A – Basic Vocabulary

When teaching children English, it is natural to teach them simpler words before more complicated words. For example, we teach children "cat" before "catenary", "dog" before "dogmatic", and "ant" before "antidisestablishmentarianism". Perhaps we should do the same with numbers!

Let $b > 1$ be a positive integer. The **base-b representation** of a positive integer n is the following string of length k:

$$\text{chr}(d_{k-1})\text{chr}(d_{k-2})\text{chr}(d_{k-3})\ldots\text{chr}(d_1)\text{chr}(d_0)$$

where $d_0, d_1, \ldots, d_{k-2}, d_{k-1}$ is the **unique** sequence of integers such that:

- $0 \le d_t < b$ for all t in $0, 1, 2, \ldots, k-1$;
- $d_{k-1} \neq 0$.
- The sum of $d_t \cdot b^t$ for $t = 0, 1, 2, \ldots, k-1$ is equal to n.

Here, $\text{chr}(i)$ outputs *a single symbol* that represents the non-negative integer i. For i from zero to nine, the output of $\text{chr}(i)$ would be the symbols $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, respectively, and for i from ten to fifteen, the output of $\text{chr}(i)$ would be the symbols A, B, C, D, E, F, respectively. For larger i, $\text{chr}(i)$ would output other symbols, a unique one for each non-negative integer. Basically, we just assume that we have a large-enough alphabet such that every non-negative integer can be represented by exactly one symbol.

For example, the base-16 representation of 12648430 is $C0FFEE$.

Given a positive integer n, find any two **distinct** bases b_1 and b_2 such that $1 < b_1, b_2 < n$, and the representation of n is base b_1 is a **prefix** of the representation of n in base b_2. If the task is impossible, you must say so as well.

Note: We say sequence S is a prefix of sequence T if both the following are satisfied:

- $\text{length}(S) \le \text{length}(T)$;
- The first $\text{length}(S)$ pairs of corresponding entries of S and T are equal, that is, $S[i] = T[i]$ for i from 1 to $\text{length}(S)$.

Input Format

The first line of input contains an integer t, the number of test cases. The descriptions of t test cases follow.

Each test case consists of a single line containing the integer n.

Output Format

For each test case, output a single line containing either:

- Two space-separated integers b_1 and b_2 denoting the two bases satisfying the conditions of the problem statement; or
- A single integer -1 if there are no such two bases.

If there are multiple pairs of bases (b_1, b_2) satisfying the conditions, any one will be accepted.

Constraints

- $1 \le t \le 3 \cdot 10^5$
- $1 \le n \le 10^{18}$

Sample

Input

```
3
87119
8
97480
```

Output

```
131 11
-1
153 27
```

Explanation

In the first test case, we have `n = 87119`.

- The base-`131` representation of `87119` is `5A4`.
- The base-`11` representation of `87119` is `5A4AA`.

Note that `5A4` is a prefix of `5A4AA`.

In the second test case, there are only `6` valid bases for `n = 8`, and one can show that none of the base representations of `8` in any of these bases is a prefix of any other.

In the first test case, we have `n = 97480`.

- The base-`153` representation of `97480` is `4@#`.
- The base-`27` representation of `97480` is `4@#A`.

where `@` here represents `chr(25)` and `#` here represents `chr(19)`. Note that `4@#` is a prefix of `4@#A`.

B – Bop to the Top

Sharpay and Ryan are aspiring Broadway actors. Both will do anything it takes to climb the ladder of success.

For example, consider this ladder right here—well, actually, it's more like a staircase. The "ladder" consists of n steps arranged in a row, whose heights are $h_1, h_2, h_3, \ldots, h_n$ from left to right. To them, a ladder is *fabulosa* if the height of each non-boundary step is the *average* of the heights of the steps to its left and right. Formally, $h_i = (h_{i-1} + h_{i+1}) / 2$ should hold for every i such that $1 < i < n$.

In order to make their ladder fabulosa, they have two kinds of operations:

- **Jump and Hop**: Choose **two** consecutive steps and make each of them $+1$ unit higher.
- **Bop Bop Bop**: Choose **three** consecutive steps and make each of them $+1$ unit higher.

Is it possible to make their ladder fabulosa using only these two operations?

