Day 3: Reverse an Array

Abhinav Yadav

"Simplicity is the soul of efficiency."
— Austin Freeman

---

1 Introduction

Reversing an array is a classic problem in computer science that emphasizes in-place manipulation of data to optimize memory usage. In this task, we reverse the array without using any additional arrays, utilizing the two-pointer technique.

2 Problem Statement

Problem: Reverse an array in-place without using an extra array. **Hint:** Use the two-pointer technique to swap the first and last elements, then move the pointers inward until they meet or cross.

3 Algorithm

3.1 Two-Pointer Technique

1. Initialize two pointers:
   - `left` at the beginning of the array.
   - `right` at the end of the array.
2. While `left` is less than `right`:
   - Swap the elements at `left` and `right`.
   - Increment `left` and decrement `right`.
3. Repeat until all elements are reversed.

4 Code

```
import java.util.Scanner;

public class ReverseArray {

    // Function to reverse an array in-place using two-pointer technique
    public static void reverseArray(int[] arr, int n) {
        int left = 0, right = n - 1;
        while (left < right) {
            // Swap the elements at the left and right pointers
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;

            // Move the pointers inward
            left++;
            right--;
        }
    }

    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();

        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();
        }

        // Reverse the array
        reverseArray(arr, n);

        // Print the reversed array
        System.out.println("Reversed Array:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();

        scanner.close();
    }
}
```

5 Step-by-Step Explanation of Swaps

1. **Initialization:** Pointers `left` and `right` are set to the first and last indices, respectively.
2. **Iteration 1:**
   - Swap `arr[left]` and `arr[right]`.
   - Move `left` and `right` pointers inward.
3. **Subsequent Iterations:** Continue swapping and moving pointers inward until `left` $\geq$ `right`.
4. **Completion:** All elements are reversed in-place.

6 Complexity Analysis

6.1 Time Complexity

- The array is traversed once, resulting in a time complexity of $O(n)$.

6.2 Space Complexity

- Since the reversal is done in-place, the space complexity is $O(1)$.

7 Advantages of Two-Pointer Technique

- Reduces memory usage as no extra array is needed.
- Efficiently handles arrays of any size.

8 Conclusion

The two-pointer technique effectively reverses an array in-place, minimizing memory usage and ensuring simplicity in implementation. This approach is ideal for scenarios where memory is constrained.

9 Output

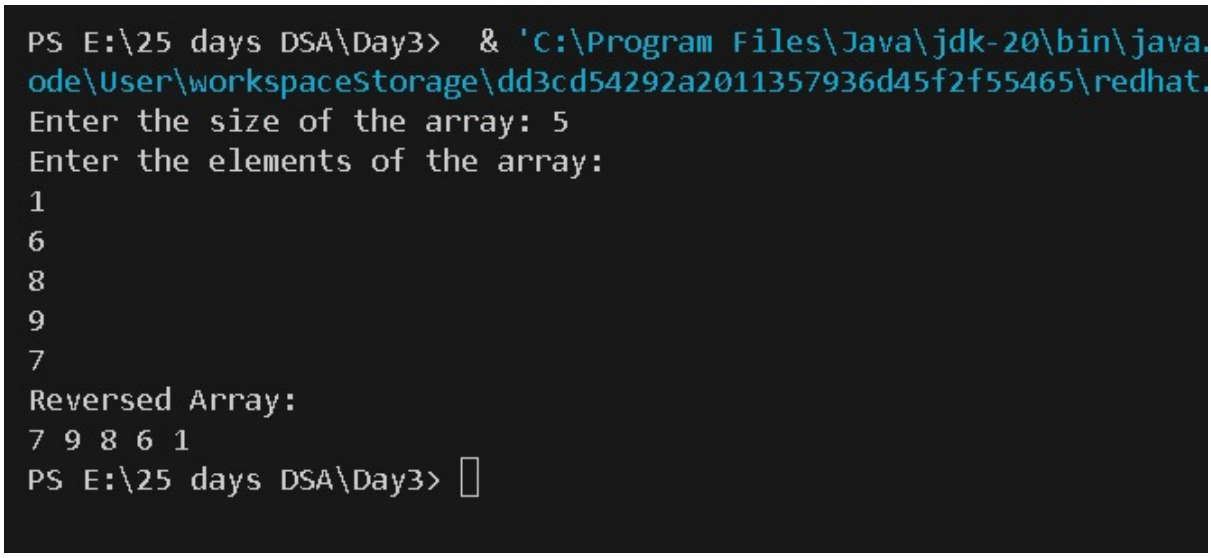

Figure 1: Output