Problem 6. Caching approximations

Input file:	`input.txt`
Output file:	`output.txt`
Time limit:	2 seconds
	10 seconds (for Java)
Memory limit:	256 megabytes

In high-precision geometric modeling it is necessary to store various complex curves, such as for instance an intersection curve of two surfaces. The complexity arises because such curves are virtually impossible to represent in both precise and usable way: the representation is either approximate or is so complex that it cannot be used in many geometrical algorithms without rewriting them. This is why a mixed approach is commonly used: to retain the maximum precision, the curve is stored in the hard-to-use but precise format, but computational algorithms use a simple approximation of the curve generated with sufficient precision.

Perhaps the most convenient approximation is the Hermite cubic spline, which is easy to generate and easy to work with. Unfortunately, as approximation precision is improved, then the cubic spline takes more memory to store, takes more time to generate, and all algorithms work slower with it. The cubic spline approximation has error of fourth order, meaning that a 16-fold improvement of precision doubles the size of the approximation.

For the purpose of this problem, we shall assume that an approximation with precision ε takes M bytes of memory, where M is calculated as:

$$M = \frac{s}{\sqrt[4]{\varepsilon}}$$

Such approximation takes T CPU cycles to generate:

$$T = aM + b$$

Here, s, a and b are some given constants.

To avoid generating the same approximation for a curve many times, it is sometimes "cached", i.e. saved along with the curve for later use. In this problem, there is only one complex curve, and you must find the best way of caching its approximations, such that the given N operations are executed as fast as possible.

The operations must be executed strictly in the same order as they are given. No operation can be executed directly on a complex curve; each operation must be given an acceptable approximation of the curve instead. Each operation has a given "tolerance" δ, which defines the requirement on input approximation precision. A curve approximation generated with the precision ε is good enough for an operation with the tolerance δ only if $\varepsilon \le \delta$, otherwise it cannot be used to execute the operation.

The operation execution time is calculated using the formula:

$$T = cM + d$$

Here c and d are certain constants defined independently for each operation, and M is the size of the curve approximation used. Recall that the size of approximation is calculated from its precision ε according to the formula provided above.

Find the minimum time necessary to complete all the operations under three different caching policies:

1. **Caching off**: there is no way to save curve approximations between operations. Once an operation is complete, the used approximation is discarded, and a new approximation must be generated for the next operation.

2. **Cache one approximation**: each time a curve approximation is generated, it is saved in the cache, evicting the old approximation if it is present. The saved approximation can be used in

future operations as many times as desired until a new approximation is generated. Note that the old approximation is always evicted, even if the new approximation is less precise, i.e. you cannot generate an approximation without caching it.

3. **Unlimited caching**: all generated approximations are saved in cache in unlimited quantity. All of them are available for use in the future. To execute an operation, any of the previously generated approximations can be chosen, provided that it is precise enough for the operation of course.

In all three variants, the cache is initially empty and does not contain any approximations. Before each operation, you can generate an approximation with any desired precision $\varepsilon > 0$, or you can choose not to generate anything. The time it takes to generate approximations is added to the total operation execution time which you need to minimize.

Input

The first line of the input file contains an integer N — the number of operations to be executed ($1 \le N \le 10\,000$). The second line contains three real numbers s, a and b — constants affecting the size and the generation time of approximations ($10^{-3} \le s \le 10^3$, $10^{-4} \le a, b \le 10^4$).

The remaining N lines describe operations. Each line contains three real numbers: δ — the required precision for the operation, and c, d — constants affecting the operation execution time ($10^{-12} \le \delta \le 1$, $10^{-4} \le c, d \le 10^4$).

It is guaranteed that the tolerances δ of any two operations either are exactly equal or differ by at least 10^{-3} in relative sense. Real numbers in the input file can be represented in exponential format, e.g. `1e-10`

Output

The output file must contain three answers; each answer contains $N + 1$ lines. Between the answers, print a line of three "equals" symbols. The first answer is for the case when caching is off, the second is for the case when one approximation is cached, and the third is for the case of unlimited caching.

An answer begins with a line with one real number τ, which is the total time of generating all approximations and executing all operations. Every ith of the remaining N lines must state whether a new curve approximation is generated before the ith operation, and if so, how precise that approximation should be ($1 \le i \le N$). If there is no need to generate an approximation, print -1 into the line; if an approximation must be generated, print a real number ε, defining the precision of the new approximation.

The checking program will check the operation tolerance requirement with a **relative** precision of 10^{-12} and the total work time with the relative precision of 10^{-8}. It is recommended to print all real numbers in **exponential** format with 15 decimal places.

Examples

input.txt	output.txt
1 1.12e-1 0.25 1.37 1.2345e-3 0.57e+2 37.019	72.59645309369008839670076843792 8 1.2345e-3 === 7.259645309369008e+1 0.12345e-2 === 0.7259645309369008e+2 0.0012345
4 1 2 1 1e-4 1e-3 1 1e-12 1 1 1e-8 1e+3 1 0.0625e-8 0.1 1	103648.01 1e-4 1e-12 1e-8 0.0625e-8 === 103628 1e-12 -1 1e-8 0.0625e-8 === 103306.1 1e-08 1e-12 -1 -1

Example explanation

In the first example, there is only one operation, so the answer is the same regardless of the caching policy; in any case, you must generate the curve approximation with a precision equal to the operation tolerance. Note that the size of the approximation is not integer here, and no rounding happens. In the output file, several different ways of printing the answer are shown.

Consider the second example with "one approximation caching" policy. It makes sense to generate an approximation with the precision of 10^{-12} from the very beginning, and use it in the first two operations, because the execution time of the first operation is barely affected by the size of the approximation used, meaning that it would be cheaper to use a more precise approximation instead of generating another smaller approximation. The third operation strongly depends on the size of the approximation, so it makes sense to regenerate the approximation with the worst precision allowed. However, such approximation cannot be used for the fourth operation, so a new and a bit more precise approximation has to be generated for it.

On the other hand, the unlimited caching policy makes it reasonable to reuse the initially generated approximation with the precision of 10^{-12} for the fourth operation. Moreover, it is possible to generate the approximation with the precision 10^{-8} earlier in order to use it for the first operation too.