DAY 07: Check if an Array is Sorted

"The best error message is the one that never shows up, because you designed your code properly."
— Thomas Fuchs

Crack DSA with Java

By: Bhavya Solanki and Pernay Chauhan

Chapters:

LIST OF FIGURES

1. INTRODUCTION

In the world of Data Structures and Algorithms (DSA), working with arrays is one of the foundational concepts. One of the most frequent checks performed on arrays is **whether the array is sorted or not**. A sorted array is one where the elements are arranged either in increasing or decreasing order. Verifying if an array is sorted is useful in many real-world applications like optimizing search operations or validating input data before processing.

In Java, this check can be implemented efficiently with a simple loop and basic conditional statements. It's a great beginner-friendly problem to understand loops, conditional logic, and array traversal.

1.1 Why Check If an Array is Sorted?

- To **optimize performance**: No need to sort again if already sorted.
- Required in **binary search**, which only works on sorted arrays.
- Common pre-processing step in **competitive programming** and **real-world systems** like file handling, report generation, etc.

2. BRIEF DESCRIPTION

Arrays are linear data structures that store elements of the same type in contiguous memory locations. In many algorithmic problems, particularly involving **searching, merging, or optimization**, we often need to know if the input array is **already sorted**. The goal is to check whether the given array follows a monotonic increasing (ascending) or monotonic decreasing (descending) pattern.

2.1 What do we mean by "Sorted"?

- **Ascending Order:** arr[i] <= arr[i + 1] for every valid i
- **Descending Order:** arr[i] >= arr[i + 1] for every valid i

2.2 Step-by-Step Logic (for Ascending Order):

1. Start from index 0
2. Compare each element with its next one.
3. If at any index i, arr[i] > arr[i + 1], the array is not sorted.
4. If no such pair is found, the array is sorted.

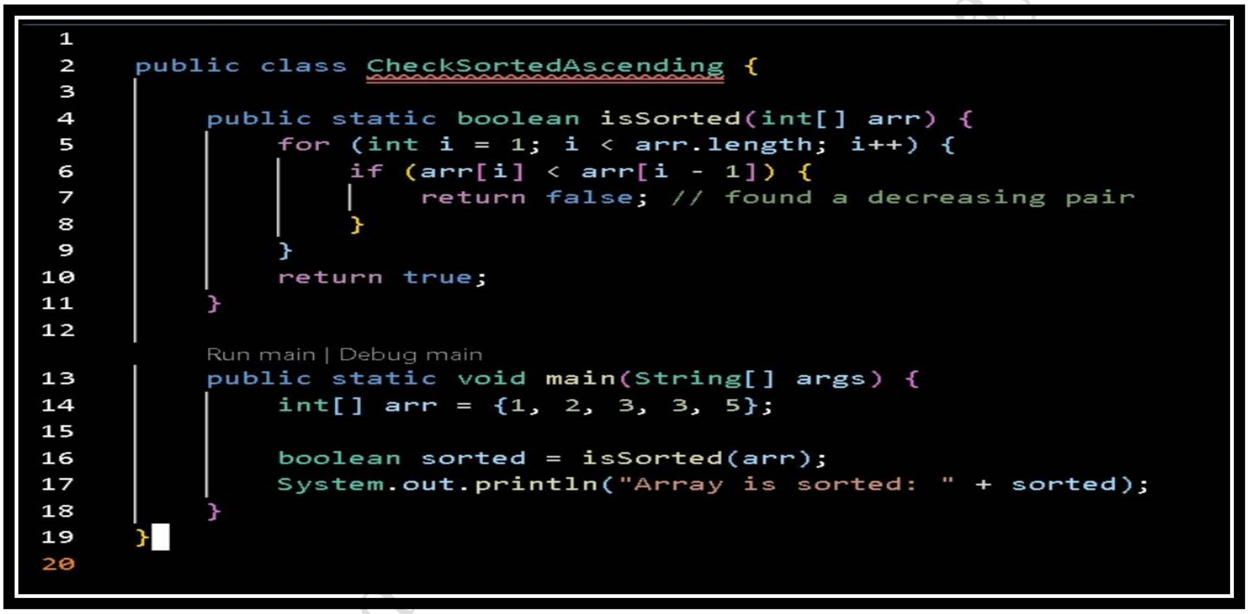
2.3 Key Takeaways:

- Works in **linear time (O(n))**
- No need for extra space
- Handles both **ascending** and **descending**
- Can be adapted for custom objects (e.g., sorting by name, age)
- Essential for **binary search**, **optimizations**, and **DSA interviews.**

3. PROBLEM 1

3.1 Problem Statement: Write a Java program to check whether an array is sorted in non-decreasing (ascending) order. Return true if sorted, else false.

