Codeforces Round 1059 (Div. 3)

A. Beautiful Average

1 second, 256 megabytes

You are given an array a of length n.

Your task is to find the maximum possible average value of any subarray* of the array a.

Formally, for any indices l, r such that $1 \le l \le r \le n$, define the average of the subarray $a_l, a_{l+1}, \ldots, a_r$ as the sum of elements divided by the number of elements or:

$$\operatorname{avg}(l, r) = \frac{1}{r - l + 1} \sum_{i=l}^{r} a_i$$

Output the maximum value of $\operatorname{avg}(l, r)$ over all choices of l, r.

*An array b is a subarray of an array a if b can be obtained from a by deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end. In particular, an array is a subarray of itself.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each testcase contains a single integer n ($1 \le n \le 10$) — the length of the array a.

The second line of each testcase contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10$) — the elements of the array.

Output

For each testcase, output a single integer — the maximum average of any subarray of the given array.

It can be shown that the answer is always an integer.

input
3 4 3 3 3 3 5 7 1 6 9 9 5 3 4 4 4 3

output
3 9 4

B. Beautiful String

1 second, 256 megabytes

You are given a binary* string s of length n.

Your task is to find any subsequence† p of s such that:

- The subsequence p is **non-decreasing**. That is, each character in p is not greater than the next one.
- Let x denote the string obtained by *removing all characters of p from s*, while preserving the order of the remaining characters. Then x must be a **palindrome**‡.

You only need to output any valid subsequence p that satisfies both conditions. If no such subsequence exists, output -1.

Note that an empty string is both non-decreasing and a palindrome.

*A binary string is a string consisting of characters '0' and '1'.

†A *subsequence* of a string $s = s_1 s_2 \ldots s_n$ is a sequence $p = s_{i_1} s_{i_2} \ldots s_{i_k}$ such that $1 \le i_1 < i_2 < \ldots < i_k \le n$. The characters are selected in order, but not necessarily contiguously. Note that an empty string is a subsequence of any string.

‡A string $t = t_1 t_2 \ldots t_m$ is a *palindrome* if $t_i = t_{m-i+1}$ for all $1 \le i \le m$. In other words, the string reads the same forward and backward.

Input

The first line contains a single integer t ($1 \le t \le 3000$) — the number of test cases.

The first line of each test case contains a single integer n ($1 \le n \le 10$) — the length of the string.

The second line contains a binary string s of length n.

Output

If a solution exists:

- On the first line, print a single integer k ($0 \le k \le n$) — the length of the subsequence p.
- On the second line, print k distinct integers $i_1, i_2, \ldots, i_k$ ($1 \le i_1 < i_2 < \cdots < i_k \le n$) — the indices of the characters in s that form p (in order as they appear in s).

Otherwise, print a single line containing -1.

input
5 3 010 3 001 5 00111 8 11010011 6 100101

output
0 2 2 3 5 1 2 3 4 5 2 3 4 2 5 6

In the first test case, we remove an empty string, resulting in $x = \texttt{010}$, which is a palindrome.

In the second test case, we remove $p = \texttt{01}$ (indices 2, 3), resulting in $x = \texttt{0}$, which is a palindrome.

In the third test case, we remove $p = \texttt{00111}$ (indices 1 to 5), resulting in an empty string, which is trivially a palindrome.

In the fourth test case, we remove $p = \texttt{01}$ (indices 3, 4), resulting in $x = \texttt{110011}$, which is a palindrome.

In the fifth test case, we remove $p = \texttt{01}$ (indices 5, 6), resulting in $x = \texttt{1001}$, which is a palindrome.

C. Beautiful XOR

2 seconds, 256 megabytes

You are given two integers a and b. You are allowed to perform the following operation any number of times (including zero):

- choose any integer x such that $0 \le x \le a$,
- set $a := a \oplus x$. Here, $\oplus$ represents the bitwise XOR operator.

After performing a sequence of operations, you want the value of a to become exactly b.

Find a sequence of at most 100 operations (values of x used in each operation) that transforms a into b, or report that it is impossible.

Note that you are not required to find the minimum number of operations, but any valid sequence of at most 100 operations.

Input

The first line of input contains a single integer t ($1 \le t \le 1000$) — the number of test cases.

Each test case contains two integers a and b ($1 \le a, b \le 10^9$).

Output

For each test case, if it is impossible to obtain b from a using the allowed operations, print a single line containing -1.

Otherwise, on the first line print a single integer k ($0 \le k \le 100$) — the number of operations. On the second line print k integers ($x_1, x_2, \ldots, x_k$) — the chosen values of x in the order you apply them.

