Day 6: Check If an Array is Sorted

Abhinav Yadav

"C is quirky, flawed, and an enormous success." — Dennis Ritchie

---

1 Introduction

Checking if an array is sorted is a fundamental problem in programming. It provides insights into data order, crucial for optimizing algorithms like binary search and efficient data manipulation. This problem evaluates whether the array is sorted in ascending or descending order using a single traversal.

2 Problem Statement

Problem: Determine if a given array is sorted in ascending or descending order. **Hint:** Traverse the array and compare adjacent elements. **Edge Case:** Arrays of size 1 or containing all identical elements are considered sorted.

3 Algorithm

3.1 Steps to Solve the Problem

1. Initialize flags to determine ascending or descending order.
2. Traverse the array and compare each element with the next:
   - If a pair violates ascending order, mark it as not ascending.
   - Similarly, if a pair violates descending order, mark it as not descending.
3. If neither flag is true, the array is unsorted.

4 Code

```
import java.util.Scanner;

public class CheckArraySorting {
```

```java
// Function to check if the array is sorted in ascending order
public static boolean isAscending(int[] arr, int n) {
    for (int i = 0; i < n - 1; i++) {
        if (arr[i] > arr[i + 1]) {
            return false;
        }
    }
    return true;
}

// Function to check if the array is sorted in descending order
public static boolean isDescending(int[] arr, int n) {
    for (int i = 0; i < n - 1; i++) {
        if (arr[i] < arr[i + 1]) {
            return false;
        }
    }
    return true;
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the size of the array: ");
    int n = scanner.nextInt();

    if (n < 2) {
        System.out.println("Array size must be at least 2 to check for
        scanner.close();
        return;
    }

    int[] arr = new int[n];
    System.out.println("Enter the elements of the array:");
    for (int i = 0; i < n; i++) {
        arr[i] = scanner.nextInt();
    }

    if (isAscending(arr, n)) {
        System.out.println("The array is sorted in ascending order.");
    } else if (isDescending(arr, n)) {
        System.out.println("The array is sorted in descending order.");
    } else {
        System.out.println("The array is not sorted.");
    }

    scanner.close();
}
```

}

5 Step-by-Step Explanation

1. Traverse the array and compare adjacent elements:
   - If `arr[i] > arr[i+1]`, it breaks ascending order.
   - If `arr[i] < arr[i+1]`, it breaks descending order.
2. Return the appropriate message based on flags.

6 Complexity Analysis

6.1 Time Complexity

- Single traversal of the array ensures $O(n)$ time complexity.

6.2 Space Complexity

- In-place computations ensure $O(1)$ space complexity.

7 Examples and Edge Cases

Input Array	Output	Remarks
{1, 2, 3, 4}	Sorted in Ascending Order	Valid Input
{4, 3, 2, 1}	Sorted in Descending Order	Valid Input
{1, 3, 2, 4}	Not Sorted	Mixed Order
{7, 7, 7}	Sorted in Ascending Order	Identical Elements

8 Conclusion

This program efficiently determines the order of a given array using $O(n)$ time and $O(1)$ space. Handling edge cases like identical elements or arrays of size 1 ensures robustness.

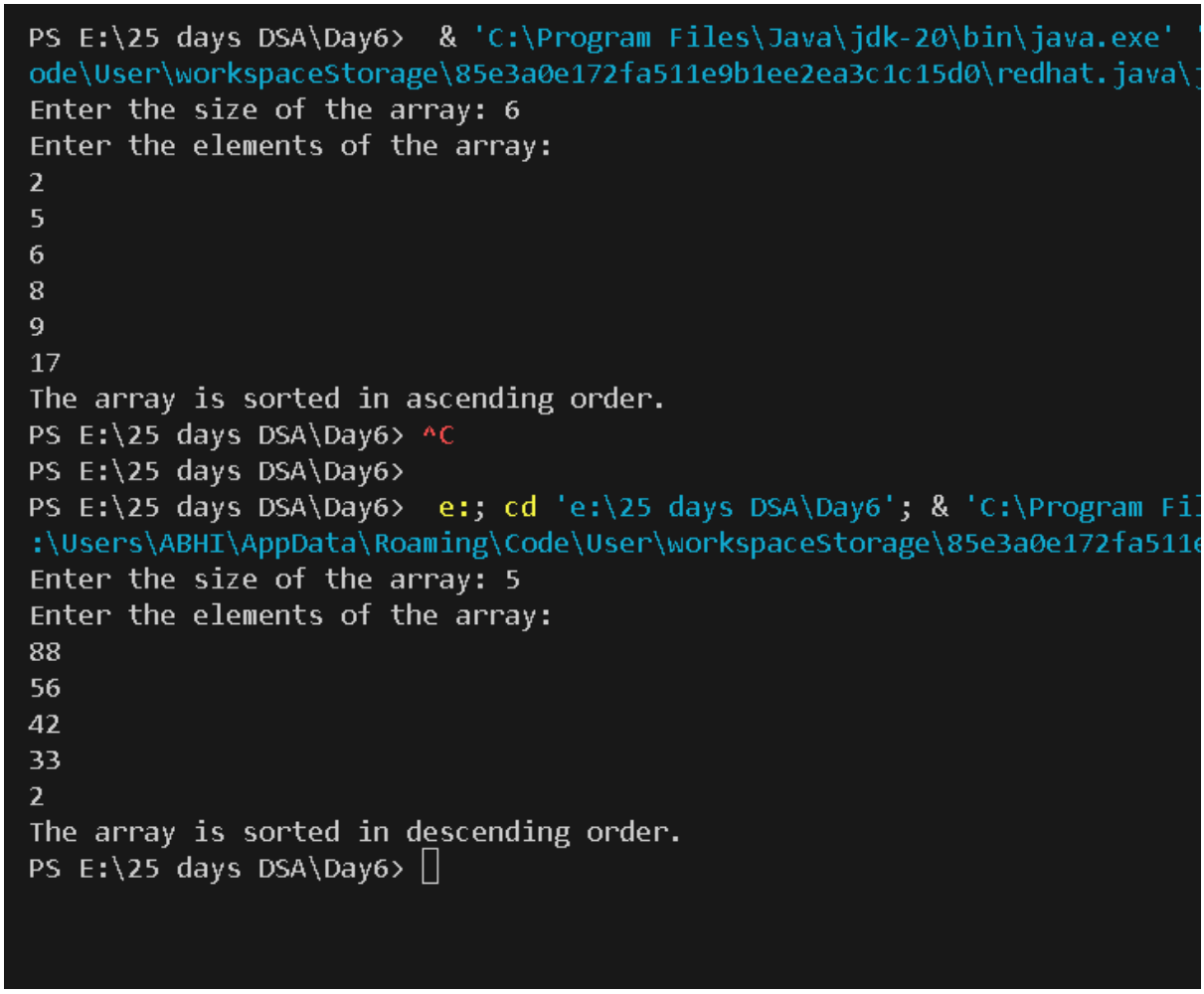

Figure 1: Output in online compiler