

Linear Search

Check every box until you find it. ■

■ What You Will Learn

- What linear search is and when to use it
- How to search through an array step by step
- Return the index or -1 if not found
- Best, worst, and average case analysis

■ Intuition First

Imagine you lost your keys somewhere in your room. You check under the pillow, then the table, then the sofa... one by one. That's **linear search** — check each element from left to right until you find the target or run out of places to look.

■ Intuition Box

Linear search doesn't need the array to be sorted. It works on ANY array. That's its strength. But it checks every element — that's also its weakness for large arrays.

■ Visual: Searching for 7

```
Array: [3] [1] [7] [9] [5]
Index: 0 1 2 3 4
Target: 7
Step 1: arr[0] = 3 ≠ 7 ✗ move right →
Step 2: arr[1] = 1 ≠ 7 ✗ move right →
Step 3: arr[2] = 7 = 7 ✓ FOUND at index 2! ■
3 → 1 → [7] → stop
Checked 3 elements out of 5
```

■■ How It Works

1

Start at index 0

Begin from the very first element of the array.

2	Compare with target Does <code>arr[i]</code> equal what we're looking for?
3	Found? Return index. Return <code>i</code> immediately — no need to check further.
4	Not found? Move right. Increment <code>i</code> and check the next element.
5	Exhausted all? Return -1. If you checked everything and didn't find it, return <code>-1</code> .

■ Code Implementation (Java)

```
public class LinearSearch {  
    // Returns index if found, -1 if not found  
    static int linearSearch(int[] arr, int target) {  
        for (int i = 0; i < arr.length; i++) {  
            if (arr[i] == target) {  
                return i; // found! return position  
            }  
        }  
        return -1; // not in the array  
    }  
    public static void main(String[] args) {  
        int[] arr = {3, 1, 7, 9, 5};  
        int result = linearSearch(arr, 7);  
        if (result != -1)  
            System.out.println("Found at index: " + result);  
        else  
            System.out.println("Not found");  
    }  
}
```

■■ Time & Space Complexity

■ TIME COMPLEXITY

$O(n)$

■ SPACE COMPLEXITY

$O(1)$

Best case: target is at index 0 $\rightarrow O(1)$. Worst case: target is last or not present $\rightarrow O(n)$. Average: $O(n/2) = O(n)$. No extra memory used.

■■ Common Mistakes

■ ■ Not returning -1 for missing elements

If the element isn't found, you must return -1 (or some sentinel value). Returning 0 is wrong — 0 is a valid index!

■ Practice Problems

#	Problem	Difficulty
1	Find if 42 is present in an array of 10 numbers	Easy
2	Return index of first occurrence of a character in a string	Easy
3	Search in a 2D array (matrix)	Medium
4	Count how many times target appears in array	Easy
5	Find and return ALL indices where target occurs	Medium

■ Quick Revision Box

- ✓ Linear search checks every element one by one
- ✓ Works on UNSORTED arrays
- ✓ Returns index if found, -1 if not found
- ✓ Time: $O(n)$ worst case
- ✓ Space: $O(1)$ — no extra memory
- ✓ Best for small arrays or when data is unsorted

■ Visual Thinking

■ Imagine It Like This

Think of a hotel receptionist checking a guest list from top to bottom. They read each name until they find yours. That's linear search. Animation idea: highlight each array cell one by one with a magnifying glass emoji moving across.