If there are multiple valid sequences, you may print any one of them.

input
6 9 6 13 13 292 929 405 400 998 244 244 353

output
2 7 8 0 -1 1 5 2 25 779 -1

For the first test case,

- choose $x = 7$, now a becomes equal to $9 \oplus 7 = 14$.
- choose $x = 8$, now a becomes equal to $14 \oplus 8 = 6$.

Thus, we can make $a = b$.
For the fourth test case, choosing $x = 5$ makes $a = b$.

D. Beautiful Permutation

2 seconds, 256 megabytes

This is an interactive problem.

There is a *permutation*[*] p of length n.

Someone secretly chose two integers l, r ($1 \le l \le r \le n$) and modified the permutation in the following way:

- For every index i such that $l \le i \le r$, set $p_i := p_i + 1$.

Let a denote the resulting array obtained by modifying the permutation.

You are given an integer n denoting the length of the permutation p.

In one query, you are allowed to choose two integers l, r ($1 \le l \le r \le n$) and ask for the sum of the subarray either of the original permutation $p[l \ldots r]$ or of the modified array $a[l \ldots r]$. The answer to such a query will be the corresponding integer sum.

Your task is to find the pair (l, r) that was chosen to obtain a in no more than 40 queries.

[*] A permutation of length n is an array consisting of n distinct integers from 1 to n in any order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (the number 2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$, but the array contains 4).

Input

The first line of input contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

Each test case contains a single integer n ($1 \le n \le 2 \cdot 10^4$) — the length of the permutation.

It is guaranteed that the sum of n over all the test cases does not exceed $2 \cdot 10^4$.

Interaction

The interaction for each test case begins by reading the integer n.

You can ask two types of queries.

- Print "`1 l r`" ($1 \le l \le r \le n$).
  In response, you should read a line containing a single integer x — the sum of the subarray of the original permutation. (Formally, $x = p_l + p_{l+1} + \cdots + p_r$).
- Print "`2 l r`" ($1 \le l \le r \le n$).
  In response, you should read a line containing a single integer y — the sum of the subarray of the modified array. (Formally, $y = a_l + a_{l+1} + \cdots + a_r$).

The permutation p and the chosen integers l, r are fixed beforehand and can't be changed during the time of interaction.

You can output the final answer by printing "`! l r`", where l, r denote the integers that were chosen to obtain a. After printing the answer, your program should proceed to the next test case or terminate if there are no more.

You can ask no more than 40 queries per testcase. **Printing the answer doesn't count as a query.** If your program performs more than 40 queries for one test case or makes an invalid query, you may receive a `Wrong Answer` verdict.

After printing a query, do not forget to output the end of line and flush[*] the output. Otherwise, you will get `Idleness limit exceeded`.

Hacks

To make a hack, use the following test format.

The first line should contain a single integer t ($1 \le t \le 10^4$) — the number of testcases.

The first line of each test case should contain a single integer n ($1 \le n \le 2 \cdot 10^4$) — the length of the permutation p.

The second line of each test case should contain n integers p_i ($1 \le p_i \le n$) — denoting the permutation p.

The third line of each test case should contain two space-separated integers l, r ($1 \le l \le r \le n$) — the chosen integers.

For example, the following is the hack format of the example test:

```
2
3
3 1 2
2 2
4
2 1 3 4
2 4
```

[*] To flush, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages

input

```
2

3

4

5

4

8

8

7
```

output

```
1 1 2

2 1 2

! 2 2

1 2 4

2 1 3

2 3 4

! 2 4
```

For the first testcase, $p = [3, 1, 2]$ and $l = 2$, $r = 2$. Hence, the modified array a will be equal to $[3, 2, 2]$.

So, querying "1 1 2" gives $p_1 + p_2 = 3 + 1 = 4$. And querying "2 1 2" gives $a_1 + a_2 = 3 + 2 = 5$.

For the second testcase, $p = [2, 1, 3, 4]$ and $l = 2$, $r = 4$.

Note that the queries shown in the sample test are only for demonstration purposes, and they may not correspond to any optimal solution.

E. Beautiful Palindromes

2 seconds, 256 megabytes

We call an array $[b_1, b_2, \ldots, b_m]$ of length m *palindromic* if the following condition holds:

- $b_i = b_{m-i+1}$ for all $1 \le i \le m$

In other words, an array is palindromic if it reads the same forward and backward.

You are given an array $[a_1, a_2, \ldots, a_n]$ of n integers where $1 \le a_i \le n$ and an integer k.

