Piece of cake!

This morning Aunt Petunia baked a cake. The cake had the shape of a box with dimensions $A \times B \times C$ centimeters.

During the day, each of the D kids in the neighborhood stopped by for a slice of the cake. Every time a kid came for some cake, Aunt Petunia took her knife and made a single cut parallel to one of the sides of the cake. The piece she cut off the cake was always exactly 1 cm thick.

Problem specification

Given the values A, B, C, and D, compute the largest possible volume the cake could have at the end of the day.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of a single line containing the integers A, B, C, and D.

You may assume that $0 < A, B, C \leq 10^{18}$ and $0 \leq D \leq A + B + C - 3$.

In the easy input you may assume that $A, B, C \leq 1000$.

Output specification

For each test case output a single line with a single integer: the largest possible volume of the rest of the cake, after each of the kids got a slice.

You may assume that for each test case the answer will conveniently fit into a 64-bit signed integer variable.

Example

input

```
3

4 5 6 0

4 5 6 3

1 1 10 9
```

output

```
120
64
1
```

One way of optimally solving the second test case: first make two cuts parallel to the 4×5 side to obtain a $4 \times 5 \times 4$ cake, and then make a cut parallel to the current 4×4 side to obtain a cube with side 4 cm long.

Quick growth

The memory of Bob's computer contains two interesting things: an array of integers and a virus.

Each midnight the virus becomes active. It takes each array in memory and replaces it with a bunch of new arrays: one for each contiguous subarray of the original array.

For example, if today the memory contains a single array $(1, 2, 1, 3)$, tomorrow it will contain the following arrays: (1), (2), (1), (3), $(1, 2)$, $(2, 1)$, $(1, 3)$, $(1, 2, 1)$, $(2, 1, 3)$, and $(1, 2, 1, 3)$.

Problem specification

You are given the length N of Bob's original array, its contents and the number of days D.

Compute the sum of all elements of all arrays that will be in the memory of Bob's computer after D days. As this number can be huge, it is sufficient to compute the remainder it gives when divided by $10^9 + 9$.

Assume that the memory of Bob's computer is sufficiently large to accommodate all the arrays.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of two lines. The first line contains the two integers N and D. The second line contains N small non-negative integers: the contents of the original array.

In the easy data set you may assume that the total number of elements in all arrays after D days will be at most $10\,000\,000$.

In the hard data set you may assume that $N \leq 50$ and $D \leq 1000$.

Output specification

For each test case output a single line with a single integer: the sum of all elements in all arrays after D days, modulo $1,000,000,009$.

Example

input

```
3

4 1
1 2 1 3

1 10
47

2 10
1 2
```

output

```
34
47
33
```

The first test case corresponds to the example in the problem statement.

An additional challenge (just for fun)

The problem is solvable even if the constraints were, say, $N, D \leq 100\,000$. Can you see a solution that would be efficient enough to solve such a huge test case?

If you think you know such a solution, you can test it on the following input: $N = D = 100\,000$, array $= (1, 2, 3, \ldots, 99\,999, 100\,000)$. The correct output is $92\,629\,705$.

Round and round it goes

Last month, Dexter created the best infinite loop ever. However, his sister DeeDee used his computer to check her Facebook page and meanwhile she managed to erase two numbers from Dexter's program.

Here is the damaged program in Java. (Check the input file for the same code in other languages.)

```
public class R {

  static boolean magic(int x) {
    int foo = 0;
    while (x > 1)
      for (int y=2; ; ++y) {
        if (y==x) return foo<=0;
        if (magic(y) && x%y==0) { x/=y; ++foo; break; }
      }
    return true;
  }

  public static void main(String[] argv) {
    int where = ??????;
    int step = ??????;
    int best = where;
    while (step != 0) {
      where += step;
      if (where < best) { best=where; System.out.println("*"); }
      if (!magic(where)) break;
    }
  }
}
```

Problem specification

In both data sets your task is to replace the two "??????" strings by a pair of integers W and S. Note that these integers must be in a valid range for a signed 32-bit integer variable (e.g., `int` in Java). That is, your values must satisfy $-2^{31} \leq W, S \leq 2^{31} - 1$.

As the solution to the easy data set R1, submit any two integers W and S such that the program will run in an infinite loop if `where` is initialized to W and `step` to S.

As the solution to the hard data set R2, submit two integers W and S such that the program will run in an infinite loop if `where` is initialized to W and `step` to S. Additionally, out of all pairs (W, S) that satisfy the first condition, your values W and S must be such that the number of stars printed by the program is maximized. (Any such pair will be accepted.)

Input specification

For your convenience, the input file contains the program in multiple languages. All these programs are equivalent.

Output specification

Output a single line with two space-separated integers W and S.