Rearrange Array Elements by Sign

Given an array with an equal number of positive and negative elements, rearrange the array so that positive and negative elements alternate. The order of positives and negatives should be maintained.

Problem Statement

You're given an array `nums` of even length `n`, containing `n / 2` positive integers and `n / 2` negative integers. Your task is to rearrange the array such that:

1. Every consecutive pair of integers have opposite signs.
2. For all positive integers, the order in `nums` is preserved.
3. For all negative integers, the order in `nums` is preserved.

Return the rearranged array.

Example:

```
Input: nums = [3,1,-2,-5,2,-4]
Output: [3,-2,1,-5,2,-4]
Explanation:
The positive integers are [3,1,2]. The negative integers are [-2,-5,-4].
The only possible arrangement satisfying all conditions is [3,-2,1,-5,2,-4].
```

Brute Force Approach

1. Create two separate lists (or arrays), one for positive numbers and one for negative numbers.
2. Iterate through the input array. If a number is positive, add it to the positive list; if it's negative, add it to the negative list.
3. Create a new array to store the rearranged elements.
4. Iterate using indexes, filling the even indexes with positive numbers from the positive list and the odd indexes with negative numbers from the negative list.

```python
def rearrange_array(nums):
    positives = []
    negatives = []
    for num in nums:
        if num > 0:
            positives.append(num)
        else:
            negatives.append(num)

    rearranged = [0] * len(nums)

    pos_index = 0
    neg_index = 0

    for i in range(len(nums)):
        if i % 2 == 0:
            rearranged[i] = positives[pos_index]
            pos_index += 1
        else:
            rearranged[i] = negatives[neg_index]
            neg_index += 1

    return rearranged
```

Time Complexity: $O(n)$ (one pass to separate positives and negatives, another to rearrange) *Space Complexity*: $O(n)$ (two lists of size `n / 2` each)

Optimal Approach

1. Create a new array `ans` of the same size as the input array `nums`.
2. Initialize two index pointers: `pos_index = 0` (for even positions where positive numbers will go) and `neg_index = 1` (for odd positions where negative numbers will go).
3. Iterate through the original `nums` array.
    - If the current number is positive, place it in the `ans` array at `pos_index` and increment `pos_index` by 2.
    - If the current number is negative, place it in the `ans` array at `neg_index` and increment `neg_index` by 2.

```python
def rearrange_array_optimal(nums):
    n = len(nums)
    ans = [0] * n
    pos_index = 0
    neg_index = 1

    for num in nums:
        if num > 0:
            ans[pos_index] = num
            pos_index += 2
        else:
            ans[neg_index] = num
            neg_index += 2
    return ans
```

Time Complexity: $O(n)$ (single pass through the array) *Space Complexity*: $O(n)$ (additional array of size `n`)

Handling Unequal Positive and Negative Elements

Problem Statement

Similar to the previous problem, but now the number of positive and negative integers may not be equal.

You're given an array `nums` containing positive and negative integers. Your task is to rearrange the array such that:

1. Every consecutive pair of integers have opposite signs, starting with a positive number.
2. If there are more positive or negative numbers, append the remaining ones at the end of the array while maintaining their order.

Example:

```
Input: nums = [3,1,-2,-5,2,-4, 6]
Output: [3,-2,1,-5,2,-4, 6]
```

General Approach

The key idea is to use a similar approach to the optimal solution but to handle the cases where positive or negative numbers are left over.