You are required to perform the following operation **exactly** k times:

- choose an integer x such that $1 \le x \le n$,
- append x to the end of the array a.

Your goal is to perform these k operations in such a way that the number of palindromic subarrays* in the resulting array is minimized.

Output the k integers you chose for each operation, in the order they were appended.

*An array b is a subarray of an array a if b can be obtained from a by deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end. In particular, an array is a subarray of itself.

Input

The first line of input contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains two integers n and k ($3 \le n \le 2 \cdot 10^5, 1 \le k \le n$) — the length of the array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the elements of the array a.

It is guaranteed that the sum of n over all the test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print the k integers chosen for the append operations, in the order they were appended, such that the total number of palindromic subarrays in the resulting array is minimized.

If there are multiple answers, you may output any one of them.

input

```
5
4 1
1 3 3 4
4 2
2 2 2 2
5 1
4 1 5 2 2
6 3
1 2 3 4 5 6
5 3
3 2 5 2 3
```

output

```
2
1 3
3
3 4 1
4 1 5
```

For the first test case, if we append 2 to the end of the array, a becomes $[1, 3, 3, 4, 2]$. Now a has only 6 palindromic subarrays — $[1]$, $[3]$, $[3]$, $[4]$, $[2]$, $[3, 3]$.

F. Beautiful Intervals

2 seconds, 256 megabytes

You are given an integer n and m intervals. Each interval is of the form $[l_i, r_i]$ and satisfies $1 \le l_i \le r_i \le n$. Note that there can be duplicate intervals.

Let p be a permutation of length n containing all the integers $0, 1, 2, \ldots, n-1$ exactly once.

There is a multiset M which is initially empty.

For each interval $[l_i, r_i]$:

- consider the subarray $p[l_i \ldots r_i]$,
- compute $v_i = \operatorname{mex}^*(p[l_i \ldots r_i])$,
- insert v_i into M.

After processing all the intervals, M will be equal to $\{v_1, v_2, \ldots, v_m\}$.

Your task is to construct a permutation p of length n containing all the integers $0, 1, 2, \ldots, n-1$ exactly once such that $\operatorname{mex}(M)$ is **minimized**.

*$\operatorname{mex}(a)$ denotes the minimum excluded (MEX) of the integers in a. For example, $\operatorname{mex}([2, 2, 1]) = 0$ because 0 does not belong to the array, and $\operatorname{mex}([0, 3, 1, 2]) = 4$ because 0, 1, 2, and 3 appear in the array, but 4 does not.

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Description of each testcase follows.

The first line contains two integers n and m ($3 \le n \le 3000$, $1 \le m \le 3000$).

The next m lines each contain two space-separated integers l_i, r_i ($1 \le l_i \le r_i \le n$) each denoting an interval.

It is guaranteed that the sum of n over all test cases does not exceed 3000, and the sum of m over all test cases does not exceed 3000.

Output

For each testcase, print a permutation p of length n containing all the integers $0, 1, 2, \ldots, n-1$ exactly once such that $\operatorname{mex}(M)$ is minimized.

If there are multiple answers, you may print any one of them.

input
5 3 1 1 2 3 5 1 1 1 2 2 2 2 2 2 3 4 5 1 2 2 3 3 4 1 1 4 4 5 4 3 5 1 1 2 4 4 4 4 2 1 3 2 4

output
2 0 1 2 1 0 0 2 1 3 2 0 1 3 4 3 1 0 2

For the first testcase, if we choose to construct $p = [2, 0, 1]$, then $M = \{\text{mex}(2, 0)\} = \{1\}$. Now, $\text{mex}(M) = 0$.

For the third testcase, if we choose to construct $p = [0, 2, 1, 3]$, then $M = \{\text{mex}(0, 2), \text{mex}(2, 1), \text{mex}(1, 3), \text{mex}(0), \text{mex}(3)\} = \{1, 0, 0, 1, 0\}$. Now, $\text{mex}(M) = 2$.

For the fourth testcase, if we choose to construct $p = [2, 0, 1, 3, 4]$, then $M = \{\text{mex}(1, 3, 5), \text{mex}(2), \text{mex}(0, 1, 3), \text{mex}(4)\} = \{0, 0, 2, 0\}$. Now, $\text{mex}(M) = 1$.

For the fifth testcase, if we choose to construct $p = [3, 1, 0, 2]$, then $M = \{\text{mex}(3, 1, 0), \text{mex}(1, 0, 2)\} = \{2, 3\}$. Now, $\text{mex}(M) = 0$.

