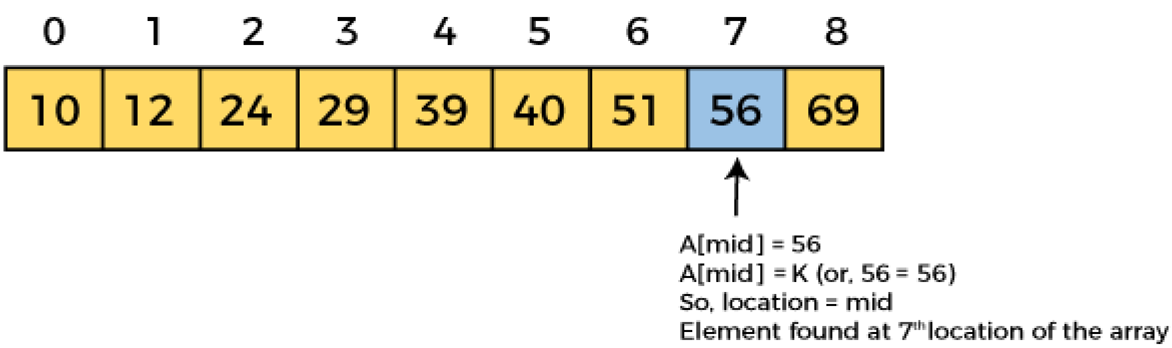
beg = 0

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.

In binary search, we know that the *search space is reduced by half* at each step and this guides us in computing the time complexity.

For an array with `n` elements, we check if the middle-most element matches the `target`. If so, we return `True` and terminate the search.
But if the middle element does not match the `target`, we perform binary search on a subarray of size at most `n/2`. In the next step, we have to search through an array of size at most `n/4`. And we continue this recursively until we can make a decision in a constant time (when the subarray is empty).
At step `k`, we need to search through an array of size at most `n/(2^k)`. And we need to find the smallest such `k` for which we have no subarray to search through.
Mathematically:

$$Smallest\ k\ for\ which\ \frac{n}{2^k} < 1$$
$$\implies \quad k = \lfloor log_2(n) \rfloor + 1$$

The time complexity of binary search is, therefore, **O(logn)**. This is much more efficient than the linear time O(n), especially for large values of n.
For example, if the array has 1000 elements. 2^(10) = 1024. While the binary search algorithm will terminate in around 10 steps, linear search will take a thousand steps in the worst case.

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).