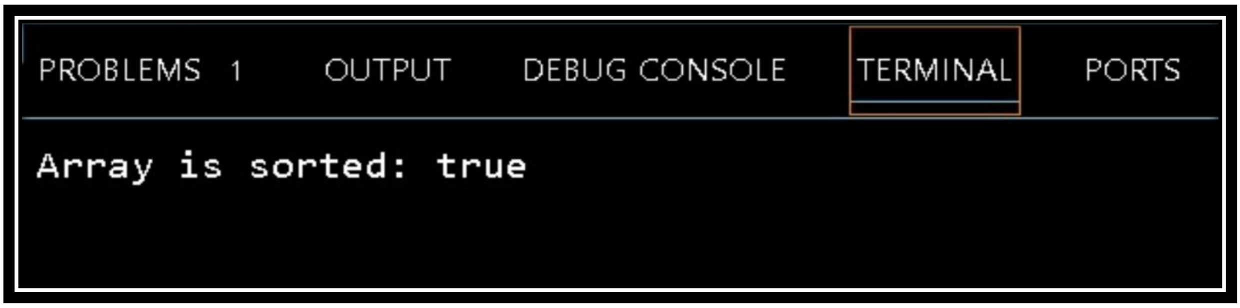
3.2 Code (Input):



```java

public class CheckSortedAscending {

    public static boolean isSorted(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] < arr[i - 1]) {
                return false; // found a decreasing pair
            }
        }
        return true;
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 3, 5};

        boolean sorted = isSorted(arr);
        System.out.println("Array is sorted: " + sorted);
    }
}

```

Figure 3.2.1: Code of Problem Statement 1

3.3 Output:



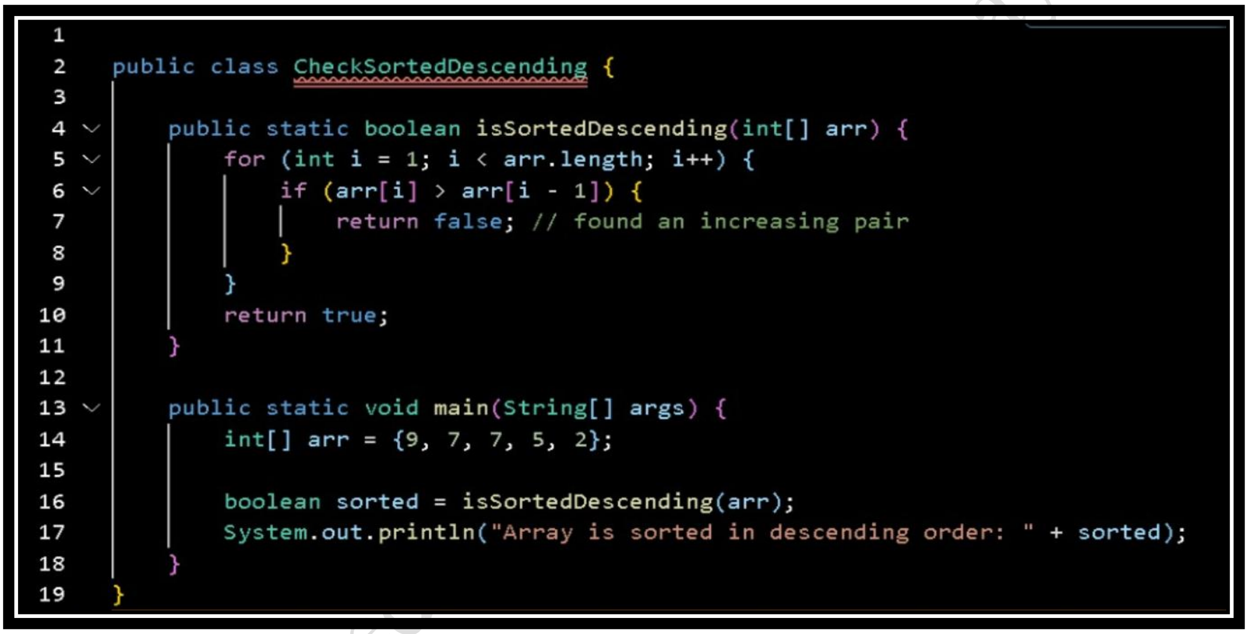
```
Array is sorted: true
```

Figure 3.3.1: Output of Problem Statement 1

4. PROBLEM 2

4.1 Problem Statement: Given an integer array, check whether it is sorted in non-increasing (descending) order. Return true if sorted, else false.

