Searching – First & Last Occurrence

1. Introduction

Finding the **first and last occurrence** of an element in an array means determining the **starting and ending positions** of a given target value.

This problem is especially important when the array contains **duplicate elements**.

It is commonly solved using a **modified Binary Search** for better efficiency.

2. What is First & Last Occurrence?

- **First Occurrence:**

  The index where the target element appears **for the first time** in the array.

- **Last Occurrence:**

  The index where the target element appears **for the last time** in the array.

Example:

```
Array:  [2, 4, 4, 4, 6, 8]
Target: 4
```

- First Occurrence → index 1
- Last Occurrence → index 3

3. Why is First & Last Occurrence Important?

This concept is important because:

- Helps count the total number of occurrences
- Used in range queries

- Required in frequency-based problems
- Commonly asked in interviews
- Forms the base for advanced searching problems

4. Prerequisite

⚠️ **Important Condition**

- The array **must be sorted**
- Binary Search is applied to efficiently find occurrences

5. Basic Idea Behind First & Last Occurrence

Instead of stopping when the target is found:

- Continue searching **left side** to find the first occurrence
- Continue searching **right side** to find the last occurrence

This ensures all duplicates are considered.

6. Logic for First Occurrence (Plain English)

1. Start binary search on the array
2. If the middle element equals the target:
    - Store the index
    - Move search to the **left half**
3. If the middle element is greater:
    - Search left half
4. If the middle element is smaller:
    - Search right half
5. Continue until search ends

7. Logic for Last Occurrence (Plain English)

1. Start binary search on the array
2. If the middle element equals the target:
    - Store the index
    - Move search to the **right half**
3. If the middle element is greater:
    - Search left half
4. If the middle element is smaller:
    - Search right half
5. Continue until search ends

8. Visualization Example

```
Array: [1, 2, 4, 4, 4, 5, 6]
Target: 4
```

Binary Search path:

```
First Occurrence → move left after finding 4
Last Occurrence  → move right after finding 4
```

Final result:

```
First Index = 2
Last Index  = 4
```

9. Time and Space Complexity

Aspect	Complexity
Time Complexity	O(log n)

Aspect	Complexity
Space Complexity	O(1)

(Binary search is performed twice)

10. Advantages

- Efficient for large arrays
- Accurate handling of duplicates
- Faster than linear scanning
- Uses minimal extra memory

11. Limitations

- Works only on sorted arrays
- Slightly more complex than basic binary search
- Requires careful boundary handling

12. Real-World Applications

- Counting frequency of elements
- Finding range of values in databases
- Search analytics
- Data indexing systems
- Competitive programming problems

13. How to Calculate Count Using First & Last Occurrence

Once found:

```
Count = Last Occurrence − First Occurrence + 1
```

14. Summary

- Used to find starting and ending index of a target
- Works efficiently using binary search
- Requires sorted array
- Time complexity is O(log n)
- Important for duplicate-element problems