Insecure (Don't Know What For)

As established in the previous problem, $m = x \oplus y \oplus z$. So, what we want to do is evaluate this sum:

$$\sum_{\text{valid } (i,j,k)} v_i \oplus v_j \oplus v_k$$

Standard Approach: Bitwise Decomposition

When doing problems involving bitwise operations, decompose your numbers into their binary representations, and consider each place value as a **separate** independent problem.

That is: can you solve the original problem if your array was just a sequence of `0`s and `1`s? If yes, then handle each *place value* separately.

Standard Toolbox: Switch the order of summation

When dealing with double (or more) summations, consider swapping the order of the Sigmas.

Changing which variable goes in the inner loop can, for example, allow you to factor out a term that is no longer dependent on the inner summation.

We can consider the binary representation of each of our integers, and explicitly write each one's expanded form. For example,

$$v_i = (\text{0th bit of } v_i) + (\text{1th bit of } v_i)2^1 + (\text{2th bit of } v_i)2^2 + \ldots + (\text{29th bit of } v_i)2^{29}$$

Since the bitwise XOR is computed by independently performing a logical XOR on each place value, we can rewrite our above sum as

$$\sum_{\text{valid } (i,j,k)} \sum_{b=0}^{30-1} 2^b \cdot ((b\text{th bit of } v_i) \oplus (b\text{th bit of } v_j) \oplus (b\text{th bit of } v_k)) .$$

Now interchange the order of the summations.

$$\sum_{b=0}^{30-1} \sum_{\text{valid } (i,j,k)} 2^b \cdot ((b\text{th bit of } v_i) \oplus (b\text{th bit of } v_j) \oplus (b\text{th bit of } v_k))$$

Since 2^b is not dependent on the inner summation—which iterates over pairwise distinct (i, j, k)—we can factor it out.

$$\sum_{b=0}^{30-1} 2^b \sum_{\text{valid } (i,j,k)} (b\text{th bit of } v_i) \oplus (b\text{th bit of } v_j) \oplus (b\text{th bit of } v_k)$$

Our sum is now just a *counting* problem. For each place-value b, answer independently: How many pairwise distinct (i, j, k) are there such that

$$(b\text{th bit of } v_i) \oplus (b\text{th bit of } v_j) \oplus (b\text{th bit of } v_k) = 1$$

It's not too hard to enumerate all such cases:

bth bit of v_i	bth bit of v_j	bth bit of v_k
1	1	1
0	0	1
0	1	0
1	0	0

Let $\mathcal{O}(b)$ be the number of elements with a `1` bit at the bth place value, and let $Z(b)$ be the number of elements with a `0` bit at the bth place value. By standard combinatorics' rule of product, the number of ways to form each of the above combinations is:

- $\mathcal{O}(b) \cdot (\mathcal{O}(b) - 1) \cdot (\mathcal{O}(b) - 2)$
- $Z(b) \cdot (Z(b) - 1) \cdot \mathcal{O}(b)$
- $Z(b) \cdot \mathcal{O}(b) \cdot (Z(b) - 1)$
- $\mathcal{O}(b) \cdot Z(b) \cdot (Z(b) - 1)$

Each of these can be computed in $O(n)$, and doing it for all b gives us an $O(n \log v)$ solution.

Implementation

For C++ users, *note that the answer can be large.* We can show that the maximum value of $\approx 2^{30} \cdot 4(10^5)^3$ does not fit in a 64 bit data type, but it *does* fit in a 128 bit integer like `__int128`.

The data type `__int128` is not supported by `cout`, but it is not too hard to manually output it digit-by-digit.