Problem A. 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