

3.3 Check if Array is Sorted

EASY

■ What You Will Learn

- ✓ How to verify ascending order in one pass
- ✓ How to handle equal adjacent elements
- ✓ Return early on first violation

■ Intuition First

Think of checking stairs. You just need to make sure each step is equal to or higher than the one before it. If you ever find a step that's lower — it's not sorted. Otherwise, reach the top and you're done.

■ The Core Idea

- Compare `arr[i]` with `arr[i+1]` for every adjacent pair
- If `arr[i] > arr[i+1]`: not sorted — return false immediately
- If you finish the loop without returning false: it's sorted!

■ Visual Trace

Sorted array ✓

1	2	3	4	5
---	---	---	---	---

`1 ≤ 2 ✓ 2 ≤ 3 ✓ 3 ≤ 4 ✓ 4 ≤ 5 ✓ → return true`

NOT sorted ✗

1	3	2	4	5
---	---	---	---	---

`1 ≤ 3 ✓ 3 > 2 ✗ → return false immediately (early exit!)`

■ Java Code

```
public static boolean isSorted(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        if (arr[i] > arr[i + 1]) {
            return false; // Found a violation – stop here
        }
    }
    return true; // No violation found
}
```

```
}

```

■ ■ Complexity

Case	Time	Space	What happens?
Best	O(1)	O(1)	First two elements out of order — early exit
Worst	O(n)	O(1)	Fully sorted — must check every pair

■ ■ Common Mistakes

- Loop condition is `i < arr.length - 1` (not `arr.length`) — avoid index out of bounds
- Use `arr[i] > arr[i+1]` for ascending; `arr[i] < arr[i+1]` for descending

■ Pro Tip

- Allow equal elements (`arr[i] <= arr[i+1]`) for non-decreasing order
- For strictly increasing, use `arr[i] >= arr[i+1]` to detect violations

■ Practice Problems

- [Easy] Check if [5,4,3,2,1] is sorted in descending order
- [Easy] Check if array is sorted and count violations
- [Med] Find the index where the array stops being sorted

■ Quick Revision

- Compare adjacent pairs — `arr[i] > arr[i+1]` means not sorted
- Return false immediately on first violation
- O(n) worst, O(1) best — early exit is key
- Loop to length-2 not length-1