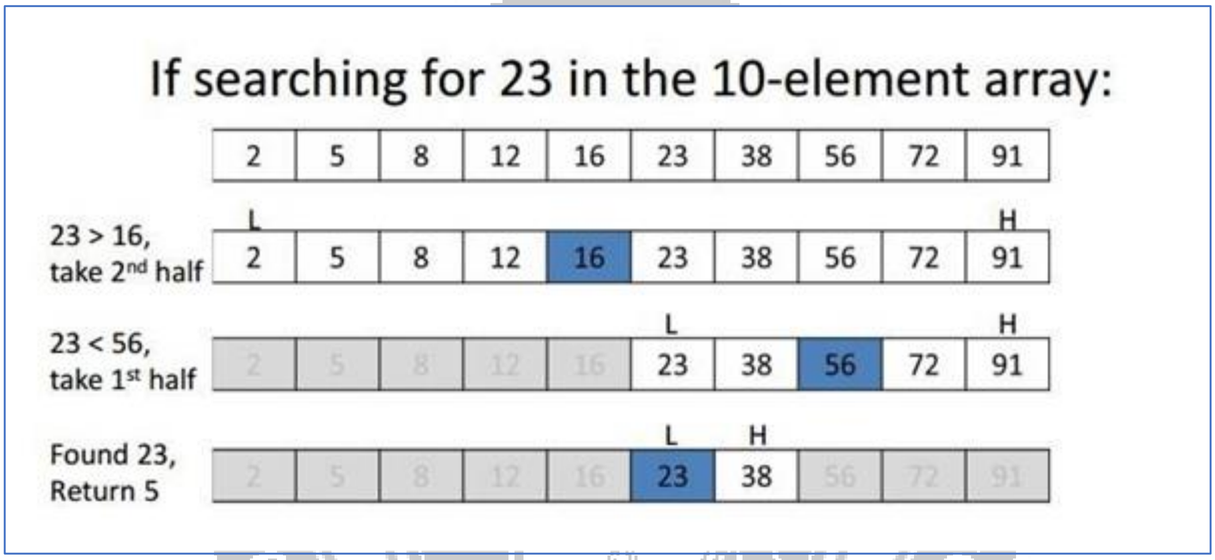
BINARY SEARCH

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Example:



- We basically ignore half of the elements just after one comparison.

Compare x with the middle element.

- If x matches with middle element, we return the mid index.
- Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
- Else (x is smaller) recur for the left half.

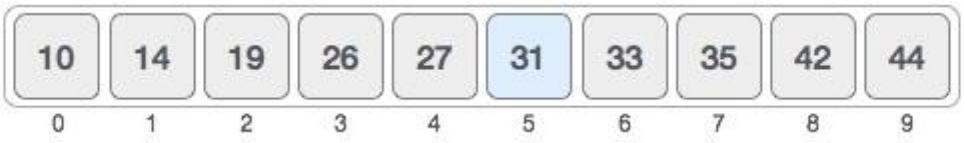
Program to implement recursive Binary Search

```
#include <stdio.h>

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,

// otherwise -1

int binarySearch (int arr[], int l, int r, int x)

{

if (r >= l)

{

int mid = l + (r - l)/2;

// If the element is present at the middle

// itself

if (arr[mid] == x)

return mid;

// If element is smaller than mid, then

// it can only be present in left subarray if (arr[mid] > x)

return binarySearch (arr, l, mid-1, x);

// Else the element can only be present

// in right subarray returnbinarySearch(arr, mid+1, r, x);

}

// We reach here when element is not

// present in array return -1;

}
```

Binary search is a fast search algorithm with run-time complexity of O(log n).

How Binary Search Works?

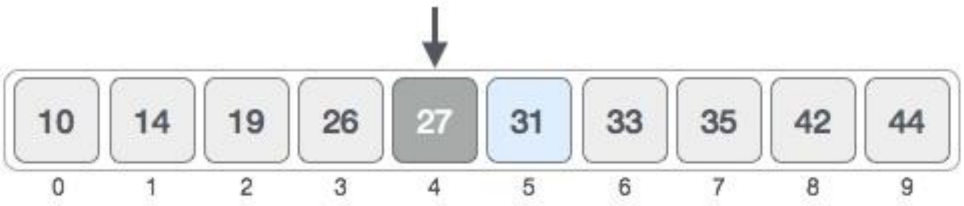
For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.



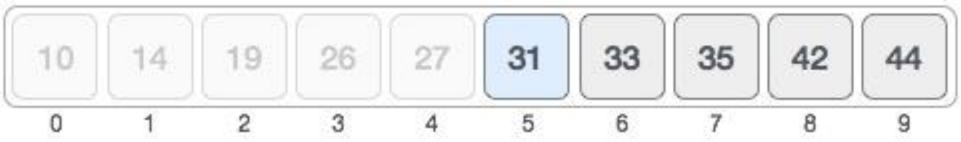
First, we shall determine half of the array by using this formula −

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.

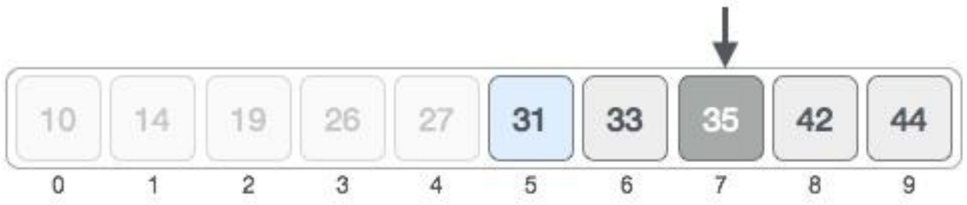


We change our low to mid + 1 and find the new mid value again.

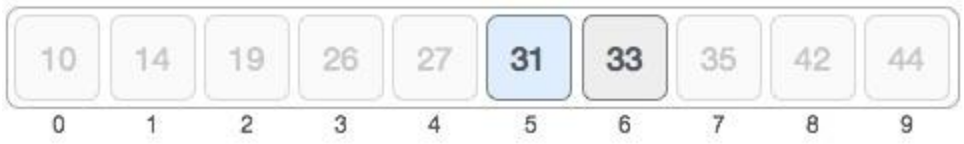
low = mid + 1

mid = low + (high - low) / 2

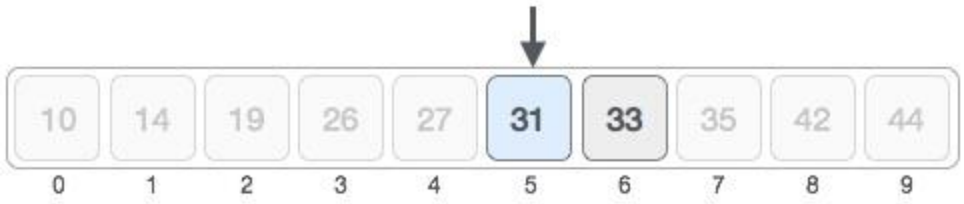
Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.



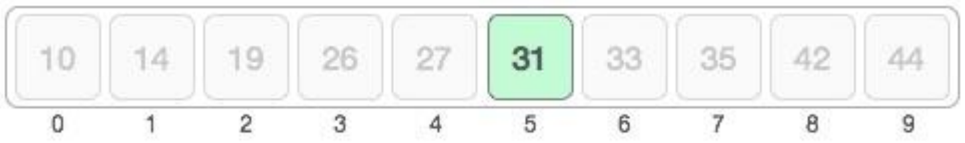
The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location



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.