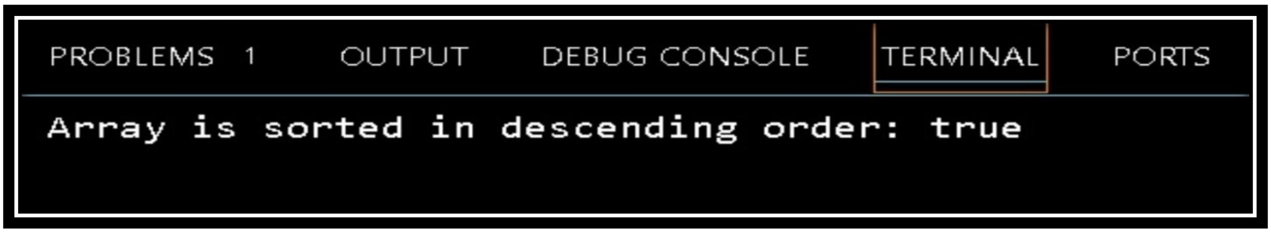
4.2 Code (Input):



```java

public class CheckSortedDescending {

    public static boolean isSortedDescending(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            if (arr[i] > arr[i - 1]) {
                return false; // found an increasing pair
            }
        }
        return true;
    }

    public static void main(String[] args) {
        int[] arr = {9, 7, 7, 5, 2};

        boolean sorted = isSortedDescending(arr);
        System.out.println("Array is sorted in descending order: " + sorted);
    }
}
```

Figure 4.2.1: Code of Problem Statement 2

4.3 Output:



```
Array is sorted in descending order: true
```

Figure 4.3.1: Output of Problem Statement 2

5. ADVANTAGES AND DISADVANTAGES

5.1 Advantages:-

1. **Time Efficient:** Only one pass through array → O(n) time.
2. **Space Efficient:** No extra data structure needed → O(1) space.
3. **Useful Before Binary Search:** Ensures validity of input.
4. **Fast Preprocessing:** Helps avoid unnecessary sorting.
5. **Good for Edge Detection:** In hybrid sorting algorithms (like TimSort).

5.2 Disadvantages:-

1. **Only Validation:** Doesn’t sort the array, only checks order.
2. **Not Useful for Very Small Arrays:** For tiny datasets, sorting might be quicker and simpler.
3. **Separate Logic for Descending:** Requires extra steps to check reverse order.
4. **Can’t Detect Patterned or Partially Sorted Data:** Doesn't indicate “almost sorted” arrays.

6. CONCLUSION

Checking whether an array is sorted is a fundamental operation in Data Structures and Algorithms (DSA), especially in Java programming. This operation helps developers determine if the data is already in the required order before applying operations like **binary search**, **merging**, or **optimization algorithms**. Since the process only requires a single linear pass, it is highly **time-efficient (O(n))** and does not require any additional space, making it **space-efficient (O(1))** as well.

This check is particularly useful in real-world applications, such as sorting data for reports, verifying sorted logs, or confirming preconditions in system workflows. It is also frequently used in **competitive programming** and **technical interviews** as a warm-up or optimization problem.

Key Points to Remember:

- It checks for **ascending**, **descending**, or **unsorted** arrays.
- Uses a simple **comparison loop** to validate order.
- Efficient with **O(n)** time and **O(1)** space complexity.
- Works for **primitive types**, **strings**, and even **custom objects**.
- Helps avoid unnecessary sorting operations and speeds up overall program execution.

By mastering this concept, you'll improve your problem-solving skills and be better prepared to optimize code in both academic and real-world software development tasks.

7. FREQUENTLY ASKED QUESTIONS (FAQs)

Q1: Can this logic be used for arrays of strings or characters?

Answer: Yes, but you must compare using .compareTo() for strings or use character ASCII values.

Q2: What is the time complexity of checking if an array is sorted?

Answer: The time complexity is O(n) since we check each element once.

Q3: How do I check for descending order?

Answer: Just modify the condition in the loop:

```
if (arr[i] < arr[i + 1]) return false;
```

Q4: Does this method change the original array?

Answer: No, it only reads the array. The original array remains unchanged.

Q5: How to check if the array is either ascending or descending?

Answer: You can use two flags:

```
boolean ascending = true, descending = true;
for (int i = 0; i < arr.length - 1; i++) {
   if (arr[i] > arr[i + 1]) ascending = false;
   if (arr[i] < arr[i + 1]) descending = false;
}
if (ascending || descending) System.out.println("Sorted");
else System.out.println("Not Sorted");
```