Note that the minimum-height step doesn't have to be the leftmost one.

Input Format

The first line of input contains an integer t, the number of test cases. The descriptions of t test cases follow.

Each test case consists of two lines. The first line contains the integer n. The second line contains n integers h_1, h_2, ..., h_n.

Output Format

For each test case, output a single line containing either YES or NO depending on whether the task is possible or not.

Note: The strings in the output are **case-sensitive**.

Constraints

- $1 \le t \le 25000$
- $2 \le n \le 10^5$
- The sum of the ns in a single test file is $\le 10^5$
- $1 \le h_i \le 10^5$

Sample

Input

```
5
3
6 6 6
3
6 9 12
3
12 9 6
6
2 3 5 7 11 13
8
2 3 5 7 11 13 17 19
```

Output

```
YES
YES
```

```
YES
YES
YES
```

Explanation

The ladders in the first three test cases are already fabulosa, so they can be made fabulosa with zero moves.

For the fourth test case, the heights of the `n = 6` steps are: `[2, 3, 5, 7, 11, 13]`. It can be made fabulosa as follows:

- **Bop Bop Bop** from steps 1 thru 3. The heights become `[3, 4, 6, 7, 11, 13]`.
- **Jump and Hop** from steps 1 and 2. The heights become `[4, 5, 6, 7, 11, 13]`.
- **Jump and Hop** from steps 3 and 4. The heights become `[4, 5, 7, 8, 11, 13]`.
- **Jump and Hop** from steps 3 and 4. The heights become `[4, 5, 8, 9, 11, 13]`.
- **Jump and Hop** from steps 3 and 4. The heights become `[4, 5, 9, 10, 11, 13]`.
- **Jump and Hop** from steps 5 and 6. The heights become `[4, 5, 9, 10, 12, 14]`.
- **Jump and Hop** from steps 1 and 2. The heights become `[5, 6, 9, 10, 12, 14]`.
- **Bop Bop Bop** from steps 4 thru 6. The heights become `[5, 6, 9, 11, 13, 15]`.
- **Jump and Hop** from steps 5 and 6. The heights become `[5, 6, 9, 11, 14, 16]`.
- **Jump and Hop** from steps 1 and 2. The heights become `[6, 7, 9, 11, 14, 16]`.
- **Bop Bop Bop** from steps 2 thru 4. The heights become `[6, 8, 10, 12, 14, 16]`.

Note that the ladder is now fabulosa.

C – Fiendish Five

The latest Spiderguy movies are a multiversal celebration of the character's long and cherished history. In it, five Spiderguys from across the franchise's history (Takuya, Tobey, Andrew, Tom, and Miles) must band together to defeat the **Fiendish Five** (The Monster, The Goblin, The Electric, The Mystery, and The King), their five most notorious villains who have teamed up from across the different dimensions to defeat Spiderguy once and for all.

Each of the Spiderguys has a pre-existing **screentime value**, represented by the five integers s_1, s_2, s_3, s_4, s_5 (for the five respective Spiderguys). Our hope is that we can use the new movies to make all of these screentime values equal.

When a new movie is to be developed, the creator first decides on a nonnegative integer k, representing that movie's length. Then, each Spiderguy fights a different member of the Fiendish Five, with some members giving more screentime than others:

1. Whoever fights The Monster or The Electric receives $+k$ screentime;
2. Whoever fights The Mystery receives $+2k$ screentime;
3. Whoever fights The Goblin or The King receives $+3k$ screentime;

Note that different movies can have different values k, and that who fights whom can differ from movie to movie.

Is it possible to make all Spiderguys' screentime values equal? If so, please provide a series of movies that achieves this goal, and do so using the fewest number of new movies.

Input Format

The first line of input contains t, the number of test cases. The descriptions of t test cases follow.

Each test case consists of a single line containing five space-separated integers s_1, s_2, s_3, s_4, s_5.

Output Format

For each test case, first print a line containing either:

- `YES` if it's possible to make all Spiderguys' screentime values equal; or
- `NO` otherwise.

In addition, if you printed `YES`, output another line containing an integer m, the fewest number of movies needed to achieve this goal. Then print m more lines, each of which describes a movie and consists of the following two things, separated by a space:

- A string with exactly `5` characters. This string must be an anagram of `MEmGK`.
- The integer k for that movie. This k must satisfy $0 \le k \le 10^{16}$.

