Power Of Two

You are given an array of integers. Your task is to write a program to swap the alternate elements in the array. If the array has an odd number of elements, leave the last element unchanged.

For example, given the array [1, 2, 3, 4, 5, 6, 7, 8], your program should swap the alternate elements to produce the array [2, 1, 4, 3, 6, 5, 7, 8].

Example:

Input:

Array: [1, 2, 3, 4, 5, 6, 7, 8]

Size: 8

Output:

Modified Array: [2, 1, 4, 3, 6, 5, 7, 8]

Note:

The array may have both positive and negative integers.

If the array has an odd number of elements, the last element should remain unchanged.

You need to implement three different approaches for swapping the alternate elements and analyze their time and space complexity.

Approach 1: Using a temporary variable inside the swapAlternative function

This approach uses a temporary variable to swap the elements at alternate positions in the array. It iterates through the array and swaps the current element with the next element by using a temporary variable.

This approach has a **time complexity of O(n)** since it iterates through the array once.

The **space complexity is O(1)** because it only requires a single temporary variable to perform the swaps.

Approach 2: Using XOR operation for swapping elements

This approach utilizes the XOR operation to swap the elements at alternate positions without using a temporary variable. It XORs the values of two elements to perform the swap.

Similar to Approach 1, this approach also has a **time complexity of O(n)** since it iterates through the array once.

The **space complexity is O(1)** as it doesn't require any additional space.

Approach 2 (using XOR operation) can be seen as an optimization technique to avoid using an additional variable. It may have a slight advantage in terms of space complexity, as it avoids using an extra variable. However, the performance gain is negligible, and it might be less intuitive for someone reading the code.