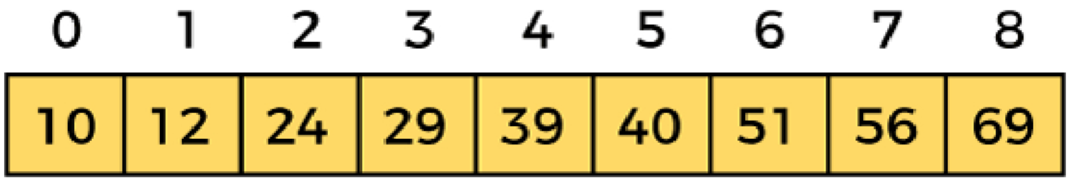
Binary search is a fast search algorithm with run-time complexity of O(log n). This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the sub-array as well until the size of the subarray reduces to zero.

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 -

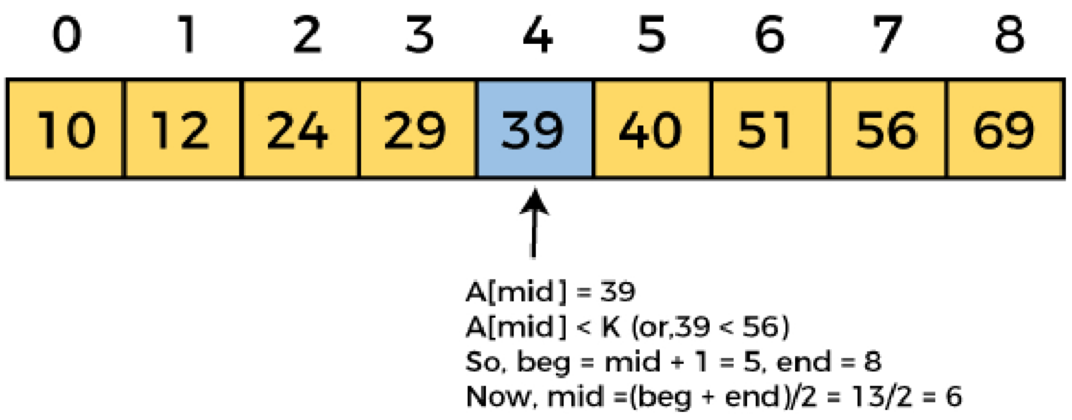
1. mid = (beg + end)/2

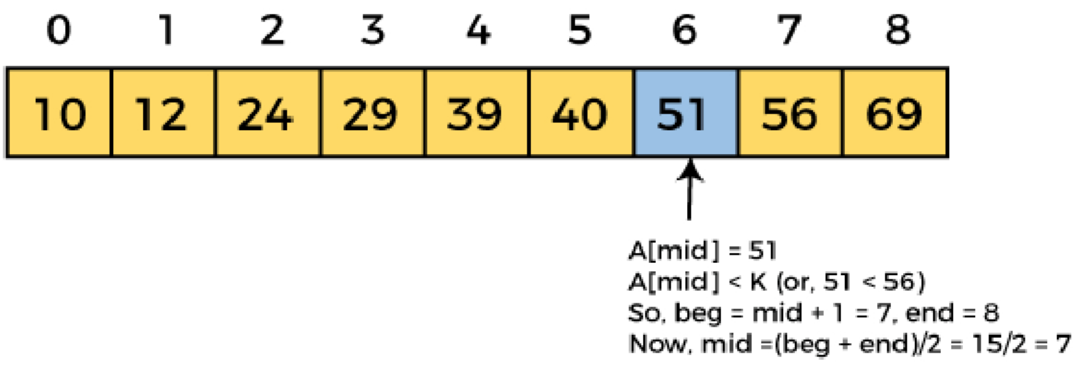
So, in the given array -

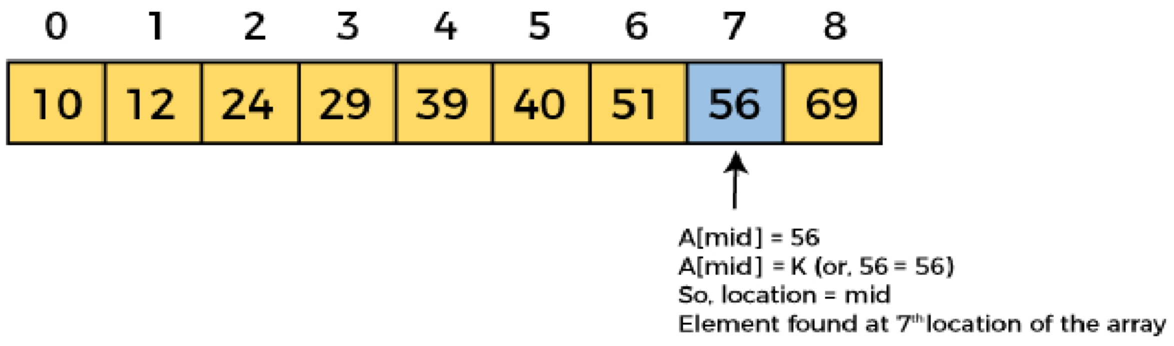
beg = 0

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.

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

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

Algorithm

```
binary Search(arr, x, low, high)
{
      while(low<High)
(
            mid = (low + high)/2
               if (x == arr[mid])
```

```
                 return mid

                else if (x > arr[mid]) // x is on the right side
                    low = mid + 1

                 else                  // x is on the left side
                    High = mid - 1
}
Return -1
}
```