

# Day 3: Reverse an Array

Gati Goyal

---

*"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

```
#include <stdio.h>

// Function to reverse an array in-place
void reverseArray(int arr[], int n) {
    int left = 0, right = n - 1;
    while (left < right) {
        // Swap the elements
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;

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

int main() {
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    reverseArray(arr, n);

    printf("Reversed Array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

## 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  $\text{left} \geq \text{right}$ .
  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 Output

The screenshot shows an online compiler interface with a C program to reverse an array. The code defines a `reverseArray` function using two pointers (`left` and `right`) and a temporary variable to swap elements. The `main` function prompts the user for the array size and elements, then calls `reverseArray` and prints the reversed array.

```

1 #include <stdio.h>
2 void reverseArray(int arr[], int n) { // Function to reverse an array
3     int left = 0, right = n - 1;
4     while (left < right) { // Swap the elements at the left and right pointers
5         int temp = arr[left];
6         arr[left] = arr[right];
7         arr[right] = temp;
8         left++; // Move the pointers inward
9         right--;
10    }
11 }
12
13 int main() {
14     int n;
15     printf("Enter the size of the array: ");
16     scanf("%d", &n);
17
18     int arr[n];
19     printf("Enter the elements of the array:\n");
20     for (int i = 0; i < n; i++) {
21         scanf("%d", &arr[i]);
22     }
23
24     reverseArray(arr, n); // Reverse the array
25
26     printf("Reversed Array:\n"); // Print the reversed array
27     for (int i = 0; i < n; i++) {
28         printf("%d ", arr[i]);
29     }
30     printf("\n");
31
32     return 0;
33 }
  
```

The output of the program is as follows:

```

Enter the size of the array: 5
Enter the elements of the array:
96
45
72
11
19
Reversed Array:
19 11 72 45 96
  
```

Below the output, it states: `=== Code Execution Successful ===`

Figure 1: Output in online compiler

## 9 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.