min-xor

Consider two integers a and b. Their bitwise `xor` (exclusive or) is computed as it follows:

- Write each of them in binary: $a = \overline{a_{k-1}a_{k-2}\dots a_0}$ and $b = \overline{b_{k-1}b_{k-2}\dots b_0}$;
- Their bitwise `xor`, denoted by a ^ b, has a bit equals to 1 in the position p if and only if exactly one of a_p and b_p has a value of 1.

For example, 9 ^ $3 = 10$ because $9 = 1001_{(2)}$, $3 = 0011_{(2)}$ and $10 = 1010_{(2)}$.

Task

You are given a sequence of N operations on a set of integers. The set is initially empty. Each operation can be one of the following:

1. *insert(x):* append integer x to the set;
2. *delete(x):* remove integer x from the set;
3. *min-xor():* print the smallest bitwise `xor` between any two integers currently in the set.

You must print the output from every *min-xor()* operation.

Input data

The first line of the input file `minxor.in` contains the number of operations *N*. Each of the following lines describes an operation using the following syntax:

- `1 x` for *insert(x)*
- `2 x` for *delete(x)*
- `3` for *min-xor()*

Output data

The output file `minxor.out` must contain a number of lines equals to the number of *min-xor()* operations. Each line must contain a single number, the answer to the corresponding *min-xor()* operation.

Limits and constraints

- $1 \le N \le 100{,}000$;
- $1 \le x \le 1{,}000{,}000{,}000$ for all *insert(x)* and *delete(x)* operations;
- For any *insert(x)* operation it is guaranteed that *x* does not exist in the set;
- For any *delete(x)* operation it is guaranteed that *x* exists in the set;
- For any *min-xor()* operation it is guaranteed that the set has at least 2 elements;
- Time limit: 0.4 second
- Memory limit: 8 MB

Example

minxor.in	minxor.out	Explanations
10 1 24 1 17 3 1 23 1 30 3 2 17 3 2 30 3	9 6 6 15	For the first *min-xor()*, the set is {24, 17}. The minimum bitwise `xor` is 24 ^ 17 = 9. For the second *min-xor()*, the set is {24, 17, 23, 30}. The minimum bitwise `xor` is 24 ^ 30 = 6 (also 17 ^ 23 = 6). For the third *min-xor()*, the set is {24, 23, 30}. The minimum bitwise `xor` is 24 ^ 30 = 6. For the last *min-xor()*, the set is {24, 23}. The minimum bitwise `xor` is 24 ^ 23 = 15.