Q1. Given an array. Find the number X in the array. If the element is present, return the index of the element, else print "Element not found in array".
Input the size of array, array from user and the element X from user. Use Linear Search to find the element.

```
Enter the number of elements you want to add : 5
Enter the elements of the array: 6
2
3
1
7
Enter the elements to be searched in array2
1
```

Example 2

```
Enter the number of elements you want to add : 5
Enter the elements of the array: 6
2
3
1
7
Enter the elements to be searched in array5
Element not found in array
```

ASS_CODE1_LS.java

Q2. Given an array and an integer "target", return the last occurrence of "target" in the array. If the target is not present return -1.

Input 1: arr = [1 1 1 2 3 4 4 5 6 6 6 6] , target = 4 Output 1: 6
Input 2: arr = [2 2 2 6 6 18 29 30 30 30] , target = 15

Output 2: -1

Solution:

ASS_CODE2_BS.java

Output:

```
Enter the number of elements you want to add : 8
Enter the elements of the array: 1 2 2 6 6 6 8 8
Enter the target : 7
The last occurrence of target is at index : -1
```

```
Enter the number of elements you want to add : 9
Enter the elements of the array: 1 2 2 4 4 4 4 7 8
Enter the target : 4
The last occurrence of target is at index : 6
```

Approach:

- Since the array is sorted, we used the binary search algorithm.
- If the target is found still we need to look in the latter half for the last occurrence but this still can happen that there might not be any more of target in the right section so we will store this answer and shift the low pointer to mid + 1 th location.

- The same step we need to do if the nums[mid] is less than target, therefore we will still look in the latter section of the array.
- But if nums[mid] > target then we need to look in the first half of the array so we did, high = mid - 1.
- We will terminate the loop once the low pointer will be equal to or higher than the high pointer
- At last we will return our answer.

Q3. Given a sorted binary array, efficiently count the total number of 1's in it.

Input 1: arr = [0 0 0 0 1 1 1 1 1 1] Output 1: 6

Input 2: arr = [0 0 0 0 0 1 1]

Output 2: 2

ASS_CODE3_BS.java

Output:

```
Enter the number of elements you want to add : 6
Enter the elements of the array: 0 0 1 1 1 1
The number of one's in the given array is/are: 4
```

Approach:

- Since the array is sorted we are going to efficiently calculate the total number of 1's
- We are going to find the first occurrence of 1.
- Once that is found we will return total elements - first occurrence index.
- This will give us the total number of 1's present because we know here we have only two numbers in the array so once the presence of 1 will begin it will continue till the end of the array.

Steps to find the first occurrence of any target value:

- 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.

Q4. Given a sorted integer array containing duplicates, count occurrences of a given number. If the element is not found in the array, report that as well.

Input: nums[] = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9] target = 5

Output: Target 5 occurs 3 times

Input: nums[] = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9] target = 6

Output: Target 6 occurs 2 times

Solution:
[ASS_CODE4_BS.java]

Output:

```
Enter the number of elements you want to add : 12
Enter the elements of the array: 1 2 2 4 4 4 4 5 6 6 6 8
enter the target: 4
The frequency of target in the given array is 4 time/times
```

Approach:

- In this question we have first calculated the first and last occurrences of the target element respectively.
- These occurrences will basically act as a boundary or window where this target element can stretch maximally.
- Since this is a sorted array we will not find the same target element elsewhere except this window.
- So we did last - first + 1.
- Added an extra 1, since if there is only one element first and last pointers will be on the same index and 0-based indexing will lead to answer as 0 but the answer should be 1.
- Therefore an additional 1 is added. Also if no occurrence of target is then the value of first and last both will be -1 so we directly handled that case separately.

Q5. Given a positive integer num, return true if num is a perfect square or false otherwise.
A perfect square is an integer that is the square of an integer. In other words, it is the product of some integer with itself.

Example 1:

Input: num = 16

Output: true

Explanation: We return true because 4 * 4 = 16 and 4 is an integer.

Example 2:

Input: num = 14

Output: false

Explanation: We return false because 3.742 * 3.742 = 14 and 3.742 is not an integer.

[ASS_CODE5_BS.java]

Output:

Approach:

- We know that the square root of any number cannot be greater than its half.
- For example, the square root of 25 cannot exceed beyond 12, it will surely be some value less than 12 and it is true as well.
- Thus to save some iterations, we started in a range from 1 to num/2.
- For every mid, we check if mid*mid == num, if yes, then the given number is a perfect square.
- If not, that means either mid*mid > num or mid*mid < num.
- If mid*mid > num we will shift in the first half of the array.
- If mid*mid < num we will shift in the second half of the array.
- At last when low>=high, if we have not returned true that means the given number is not a valid square and we will return false.