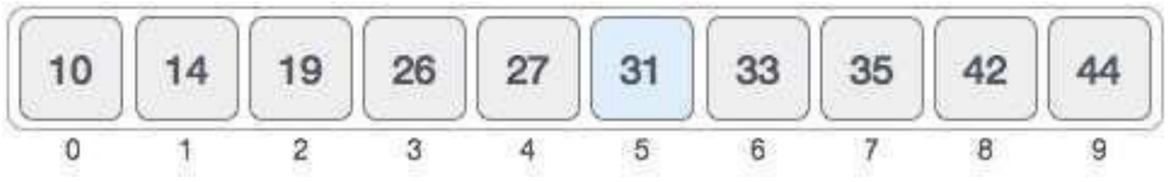
Binary search :

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 right of the middle item. Otherwise, the item is searched for in the sub-array to the left of the middle item. This process continues on the sub-array as well until the size of the subarray reduces to zero.

Binary Search Concept:

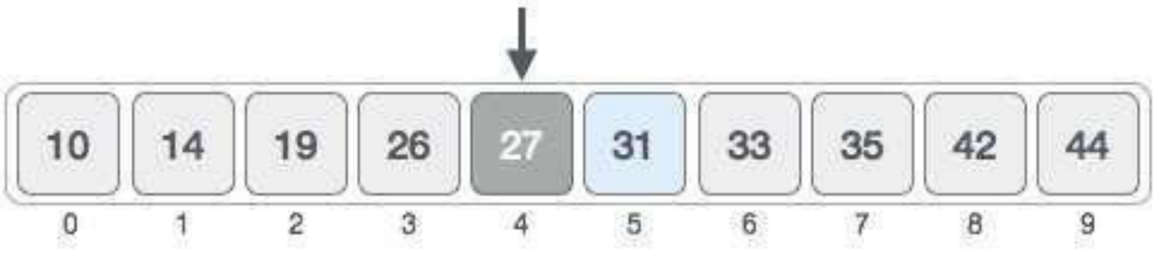
For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.



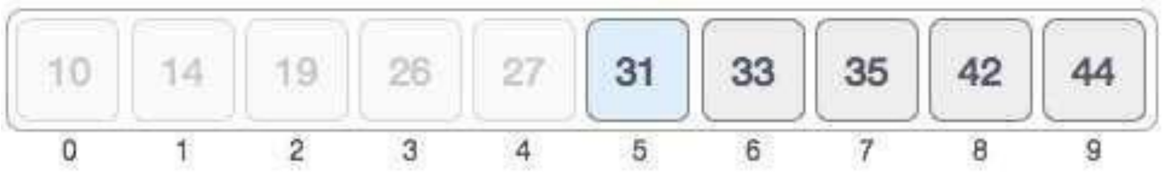
First, we shall determine half of the array by using this formula −

```
mid = low + (high - low) / 2
```

Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5). So, 4 is the mid of the array.



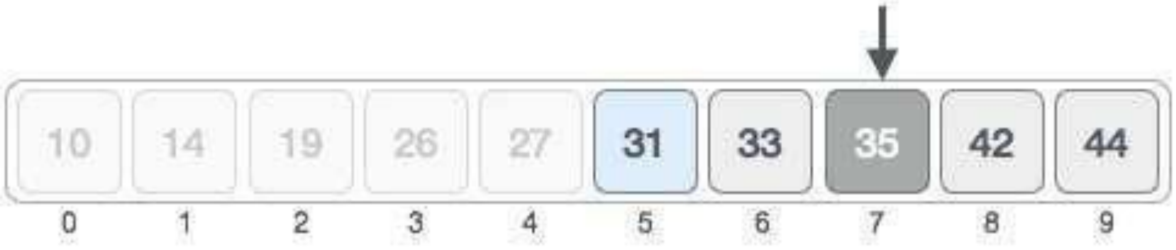
Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.



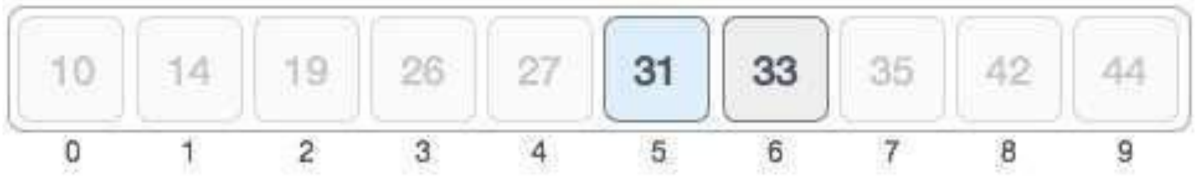
We change our low to mid + 1 and find the new mid value again.

```
low = mid + 1
mid = low + (high - low) / 2
```

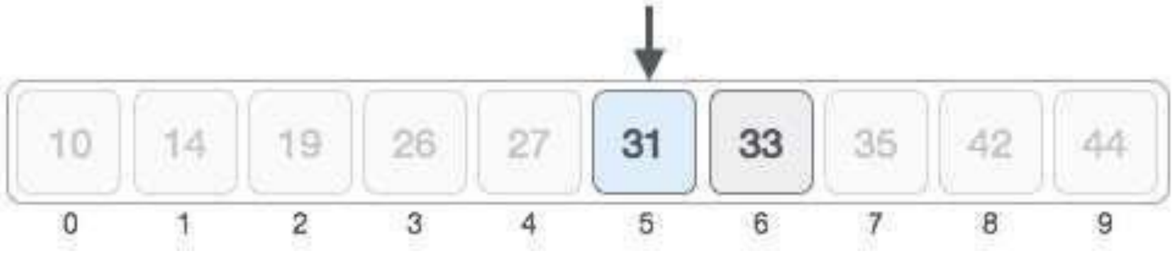
Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.



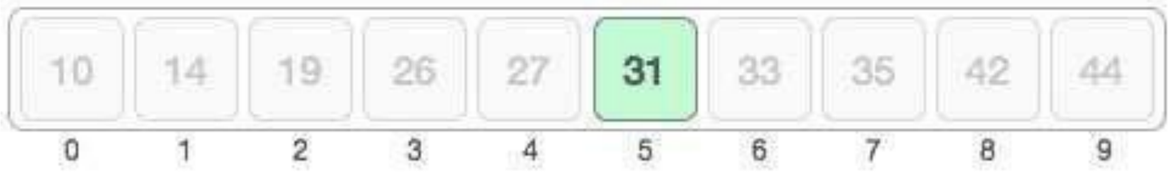
The value stored at location 7 is not a match, rather it is less than what we are looking for. So, the value must be in the lower part from this location.



Hence, we calculate the mid again. This time it is 5.



We compare the value stored at location 5 with our target value. We find that it is a match.



We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

Binary Search implementation in C:

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 a sorted form.

Implementation in C

```
#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;
      }
         /* if the item to be searched is smaller than middle, then it can only b
e 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 b
e in right subarray */
      else
```

```
            {
                return binarySearch(a, beg, mid-1, val);
            }
        }
        return -1;
    }
    int main() {
     int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70}; // given array
     int 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 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;
}
```