G. Beautiful Tree

2 seconds, 256 megabytes

A tree is a connected graph without cycles.

A tree is called *beautiful* if the sum of the products of the vertex labels for all its edges is a **perfect square**.

More formally, let E be the set of edges in the tree. The tree is called *beautiful* if the value

$$S = \sum_{\{u,v\} \in E} (u \cdot v)$$

is a perfect square. That is, there exists an integer x such that $S = x^2$.

You are given an integer n. Your task is to construct a beautiful tree having n vertices or report that such a tree does not exist.

Input

The first line of input contains a single integer t ($1 \le t \le 10^4$) — the number of testcases.

Each testcase contains a single integer n ($2 \le n \le 2 \cdot 10^5$).

It is guaranteed that the sum of n over all the testcases does not exceed $2 \cdot 10^5$.

Output

For each testcase, if there is no beautiful tree having n vertices, print -1.

Otherwise, print $n - 1$ lines denoting the edges of a beautiful tree having n vertices. Each line should contain two space-separated integers u, v ($1 \le u, v \le n$) representing an edge.

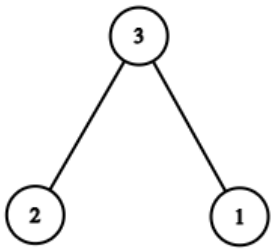
The vertices can be printed in any order within an edge, and the edges can be printed in any order.

input
3 2 3 4

output
-1 1 3 2 3 1 2 3 1 4 1

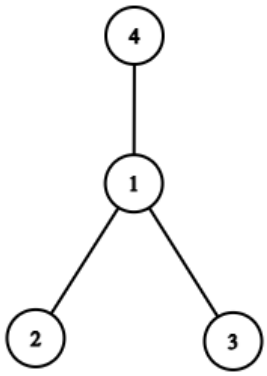
Test case 1: No beautiful tree exists with 2 vertices. Hence, print -1.

Test case 2:



$S = (2 \cdot 3) + (1 \cdot 3) = 9 = (3)^2$

Test case 3:



$S = (2 \cdot 1) + (3 \cdot 1) + (4 \cdot 1) = 9 = (3)^2$

H. Beautiful Problem

2 seconds, 256 megabytes

For an array a of length n and three integers x, l, and r $(1 \le l \le r \le n)$, define:

$$f(a, x, l, r) = \begin{cases} 0, & \text{if } (x - \min_{j=l}^{r}(a_j)) \cdot (x - \max_{j=l}^{r}(a_j))) < 0 \\ 1, & \text{if } (x - \min_{j=l}^{r}(a_j)) \cdot (x - \max_{j=l}^{r}(a_j))) \ge 0 \end{cases}$$

You are given an array a of length n ($1 \le a_i \le n$), and m intervals $[l_i, r_i]$ ($1 \le l_i \le r_i \le n$).

For each $x = 1, 2, \ldots, n$, answer the following question **independently**:

- Does there exist a rearrangement a' of a, such that for **all** $1 \le i \le m$, $f(a', x, l_i, r_i) = 1$?

Input

The first line contains a single integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Description of each testcase follows.

The first line contains two integers n and m ($2 \le n \le 2000$, $1 \le m \le 2000$).

The next line contains n space-separated integers $a_1, a_2, \cdots, a_n$ ($1 \le a_i \le n$).

The next m lines each contain two space-separated integers l_i, r_i ($1 \le l_i \le r_i \le n$), each denoting an interval.

It is guaranteed that the sum of n^2 and the sum of m^2 over all test cases does not exceed $4 \cdot 10^6$, respectively.

Output

For each test case, output a binary string s. For $x = 1, 2, \ldots, n$, $s_x = 1$ only if there exists a rearrangement a' of a, such that for **all** $1 \le i \le m$, $f(a', x, l_i, r_i) = 1$. Otherwise, $s_x = 0$.

input

```
4
4 2
1 1 3 4
1 2
2 4
3 2
1 1 3
1 2
2 3
3 1
1 1 1
1 3
9 3
4 5 9 1 1 1 2 2 3
1 6
3 7
7 9
```

output

```
1011
101
111
100100001
```

In the first test case,

- For $x = 1$, one valid rearrangement is $a' = [1, 1, 3, 4]$.
- For $x = 2$, there is no rearrangement a' of a satisfying $f(a', 2, 1, 2) = f(a', 2, 2, 4) = 1$.
- For $x = 3$, the **only** valid rearrangement is $a' = [4, 3, 1, 1]$.
- For $x = 4$, one valid rearrangement is $a' = [1, 1, 3, 4]$.