Problem A. 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	Yes
33	
2	
567	
4	No
1000	
1000	
1000	
1000	
8	No
512	
512	
512	
512	
512	
512	
512	
512	

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 4722366482869645213696 which is outside the range of 32 bit signed integer.