The meaning of the string is as follows: The ith character denotes the Fiendish Five member that the ith Spiderguy will fight. The letters correspond to the members as follows:

- `M` is for The Monster;
- `E` is for The Electric;
- `m` is for The Mystery;
- `G` is for The Goblin;
- `K` is for The King.

If there are multiple valid series of movies with minimal m, any one will be accepted.

Note: The strings in the output are **case-sensitive**.

Constraints

- $1 \le t \le 50000$
- $1 \le s_i \le 10^{15}$
- The input will be such that the sum of all m in the output is ≤ 400000

Sample

Input

```
3
1000 1000 1000 1000 1000
1 1 1 1 100000000000
200 600 400 100 200
```

Output

```
YES
0
NO
YES
2
KEMmG 300
MmKGE 200
```

Explanation

For the first test case, all screentimes are already equal, so 0 movies are needed.

For the second test case, it can be shown that it is impossible to make all screentimes equal no matter how many extra movies are made, so the answer is NO.

For the third test case, it can be shown that at least 2 movies are needed, and the sample output describes a way to make all screentimes equal with just two movies. Specifically:

- Initially, the screentimes are $[s_1, s_2, s_3, s_4, s_5] = [200, 600, 400, 100, 200]$.
- After the first movie (which has $k = 300$), the screentimes become $[s_1, s_2, s_3, s_4, s_5] = [1100, 900, 700, 700, 1100]$.
- After the second movie (which has $k = 200$), the screentimes become $[s_1, s_2, s_3, s_4, s_5] = [1300, 1300, 1300, 1300, 1300]$.

D – Modular Inverse Square Sum

Miss Missouri needs your help in solving a number theory problem!

Let $I(n, k)$ be the modular inverse of k^2 modulo n. If the modular inverse doesn't exist, $I(n, k)$ is defined to be 0. (See below for the definition of "modular inverse".)

Let $S(n)$ be the sum of $I(n, k)$ for $k = 0, 1, \ldots, n-1$.

Given l and m, find the sum of $(S(n) \bmod n)$ for all integers n such that $1 \le n \le l$ and n divides $m!$ (the factorial of m). Find this sum modulo 998244353.

Note: A **modular inverse** of an integer x modulo n is an integer y such that $0 \le y < n$ and $xy \equiv 1$ modulo n. It can be shown that if x has a modular inverse modulo n, then it is unique.

Input Format

The input consists of one line containing two space-separated integers l and m.

Output Format

Output a single line containing a single integer denoting the answer.

Constraints

- $10 \le l \le 10^{11}$
- $5 \le m \le 10^6$

Sample

Input

```
10 5
```

Output

```
11
```

E – Parker Rectangle

Matt is giving it a go again! This time, he's looking for a grid of nonnegative integers whose row and column sums are "close enough".

More precisely, we say an $r \times c$ grid of integers is a **Parker Rectangle** of order n iff it satisfies the following properties:

- $r > 0$ and $c > 0$
- $r + c = n$
- The elements of the grid are *distinct* nonnegative integers, and they are at most 10^9.
- Let R_i be the sum of the elements in row i, let C_j be the sum of the elements in column j, and let S be the list $[R_1, R_2, \ldots, R_r, C_1, C_2, \ldots, C_c]$. (Thus, list S has $r + c$ elements.) Then every pair of elements of S have an *absolute difference* of at most $\lfloor n/2 \rfloor$.

Matt thinks that a difference of $\lfloor n/2 \rfloor$ "ought to be small enough", and that anything smaller would just make it much harder to find valid rectangles.

Given n, find a Parker Rectangle of order n, or determine if there aren't any. If there are multiple answers, you may print any one.

Input Format

The first line of input contains a single integer t denoting the number of test cases. The descriptions of t test cases follow.

Each test case consists of a single line containing a single integer n.

Output Format

For each test case:

- If there exists no Parker Rectangle of order n, print a single line containing -1.
- Otherwise, first print a single line containing two integers r and c ($r > 0$, $c > 0$, $r + c = n$) denoting the dimensions of the Parker Rectangle.

