N -> N/2 -> N/4 -> ... -> N/(2^k)
Here k represents the total number of iterations.
N/2^k = 1 [because we will stop when we have just one element in our search space.]
N = 2^k
Taking logarithm both sides with base 2:
$\log_2 N = k$
Hence we can say that total iterations required will be $\log_2 N$ in the worst case.
Thus, giving us a time complexity of $O(\log_2 N)$.
$O(\log_2 N)$ is a better time complexity than O(n) i.e. the time complexity for linear search. Hence we can see why it is preferred over linear search.

Space Complexity Analysis

In the iterative version of the binary search, we are generating only 3 variables- low, high and mid. Therefore the space complexity of binary search is O(1).

Interview Problem: Lower Bound/First occurrence in an Array

Lower bound is also known as the first occurrence of an element in an array. When the element is present single or in duplicates, the index of its first occurrence is known as Lower Bound.

Q2. Given a sorted array containing duplicates and an integer 'target'. Find the first occurrence of the target in the array. If target does not exist return -1.
Input 1: arr = [1 1 1 2 2 3 4 5 5 5] , target = 5
Output 1: 7
Input 2: arr = [1 2 2 5 5 5 6 8 8 8 8] , target = 7
Output 2: -1

LP_CODE3_BS.java

Output:

```
enter the number of elements you want : 6
enter the elements :
1 2 3 3 4 4
enter the target: 3
The first occurrence of given target is at 2
```

Approach:

- Given that the array is sorted, to search the given element we can use Binary Search Algorithm.
- If we are able to find any occurrence of the target in the array, we reduce our search space to the first half of the array because we want the minimum index that has the value equal to target.
- But there is an equal chance that there may not be any occurrence of the target in the first half, that's why we recorded the current index.
- Because this will be serving as an answer if nothing better works out.

Interview Problem 2: Square root of a number

Q1. Given a positive number, return the square root of it. If the number is not a perfect square, return the floor of its square root.
Floor of a number: floor(4.15) = 4, floor(5.98) = 5

For example,
Input: x = 12
Output: 3
Input: x = 16
Output: 4

LP_CODE4_BS.java

Output :

```
enter the number whose square root you want : 19
The square root of the given number is : 4
```

Approach:

- We know that the square of the number will always be less than it, so we chose the range from 0 to the given number.
- Here you can observe one thing that we do not have any array over which we are planning to apply binary search rather we have a monotonic search region.
- So it is not always mandatory to apply binary search on sorted arrays but any monotonic region of search will work.
- Now we have found the 'mid' element, this can be a probable candidate for square root to we checked if 'mid*mid' equals the given number. If yes, then we found the square root.
- Second possibility could be mid*mid < x, if this is the case that means we need to increase the mid value, so we will move to the right half / latter half of the array. But since there is no guarantee of finding a perfect square root, we need to store the currently obtained mid value as our answer until we get something more precise and accurate than this value.
- The last case would be if mid*mid > x, in that case we need to reduce the value of mid. So we lower the 'high' pointer.
- We will terminate once the value of low will be equal to or greater than high. That indicates we have traversed over all the possibilities.

Next Class Teaser

- Sorting in array
- Bubble Sort
- Insertion Sort
- Selection Sort