,

Problem A. A plus B multiply C

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	64 megabytes

The basic format of most competitive programming competition involve creating a program which would accept inputs from standard input and correctly output its result through its standard input. In this example, your task is to create a program which would accept three integer A, B, and C and output one integer D which is $A + B \times C$.

In this case, your code should be roughly equal to:

```
int A,B,C;
cin >> A >> B >> C;
cout << A + (B*C) << endl;
```

Look at the example input and output if you are confused. Your program needs to replicate the behavior of the example input and output. In another word, you program needs to be able to output the same output for the given input.

In each problem, your solution may be tested by more than one test case to make sure that it is fully correct. Some test case is hidden. It may not matter much in this case, but with harder problem, the problem author may create hidden critical test case which is designed to make your solution fail if you have bugs in your program.

Input

Three integer in a line A, B and C. $(0 \leq A, B, C \leq 1000)$

Output

One integer D which is equal to $A + B \times C$. D will not exceed 2^{31}.

Examples

standard input	standard output
2 3 4	14
0 20 20	400
10 0 1000	10

,

Problem B. Sum N numbers.

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	64 megabytes

This is another example of input format for a problem. In this case, you are given N integers. Your job is to calculate the sum of these N integers. Usually with this kind of test case where the number of input is not fixed, the first number is the integer N which is the number of input. The next N integer is the actual number you need to sum. So to solve this, you need to make a loop which run N times which input each number. See the input format for more information. However, some question does not mention how many number of input are there. Your program is expected to be clever enough to figure-out if there are no more input.

Also, usually the questions have a story. The story can add clarity to the problem, but more often it is used to waste participant's time on finding the actual problem, or distract you from the actual solution. So here is the story for this question:

Ahmad is a student in IIUM. He studies economics. He is 23 years old and lives in Mahallah Siddiq. Ahmad have a lot of money, but for some reason, his money is stored in N wallets. Ahmad wants to buy a new computer so he needs to know how much money he actually have. Given the number of wallet, and the amount of money he have for each wallet, calculate T, the total amount of money Ahmad have in Ringgit Malaysia.

Input

The first line consist of a single integer N ($0 \leq N \leq 1000$). Each of next N line consist of a single integer X_i ($0 \leq X_i \leq 1000$) which is the amount of money in the ith wallet in Ringgit Malaysia.

Output

Output a single integer T, which is the total amount of money Ahmad have in Ringgit Malaysia.

Example

standard input	standard output
4 1 23 42 2	68

,

Problem C. Big Number

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	64 megabytes

Sometimes a problem involves only things like basic `if`s and `while`s. Sometimes, the problem involves exploiting a particular pattern in the problem. Sometimes you need to know the underlying limitation of CPU to answer it. This is the later case.

In C++ and Java, (on x86 CPU) the `int` datatype is represented as 32 bit binary (even on 64-bit capable CPU). One of those bit is used for the sign. That means, the range of possible values that can be represented with an `int` is between -2^{31} to $2^{31} - 1$. (You can check this in C++ with the `INT_MIN` and `INT_MAX` built in global variable in "limits.h" header file). That is roughly equal to -10^9 to 10^9. If you assign a value outside this range, it will overflow and result in a smaller or bigger number depending on situation.

So, in some problem if we know that we will need to use number bigger than 32 bit binary, we can use the `long long int` datatype (or just `long` in Java). The `long long int` data type is represented as 64 bit binary. That means the range of representable number is between roughly -10^{18} to 10^{18}. Some competitive programmer even use `long long int` exclusively just in case.

The floating datatype, `float` and `double` have a different representation than the integer datatype. It is important to note that these two data type are not precise. Although the range of `double` is far larger which is -1.7×10^{308} to 1.7×10^{308}, not all number between the range can be represented due to a phenomena called "aliasing". That is why, it is best to avoid floating point data type if possible as some problem require exact answer, not just very good approximation.

Java users have the `BigDecimal` and `BigInteger` class. These two class can represent arbitrarily large number. This is very useful when the problem require the use of number larger than what 64 bit binary can represent. In that case, Java is highly preferred over C++. However, these two class have a very significant performance penalty over native datatype.

Now, after taking your time reading the question, here comes the actual problem. Given a number represented by N factors of that number, determine if the number can fit into a 32 signed integer or not.

Input

The first line consist of an integer N ($1 \leq N \leq 1000$). The next N lines consist of an integer A_i ($-1000 \leq A_i \leq 1000$) which is the ith factor of the number.

Output

On a single line, output "Yes" if the number can be represented with a 32 bit signed integer. Otherwise, print "No". You need to output the output EXACTLY. That means "Yes" is not "yes" or "YES".

,

Examples

standard input	standard output
3 33 2 567	Yes
4 1000 1000 1000 1000	No
8 512 512 512 512 512 512 512 512	No

Note

In the first example, the three factors are 33, 2, and 567. The actual number is 37422 which is within the range of 32 bit signed integer.

In the second example, the four factors are 1000, 1000, 1000 and 1000. The actual number is 1000000000000 which is outside the range of 32 bit signed integer.