Then, print r lines, each containing c integers. These denote the elements of the grid. The j^{th} integer of the i^{th} line must be the element present in the i^{th} row and j^{th} column of the grid. Each element must be a nonnegative integer at most 10^9.

Constraints

- $1 \le t \le 150$
- $2 \le n \le 1024$
- The sum of n^2 across all test cases is at most 1048576.

Sample

Input

```
2
6
15
```

Output

```
3 3
1 7 6
```

```
9 5 2
4 3 8
-1
```

Explanation

For the first test case:

- $R_1 = 1 + 7 + 6 = 14$
- $R_2 = 9 + 5 + 2 = 16$
- $R_3 = 4 + 3 + 8 = 15$
- $C_1 = 1 + 9 + 4 = 14$
- $C_2 = 7 + 5 + 3 = 15$
- $C_3 = 6 + 2 + 8 = 16$

Thus, $S = [14, 16, 15, 14, 15, 16]$. Note that the absolute difference between any two elements of S is $\leq \lfloor 6/2 \rfloor = 3$, so the output is a valid Parker Rectangle of order 6.

For the second test case, it can be shown that there exists no Parker Rectangle of order 15.

F – Seating Sweethearts

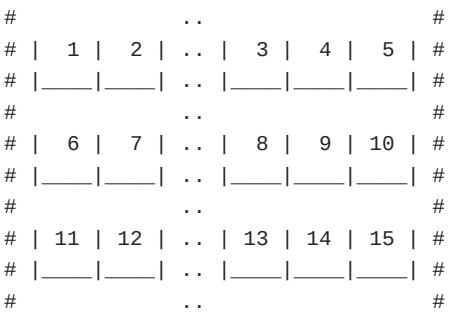
Reserving consecutive/adjacent seats for a couple on a train can be challenging. This problem asks you to help a couple identify whether the seat numbers they reserved are together or not.

Trains typically consist of multiple similar coaches/buggies, each comprising multiple rows. Each row contains two groups of continuous/together seats, with a gap between the two groups to allow passengers to walk through the coach. The first group contains two seats, and the second group contains three seats.

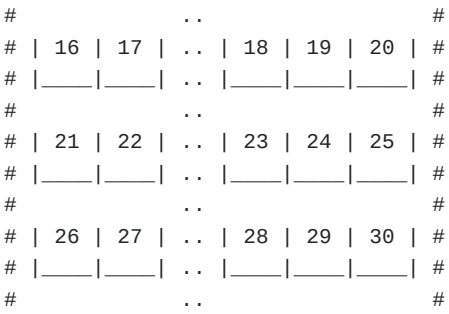
Let's say the train has `n` coaches, and each coach contains `r` rows. Then, the seats in the train are numbered "consecutively", as illustrated with an example below.

Here's an example configuration for `n = 2` and `r = 3`:

First Coach:



Second Coach:



Here, the dots denote the space between the two groups to allow the passengers to walk through the coach.

We can see that there are `5·n·r` seats, and that the seat numbers will go from `1` to `5·n·r`. The seats are numbered in order from the first to the last coach, and for each coach, from the first to the last row.

We say two seats are **together** if they share an armrest between them. For example, seats `1` and `2` are together, seats `3` and `4` are together, and seats `4` and `5` are together as well. However, seats `2` and `3` are not together, seats `3` and `5` are not together, seats `1` and `10` are not together, and seats `10` and `11` are not together.

Loid and Yor are a couple, and have been allocated two seat numbers, s_1 and s_2, where $s_1 \neq s_2$. Are these two seats together or not?

Input Format

The first line of input contains a single integer `t` denoting the number of test cases. The descriptions of `t` test cases follow.

Each test case consists of a single line containing four space-separated integers n, r, s_1 and s_2 where n denotes the number of coaches, r denotes the number of rows in a single coach, s_1 denotes the first seat number, and s_2 denotes the second seat number.

Output Format

For each test case, output a single line containing `yes` or `no`, according to whether Loid and Yor are seated together or not.

Note: The strings in the output are **case-sensitive**.

