Problem I. GCD vs. XOR

Input file:	*standard input*
Output file:	*standard output*
Time limit:	20 seconds
Memory limit:	512 mebibytes

Optimizing is fun! Especially when it's not exactly required.

Everyone knows that bit operations (e.g. bitwise XOR) are faster than recursive functions (such as the greatest common divisor, GCD). To impress your internship supervisors you replaced, in the company's flagship project, all instances of $gcd(x, y)$ with much quicker $xor(x, y)$.

That was yesterday, on Friday. Now you start thinking whether you should have tested your new code before deploying to production... Well, better late than never. Given a sequence of numbers $a_1, \ldots, a_n$, determine how many pairs (i, j) $(1 \le i < j \le n)$ actually satisfy $gcd(a_i, a_j) = xor(a_i, a_j)$. Recall that $gcd(x, y)$ denotes the greatest common divisor of x and y, while $xor(x, y)$ is the bitwise-XOR operation on x and y.

Input

The first line of input contains the number of test cases z $(1 \le z \le 20)$. The descriptions of the test cases follow.

The first line of a test case contains an integer n $(1 \le n \le 2\,000\,000)$. The second line contains integers $a_1, a_2, \ldots, a_n$, all positive and not exceeding $1\,000\,000$.

The total length of all sequences over all test cases does not exceed $3 \cdot 10^7$.

Output

For each test case output a single integer: the number of pairs (a_i, a_j) with $i < j$ satisfying $gcd(a_i, a_j) = xor(a_i, a_j)$.

Example

standard input	standard output
1 4 2 3 4 3	2