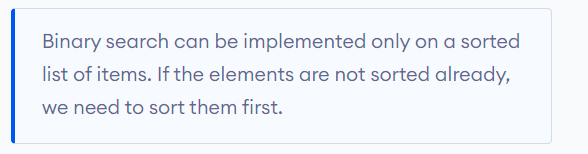
**Binary Search**

Binary Search is a searching algorithm for finding an element's position in a sorted array.

In this approach, the element is always searched in the middle of a portion of an array.



## Binary Search Working

Binary Search Algorithm can be implemented in two ways which are discussed below.

1. Iterative Method
2. Recursive Method

The recursive method follows the divide and conquer approach.

The general steps for both methods are discussed below.

1. The array in which searching is to be performed is:

|  |
| --- |
| initial array Binary Search |
| Initial array |

Let x = 4 be the element to be searched.

1. Set two pointers low and high at the lowest and the highest positions respectively.

|  |
| --- |
| setting pointers Binary Search |
| Setting pointers |

1. Find the middle element mid of the array ie. arr[(low + high)/2] = 6.

|  |
| --- |
| mid element Binary Search |
| Mid element |

1. If x == mid, then return mid.Else, compare the element to be searched with m.
2. If x > mid, compare x with the middle element of the elements on the right side of mid. This is done by setting low to low = mid + 1.
3. Else, compare x with the middle element of the elements on the left side of mid. This is done by setting high to high = mid - 1.

|  |
| --- |
| finding mid element Binary Search |
| Finding mid element |

1. Repeat steps 3 to 6 until low meets high

|  |
| --- |
| mid element Binary Search |
| Mid element |

1. x = 4 is found.

|  |
| --- |
| found Binary Search |
| Found |

## **Binary Search Algorithm**

### Iteration Method

do until the pointers low and high meet each other.

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

### Recursive Method

binarySearch(arr, x, low, high)

if low > high

return False

else

mid = (low + high) / 2

if x == arr[mid]

return mid

else if x > arr[mid] // x is on the right side

return binarySearch(arr, x, mid + 1, high)

else // x is on the right side

return binarySearch(arr, x, low, mid - 1)

## **Binary Search Complexity**

**Time Complexities**

* **Best case complexity**: O(1)
* **Average case complexity**: O(log n)
* **Worst case complexity**: O(log n)

**Space Complexity**

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

## Binary Search Applications

* In libraries of Java, .Net, C++ STL
* While debugging, the binary search is used to pinpoint the place where the error happens.