Constraints

- $1 \le t \le 10^5$
- $1 \le n, r \le 100$
- $1 \le s_1, s_2 \le 5 \cdot n \cdot r$
- $s_1 \ne s_2$

Sample

Input

```
7
2 3 1 2
2 3 3 4
2 3 4 5
2 3 2 3
2 3 3 5
2 3 1 10
2 3 10 11
```

Output

```
yes
yes
yes
no
no
no
no
```

Explanation

The sample test case is already explained in the problem statement.

G – Skill Issue

You are the CTO of X Inc., and its CEO, Melon Husk, has assigned you to conduct training camp to "level up" the team, while keeping its diversity.

X Inc. has n employees numbered 1 to n. Employee i has skill level s_i. For "diversity" purposes, all s_i are currently distinct.

There will be daily training in the training camp. Every day, one employee is chosen for training, and at the end of the day, their skill level increases by 1. It is up to you to choose the number of days of the training camp (including none), as well as which employee to train for each day. However, Melon Husk wants the following requirement to be satisfied by the end of the training camp:

For every i from 1 to n, there must be at least one employee with skill level t_i.

Note that the employee with skill level t_i does not need to be employee i; it can be another employee. Also, note that all t_i values are distinct, for "diversity" purposes again.

Note that some employees may have equal skill levels on some intermediate days, but by the end of the camp, the skill levels will be distinct (because it will be a permutation of $t_1, t_2, \ldots, t_n$).

What is the number of possible ways in which the training camp can be organized? We say two ways of organizing the training camp are the same iff the following conditions are satisfied:

- The two training camps last the same number of days, say d.
- For each i from 1 to d, the employee chosen for training on day i on both camps are the same.

Print the answer modulo the prime number p.

Input Format

The input consists of three lines.

- The first line contains two space-separated integers, n and p.
- The second line contains n space-separated integers $s_1, s_2, \ldots, s_n$.
- The third line contains n space-separated integers $t_1, t_2, \ldots, t_n$.

Output Format

Print a single line containing the number of ways to organize the training camp, modulo p.

Constraints

- $1 \le n, s_i, t_i \le 33$
- $10^8 < p < 10^9$
- p is prime
- The s_i values are all distinct.
- The t_i values are all distinct.

Sample

Input

```
2 998244353
1 2
3 2
```

Output

```
3
```

Explanation

The following are the three ways to organize the training camp:

1. The training camp lasts two days.
    - Train employee 1 on the first day.
    - Train employee 2 on the second day.
2. The training camp lasts two days.
    - Train employee 1 on the first day.
    - Train employee 1 on the second day.
3. The training camp lasts two days.
    - Train employee 2 on the first day.
    - Train employee 1 on the second day.

H – The Cabins in the Woods

You are in the middle of the woods. You don't know how you got here. You don't even remember your name. All you remember is that a few moments ago, you were in a dark room, sorting VHS tapes, and while watching one of the tapes, you found your consciousness transported into these woods.

After walking a bit, you found yourself in front of n cabins. These cabins are mysteriously linked to each other. Specifically, if you enter a cabin and then exit, you might find yourself in a different cabin than the one you started with!

More specifically, if you enter cabin x, then you'll exit at cabin C_x. (We number the cabins 1 to n.) Note that this teleportation is "one-way"—in particular, if you enter cabin C_x, you don't necessarily exit at cabin x. However, it is guaranteed that the C_x values are *distinct*. This is equivalent to saying that for each cabin y, there is at most one cabin you can enter such that you exit at cabin y.

We say two cabins x and y are **separated** if you cannot get from cabin x to cabin y or vice versa by *only* entering and exiting cabins (and without, say, just walking directly from cabin x to cabin y). We also define the **fracture value** of the cabins to be the maximum number of cabins that are *pairwise* separated.

For example, suppose $n = 6$, and $[C_1, C_2, C_3, C_4, C_5, C_6] = [4, 5, 1, 3, 2, 6]$. Then cabins 1 and 2 are separated, but cabins 1 and 3 are *not* separated, and cabins 1 and 4 are *not* separated as well. Also, note that cabins 3, 5 and 6 are pairwise separated, but it can be shown that there is no set of four cabins such that every pair is separated, so the fracture value is 3.

You spend q days in the woods. On the ith day, after a bit of playing around, you noticed that some cabins have changed. Specifically, you noticed that on the ith day, the destinations of cabins x_i and y_i have *swapped*! In other words, the values C_{x_i} and C_{y_i} have swapped.

