Documentation

To solve the problem of finding two unique numbers in an array where all other elements appear exactly twice, we can utilize bitwise operations, specifically the XOR operation. This approach is particularly efficient as it runs in linear time complexity, (O(n)), and only requires constant space, satisfying the problem's constraints. The key insight is to use the properties of XOR to cancel out numbers that appear twice, leaving only the two unique numbers. This allows us to avoid sorting or using additional data structures like hash maps, making the solution both space- and time-efficient.

The solution starts by XORing all elements in the array. Since XORing a number with itself results in zero (i.e., ($x \oplus x = 0$)), any numbers that appear twice in the array effectively cancel out, leaving the XOR of just the two unique numbers, (a) and (b). The result of this XOR operation (let's call it xor_all) will thus be the XOR of (a) and (b) alone, as all duplicates have been eliminated. However, this result contains the combined XOR of (a) and (b), but it does not tell us each number.

To isolate these two unique numbers from each other, we need to identify a bit position where (a) and (b) differ, as this is the basis for distinguishing between the two. By identifying any bit that is set in xor_all, we can be sure that (a) and (b) differ at that position. A useful trick to find the rightmost set bit in a number is by calculating xor_all & -xor_all. This operation isolates the lowest bit that is set to 1 in xor_all, revealing a bit position where (a) and (b) must differ. This bit, known as the "difference bit," will serve as the criterion to separate the array into two distinct groups.

Once we have the difference bit, we split the numbers in the array into two groups based on whether each number has this bit set or not. Since (a) and (b) differ at this bit, they will end up in separate groups. Meanwhile, each group will also contain other numbers, but because all other elements appear twice, they will cancel out within each group. This leaves only (a) in one group and (b) in the other, as all duplicates are neutralized by the XOR operation.

Finally, we perform an XOR operation on each group individually. In the first group, we XOR all numbers where the difference bit is set to 1, which ultimately results in one of the unique numbers, say (a). In the second group, where the difference bit is 0, we XOR all the numbers, leaving us with the second unique number, (b). Thus, by dividing the numbers based on a single bit difference and applying XOR within each group, we can identify the two unique numbers in the array without extra space or additional data structures.

This method is optimal for cases requiring linear time complexity and minimal space usage, making it a powerful approach for handling large arrays with similar requirements. The solution is elegant in its simplicity and leverages the XOR operation effectively to solve a problem that might otherwise require more cumbersome methods. Through this approach, we achieve both efficiency and clarity, ensuring the two unique numbers are found accurately and with minimal computational overhead.