Count Occurrences in Sorted Array

Problem Statement: You are given a sorted array containing N integers and a number X, you have to find the occurrences of X in the given array.

▼ **Examples**

Example 1:
Input: N = 7, X = 3 , array[] = {2, 2 , 3 , 3 , 3 , 3 , 4}
Output: 4
Explanation: 3 is occurring 4 times in
the given array so it is our answer.

Example 2:
Input: N = 8, X = 2 , array[] = {1, 1, 2, 2, 2, 2, 2, 3}
Output: 5
Explanation: 2 is occurring 5 times in the given array so it is our answer.

Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

▼ Brute Force Approach
▼ Algorithm / Intuition ›

Approach:

The approach is simple. We will linearly search the entire array, and try to increase the counter whenever we get the target value in the array. Using a for loop that runs from 0 to n - 1, containing an if the condition that checks whether the value at that index equals target. If true then increase the counter, at last return the counter.

▼ Code ›

```
def count(arr: [int], n: int, x: int) -> int:
    cnt = 0
    for i in range(n):
        if arr[i] == x:
            cnt += 1
    return cnt

if __name__ == "__main__":
    arr = [2, 4, 6, 8, 8, 8, 11, 13]
    n = 8
    x = 8
    ans = count(arr, n, x)
    print("The number of occurrences is:", ans)
```

Output: The number of occurrences is: 3

▼ Complexity Analysis ›

Time Complexity: O(N), N = size of the given array
Reason: We are traversing the whole array.

Space Complexity: O(1) as we are not using any extra space.

▼ Optimal Approach
▼ Algorithm / Intuition ›

Optimal Approach(Binary Search):

In the previous article, we discussed how to find the first and the last occurrences of a number in a sorted array using Binary Search.

The primary objective of the Binary Search algorithm is to efficiently determine the appropriate half to eliminate, thereby reducing the search space by half. It does this by determining a specific condition that ensures that the target is not present in that half.

Now in order to solve this problem, we are going to use the previous concept. We will find the first and the last occurrences and figure out the number of occurrences like the following:

Total number of occurrences = last occurrence - first occurrence + 1

Algorithm:

- We will get the first and the last occurrences of the number using the function **firstAndLastPosition()**. For the implementation details of the function, please refer to the previous article.
- After getting the indices, we will check the following cases:
    - **If the first index == -1:** This means that the target value is not present in the array. So, we will return 0 as the answer.
    - **Otherwise:** We will find the total number of occurrences like this:
      The total number of occurrences = (last index - first index + 1) and return this length as the answer.

▼ Code ›

```
def firstOccurrence(arr, n, k):
    low = 0
    high = n - 1
    first = -1

    while low <= high:
        mid = (low + high) // 2
        # maybe an answer
        if arr[mid] == k:
            first = mid
            # look for smaller index on the left
            high = mid - 1
        elif arr[mid] < k:
            low = mid + 1  # look on the right
        else:
            high = mid - 1  # look on the left

    return first


def lastOccurrence(arr, n, k):
    low = 0
    high = n - 1
    last = -1

    while low <= high:
        mid = (low + high) // 2
        # maybe an answer
        if arr[mid] == k:
            last = mid
            # look for larger index on the right
            low = mid + 1
        elif arr[mid] < k:
            low = mid + 1  # look on the right
        else:
            high = mid - 1  # look on the left

    return last


def firstAndLastPosition(arr, n, k):
    first = firstOccurrence(arr, n, k)
    if first == -1:
        return (-1, -1)
    last = lastOccurrence(arr, n, k)
    return (first, last)

def count(arr: [int], n: int, x: int) -> int:
    first, last = firstAndLastPosition(arr, n, x)
    if first == -1:
        return 0
    return last - first + 1

if __name__ == "__main__":
    arr = [2, 4, 6, 8, 8, 8, 11, 13]
```

```
n = 8
x = 8
ans = count(arr, n, x)
print("The number of occurrences is:", ans)
```

Output: The number of occurrences is: 3

▼ Complexity Analysis ›

Time Complexity: O(2*logN), where N = size of the given array.
Reason: We are basically using the binary search algorithm twice.

Space Complexity: O(1) as we are using no extra space.