1. Use the same optimal approach until one of the categories runs out of elements
2. Append the remaining elements to the end

Python Implementation

```python
def rearrange_array_unequal(nums):
    n = len(nums)
    ans = []
    pos_nums = [num for num in nums if num > 0]
    neg_nums = [num for num in nums if num < 0]

    pos_index = 0
    neg_index = 0

    while pos_index < len(pos_nums) and neg_index < len(neg_nums):
        ans.append(pos_nums[pos_index])
        ans.append(neg_nums[neg_index])
        pos_index += 1
        neg_index += 1

    # Add any remaining positive numbers
    while pos_index < len(pos_nums):
        ans.append(pos_nums[pos_index])
        pos_index += 1

    # Add any remaining negative numbers
    while neg_index < len(neg_nums):
        ans.append(neg_nums[neg_index])
        neg_index += 1

    return ans
```

Time Complexity: $O(n)$ *Space Complexity*: $O(n)$

Solving the Problem When Positives and Negatives Are Not Equal

When the number of positive and negative numbers in the array are not equal, the optimal solution used previously (where we alternate positive and negative numbers) can't be directly applied. In such cases, we need to consider two scenarios:

1. The number of positives is greater than the number of negatives.
2. The number of negatives is greater than the number of positives.

Since the optimal solution relies on having an equal number of positives and negatives, we fall back to a **brute force** approach.

Brute Force Approach

The brute force approach involves the following steps:

1. Create two separate arrays: one for storing positive numbers and another for storing negative numbers. Maintain the relative order of elements as you iterate through the original array.

2. After separating the positive and negative numbers, determine the smaller count between them. For instance, if there are more positive numbers than negative numbers, the number of negative numbers becomes the limiting factor. The initial elements in the rearranged array can then follow an alternating pattern based on this limiting factor.

3. Once the alternating pattern is established using the minimum count of either positive or negative numbers, any remaining numbers (either positive or negative) are appended to the end of the rearranged array.

Example Scenario

Let's say we have an array where the number of positives is not equal to the number of negatives. We create two additional arrays, a positive array and a negative array, to store the numbers.

For example, with two negatives, the first four elements will be in order (e.g., 2, -1, 3, -3) because the number of negatives is two, which is less. In this case, we're sure of having two positives and two negatives that can be arranged in an alternating order.

Algorithm Steps

1. Iterate up to the smaller count (e.g., number of negatives).
2. Fill the array using the formula:
    - `array[i * 2] = positive[i]`
    - `array[i * 2 + 1] = negative[i]`

    This ensures that the array is filled with alternating positive and negative numbers up to the point where either all positive or all negative numbers are exhausted.
3. After placing the known elements in an alternating pattern, handle the remaining positive numbers. Start from the index in the positive array where the elements have not been placed yet. Then, place these remaining positive numbers in the resultant array.
4. Fill the remaining values in the array.

Coding the Solution

Here's how to approach the coding part:

1. Store the positives and negatives in separate data structures (`positive` and `negative`).

2. Iterate through the array to determine the size of the array.

```
int size = array.size();
```

3. Iterate through the array and store positive and negative numbers into their respective lists.

```
for (int i = 0; i < size; i++) {
    if (array[i] > 0) {
        positive.push_back(array[i]);
    } else {
        negative.push_back(array[i]);
    }
}
```

4. Check if the number of positive numbers is greater than the number of negative numbers or vice versa.

5. Iterate based on the count of the smaller array. For example, if the number of negatives is less than the number of positives:

```
for (int i = 0; i < negative.size(); i++) {
    array[i * 2] = positive[i];
    array[i * 2 + 1] = negative[i];
}
```

6. For the remaining positive numbers, start from the correct index and fill them in the array:

```
int index = negative.size() * 2;
for (int i = negative.size(); i < positive.size(); i++) {
    array[index] = positive[i];
    index++;
}
```

7. The same steps are repeated if there are more negatives than positives, but the roles are reversed.

8. Return the modified array.

Time and Space Complexity

- Time Complexity: `O(n)` in the worst-case scenario because we iterate through the array to separate positives and negatives, and then we iterate again to rearrange them.
    - First iteration to get all positives and negatives.
    - Second loop runs for the minimum of positives or negatives.
    - Third loop runs for the leftovers.
- Space Complexity: `O(n)` because we store all the positive and negative numbers in separate arrays.