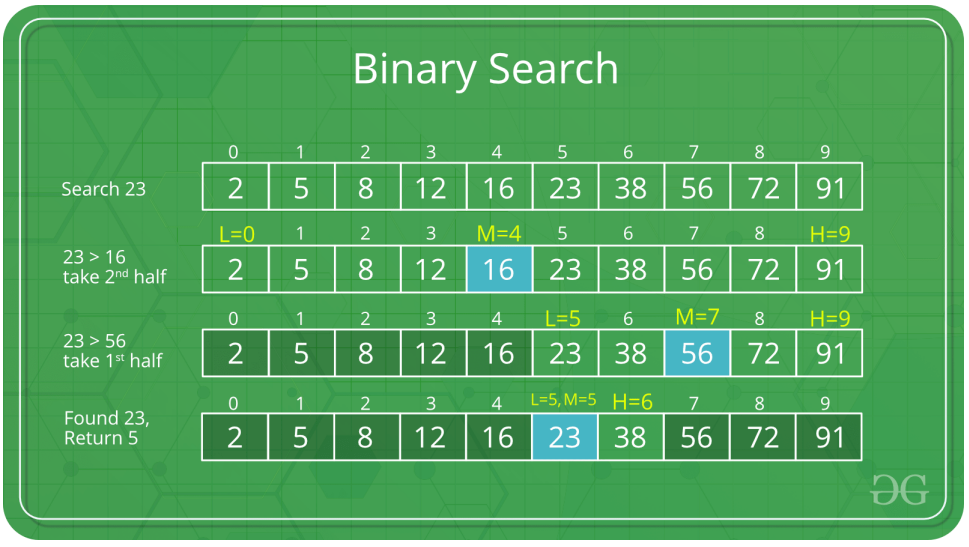
Complexity Analysis of Binary Search

Complexities like **O(1)** and **O(n)** are simple to understand. O(1) means it requires constant time to perform operations like to reach an element in constant time as in case of dictionary and O(n) means, it depends on the value of n to perform operations such as searching an element in an array of n elements.

But for **O(Log n)**, it is not that simple. Let us discuss this with the help of Binary Search Algorithm whose complexity is **O(log n)**.

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:




```
Sorted Array of 10 elements: 2, 5, 8, 12, 16, 23, 38, 56, 7
2, 91
Let us say we want to search for 23.
```

Finding the given element:

Now to find 23, there will be many iterations with each having steps as mentioned in the figure above:

- **Iteration 1:**

```
Array: 2, 5, 8, 12, 16, 23, 38, 56, 72, 91
```

-
    - Select the middle element. (**here 16**)
    - Since 23 is greater than 16, so we divide the array into two halves and consider the sub-array after element 16.
    - Now this subarray with the elements after 16 will be taken into next iteration.
- **Iteration 2:**

```
Array: 23, 38, 56, 72, 91
```

-
    - Select the middle element. (**now 56**)
    - Since 23 is smaller than 56, so we divide the array into two halves and consider the sub-array before element 56.
    - Now this subarray with the elements before 56 will be taken into next iteration.
- **Iteration 3:**

```
Array: 23, 38
```

-
    - Select the middle element. (**now 23**)
    - Since 23 is the middle element. So the iterations will now stop.
    - Let say the iteration in Binary Search terminates after **k** iterations. In the above example, it terminates after 3 iterations, so **here k = 3**
    - At each iteration, the array is divided by half. So let's say the length of array at any iteration is **n**
    - At **Iteration 1**,

```
Length of array =n
```

- At **Iteration 2**,

```
Length of array =n/2
```

- At **Iteration 3**,

```
Length of array =(n/2)/2 =n/22
```

- Therefore, after **Iteration k**,

```
Length of array =n/2k
```

- Also, we know that after

```
After k iterations, thelength of array becomes 1
```

- Therefore

```
Length of array =n/2k=1
=> n = 2k
```

- Applying log function on both sides:

```
=>log2 (n) = log2 (2k)
=>log2 (n) = k log2 (2)
```

-

- As **(loga (a)** = **1)** Therefore,

```
=>k = log2 (n)
```