For each day, you wonder: what is the fracture value of the cabins at the end of that day?

Note that the numbers x_i and y_i are computed in an unusual way from the input:

- You will be given two sequences $u_1, \ldots, u_q$ and $v_1, \ldots, v_q$ as input.
- For $i > 0$, let a_i be the answer for day i, and let $a_0 = 0$.
- For each i, the values x_i and y_i are computed as $x_i = a_{i-1} \oplus u_i$ and $y_i = a_{i-1} \oplus v_i$, where $\oplus$ denotes the bitwise XOR.

Input Format

The input consists of four lines.

- The first line contains contains two space-separated integers n and q.
- The second line consists of n space-separated integers $C_1, C_2, \ldots C_n$.
- The third line consists of q space-separated integers $u_1, u_2, \ldots u_q$.
- The fourth line consists of q space-separated integers $v_1, v_2, \ldots v_q$.

Output Format

Output a single line containing q space-separated integers, the ith of which should be a_i, the fracture value of the cabins at the end of day i.

Constraints

- $1 \le n, q \le 100000$
- $u_i \ge 0$
- $v_i \ge 0$

- $1 \le C_i, x_i, y_i \le n$
- $x_i \ne y_i$
- All C_i values are distinct.

Sample

Input

```
3 3
1 2 3
1 0 2
2 1 0
```

Output

```
2 1 2
```

Explanation

1. We have $x_1 = 1$ and $y_1 = 2$, so after the first day, we have $[C_1, C_2, C_3] = [2, 1, 3]$. The fracture value is 2.
2. We have $x_2 = 2$ and $y_2 = 3$, so after the second day, we have $[C_1, C_2, C_3] = [2, 3, 1]$. The fracture value is 1.
3. We have $x_3 = 3$ and $y_3 = 1$, so after the third day, we have $[C_1, C_2, C_3] = [1, 3, 2]$. The fracture value is 2.

I – Weightiest Watermelon

In the mystical village of Westasia, there's a tradition during the weekly Wookiee Festival where villagers gather to play a game called Watermelon Warfare.

In Watermelon Warfare, the players are given n watermelons, and the players work together to find the heaviest watermelon. Initially, the weights of the watermelons are unknown, but it is guaranteed that the weights are distinct. The players then weigh the watermelons against each other with a weighing scale to gather information about the weights.

Wesley and Wendy would like to analyze this game more deeply.

They figured that at any point in the game, the partial information can be encoded as follows. First, number the watermelons 1 to n. Then, for each pair (i, j) such that $1 \le i < j \le n$, let $\texttt{cmp}_{i,j}$ be the information about the comparison of watermelons i and j. Specifically, it is one of:

- $<$ if watermelon i is strictly lighter than watermelon j;
- $>$ if watermelon i is strictly heavier than watermelon j;
- $?$ if we don't have information either way.

Thus, there are 3 possible values of $\texttt{cmp}_{i,j}$, and there are $3^{n(n-1)/2}$ possible ways to assign the values of *all* $\texttt{cmp}_{i,j}$.

For each such assignment A, let $W(A)$ be the number of watermelons i such that it is consistent for watermelon i to be the heaviest watermelon given the assignment A. Note that if A is "inconsistent", then $W(A) = 0$.

Help Wesley and Wendy find the sum of $W(A)$ across all $3^{n(n-1)/2}$ assignments! However, because this number may be large, you only need to find it modulo a given prime number p.

Input Format

The first line contains a single integer t, the number of test cases. The descriptions of t test cases follow.

Each test case consists of one line containing two space-separated integers n and p.

Output Format

For each test case, print a single line containing the sum of $W(A)$ across all assignments A, modulo p.

Constraints

- $1 \le t \le 3$
- $1 \le n \le 2000$
- $10^8 < p < 10^9$
- p is prime

Sample

Input

```
3
3 100000007
5 100000007
7 100000007
```

Output

```
36
43440
94113232
```

Explanation

For $n = 3$, there are $3^{3(3-1)/2} = 27$ possible assignments A. The sum of $W(A)$ across all these assignments is 36 (modulo 100000007).