The third example, the factors are eight 512. So the actual number is 512^8 which is 472236648286964521 3696 which is outside the range of 32 bit signed integer.

Problem D. Speed Lookup

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	64 megabytes

One of the most essential skill in competitive programming is estimating computational complexity. By 'estimating computational complexity', I mean estimating how fast can the program run. Competitive Programming problems generally have time limit within the range of 1 to 2 second. That means, your program must produce the output for each test case within the time limit, or you will get the "Time Limit Exceeded (TLE)" verdict.

Lets use this question as an example. You are given N words where N can be as high as 100 000. You are also given M query which can also be as high as 100 000. Each query consist of one word. Your task is to determine for each query if the word is included in the N words given before.

Normally you would put the N words in an array, and then for each M query, you would loop for each N item in the array to check if the query string is included. That would mean, at worst case you would be doing $100\,000 \times 100\,000 = 10^{10}$ operations. The rule of thumb in competitive programming is, if it takes more than 10^8 operations, it is too slow, like in this case. Try submitting with this algorithm, you'll get a TLE.

Instead of putting everything in an array, you can use a C++'s `set` or Java's `TreeSet`. These two container internally uses balanced binary search tree to store their items. Because of this structure, you can check if an item is in the collection in $\log_2 N$ operations where N is the number of item in the collection. Including the number of query, the total combined computational complexity is about $N \log_2 N$. So by using a set, the number of operation is now roughly equal to $100\,000 \times \log_2 100\,000 \approx 1\,600\,000$ a much lower number of operation.

Input

The first line consist of a single integer N ($1 \leq N \leq 100\,000$). The next N line consist of a single word w_i which is the ith word. The following line consist of a single integer M ($1 \leq M \leq 100\,000$). The next M line consist of a single word q_i which is the ith query.

Output

For each q_i, print "In" in its own line if the query string is included in list. If not, print "Out".

,

Example

standard input	standard output
5 ali abu bakar ella ayub 10 ali taufan el akar bakar ayub alla abu ali kenangan	In Out Out Out In In Out In In Out

Problem E. Easy Mod Pow

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

The reason why sometimes we call a "competitive programming" competition a "problem solving" competition is because sometimes (or a lot of time) the hard part is not programming, but figuring out how to solve the problem.

However, programming skill is very important in competitive programming. Knowing what is available to use from standard library can help you tremendously. For example, there is effectively no need for you to implement a sorting function in a programming competition. Both C++ and Java (and effectively every programming language) already have built in sorting function. Even if you want to sort it based on a class property or sort it based on some arbitrary condition, or reverse the direction, there are ways to utilize the built in sort function to do so.

Between C++ and Java, arguably Java have more built in functionality. Java is more cumbersome to use compared to C++, but there are cases where Java is a clear winner. For example, the BigInteger class which C++ does not have (built in). Another example is Java's GrogerianCalendar class. Java's BigInteger class also have rare functionality like calculating greatest common divisor (GCD), determining (approximately) if a number is a prime number or counting the number of on bit in the integer's binary representation.

In this problem, if you can find the library function to solve it, the problem can be solved in just one line (excluding class structures and input output). Given x and n. Find $x^n \mod 1\,000\,007$. The n can be very large, which means the actually number is huge. To simplify (or complicate?) stuff, you'll need to mod the huge number by this fixed prime number 1 000 007. Hint: $(a \mod c) \times (b \mod c) = (a \times b) \mod c$). This pattern where you have to mod the actual result with a prime number is quite common in the competitive programming world due to the nature of the mod operator.

Notice in the input format below, the n can reach up to 10^9. If we would multiply the number n times, we would need 10^9 operations which does not conform to our "less than 10^8 operations or you'll get TLE" rule. You will need to figure out a way to do so in $\log_2 n$ operations. Luckily, there is a function that does exactly that (although it does not specify how fast it run). You will need to figure out the clever $\log_2 n$ algorithm, or find the function.

In addition to that, this question have multiple test in each input. Up to 1000 test. This means, your solution will have to loop through all the test and answer them separately. This also means in estimating the number of operations required, you will need to multiply that by the maximum number of test. This also means, repeatedly multiplying the number will never work in time as the number of operations is now about 10^{12}. Multiple test are easy way for the problem author to force your solution to reach the time limit. Also, notice the output format given. You will need to output with the string format **Case #x: A** EXACTLY. That means, the uppercase 'C' is important. The space between 'e' and '#' is important. The '#' is important. The space between ':' and the answer A is important. If you miss any one of the character, your solution will be considered wrong. A common mistake is to include or exclude the '#' character when the output format does not include it or include it.

Input

The first line has a positive integer T, $(0 \le T \le 1000)$, denoting the number of test cases. This is followed by each test case per line. Each test case consist of two integers x and n $(1 \le x \le 10^6, 1 \le n \le 10^9)$.

Output

For each test case, output a line in the format **Case #x: A**, where x is the case number (starting from 1) and A is the result of $x^n \mod 1\,000\,007$.

Example

standard input	standard output
3 10 10 1 1000000 2 31	Case #1: 930007 Case #2: 1 Case #3: 468619