Codeforces Round #651 (Div. 2)

A. Maximum GCD

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Let's consider all integers in the range from 1 to n (inclusive).

Among all pairs of **distinct** integers in this range, find the maximum possible greatest common divisor of integers in pair. Formally, find the maximum value of $\gcd(a, b)$, where $1 \leq a < b \leq n$.

The greatest common divisor, $\gcd(a, b)$, of two positive integers a and b is the biggest integer that is a divisor of both a and b.

Input
The first line contains a single integer t ($1 \leq t \leq 100$) — the number of test cases. The description of the test cases follows.

The only line of each test case contains a single integer n ($2 \leq n \leq 10^6$).

Output
For each test case, output the maximum value of $\gcd(a, b)$ among all $1 \leq a < b \leq n$.

Example

input
2 3 5
output
1 2

Note
In the first test case, $\gcd(1, 2) = \gcd(2, 3) = \gcd(1, 3) = 1$.

In the second test case, 2 is the maximum possible value, corresponding to $\gcd(2, 4)$.

B. GCD Compression

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Ashish has an array a of consisting of $2n$ positive integers. He wants to compress a into an array b of size $n-1$. To do this, he first discards exactly 2 (any two) elements from a. He then performs the following operation until there are no elements left in a:

- Remove any two elements from a and append their sum to b.

The compressed array b has to have a special property. The greatest common divisor ($\gcd$) of all its elements should be greater than 1.

Recall that the $\gcd$ of an array of positive integers is the biggest integer that is a divisor of all integers in the array.

It can be proven that it is always possible to compress array a into an array b of size $n-1$ such that $gcd(b_1, b_2 \ldots, b_{n-1}) > 1$.

Help Ashish find a way to do so.

Input
The first line contains a single integer t ($1 \leq t \leq 10$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($2 \leq n \leq 1000$).

The second line of each test case contains $2n$ integers $a_1, a_2, \ldots, a_{2n}$ ($1 \leq a_i \leq 1000$) — the elements of the array a.

Output
For each test case, output $n-1$ lines — the operations performed to compress the array a to the array b. **The initial discard of the two elements is not an operation, you don't need to output anything about it.**

The i-th line should contain two integers, the indices (1 —based) of the two elements from the array a that are used in the i-th

operation. All $2n-2$ indices should be distinct integers from 1 to $2n$.

You don't need to output two initially discarded elements from a.

If there are multiple answers, you can find any.

Example

input
3 3 1 2 3 4 5 6 2 5 7 9 10 5 1 3 3 4 5 90 100 101 2 3
output
3 6 4 5 3 4 1 9 2 3 4 5 6 10

Note

In the first test case, $b=\{3+6,4+5\}=\{9,9\}$ and $\gcd(9,9)=9$.

In the second test case, $b=\{9+10\}=\{19\}$ and $\gcd(19)=19$.

In the third test case, $b=\{1+2,3+3,4+5,90+3\}=\{3,6,9,93\}$ and $\gcd(3,6,9,93)=3$.

C. Number Game

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Ashishgup and FastestFinger play a game.

They start with a number n and play in turns. In each turn, a player can make **any one** of the following moves:

- Divide n by any of its odd divisors greater than 1.
- Subtract 1 from n if n is greater than 1.

Divisors of a number include the number itself.

The player who is **unable to make a move** loses the game.

Ashishgup moves first. Determine the winner of the game if both of them play optimally.

Input

The first line contains a single integer t ($1 \leq t \leq 100$) — the number of test cases. The description of the test cases follows.

The only line of each test case contains a single integer — n ($1 \leq n \leq 10^9$).

Output

For each test case, print "`Ashishgup`" if he wins, and "`FastestFinger`" otherwise (without quotes).

Example

input
7 1 2 3 4 5 6 12
output
FastestFinger Ashishgup Ashishgup FastestFinger Ashishgup FastestFinger Ashishgup

Note

In the first test case, $n = 1$, Ashishgup cannot make a move. He loses.

In the second test case, $n = 2$, Ashishgup subtracts 1 on the first move. Now $n = 1$, FastestFinger cannot make a move, so he loses.

In the third test case, $n = 3$, Ashishgup divides by 3 on the first move. Now $n = 1$, FastestFinger cannot make a move, so he loses.

In the last test case, $n = 12$, Ashishgup divides it by 3. Now $n = 4$, FastestFinger is forced to subtract 1, and Ashishgup gets 3, so he wins by dividing it by 3.

D. Odd-Even Subsequence

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Ashish has an array a of size n.

A subsequence of a is defined as a sequence that can be obtained from a by deleting some elements (possibly none), without changing the order of the remaining elements.

Consider a subsequence s of a. He defines the cost of s as the minimum between:

- The maximum among all elements at odd indices of s.
- The maximum among all elements at even indices of s.

Note that the index of an element is its index in s, rather than its index in a. The positions are numbered from 1. So, the cost of s is equal to $min(max(s_1, s_3, s_5, \ldots), max(s_2, s_4, s_6, \ldots))$.

For example, the cost of $\{7, 5, 6\}$ is $min(max(7, 6), max(5)) = min(7, 5) = 5$.

Help him find the minimum cost of a subsequence of size k.

Input

The first line contains two integers n and k ($2 \leq k \leq n \leq 2 \cdot 10^5$) — the size of the array a and the size of the subsequence.

The next line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^9$) — the elements of the array a.

Output

Output a single integer — the minimum cost of a subsequence of size k.

Examples

input
4 2 1 2 3 4
output
1

input
4 3 1 2 3 4
output
2

input
5 3 5 3 4 2 6
output
2

input
6 4 5 3 50 2 4 5
output
3

Note

In the first test, consider the subsequence $s = \{1, 3\}$. Here the cost is equal to $min(max(1), max(3)) = 1$.

In the second test, consider the subsequence $s = \{1, 2, 4\}$. Here the cost is equal to $min(max(1, 4), max(2)) = 2$.

In the fourth test, consider the subsequence $s = \{3, 50, 2, 4\}$. Here the cost is equal to $min(max(3, 2), max(50, 4)) = 3$.

E. Binary Subsequence Rotation

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Naman has two binary strings s and t of length n (a binary string is a string which only consists of the characters "0" and "1"). He wants to convert s into t using the following operation as few times as possible.

In one operation, he can choose any subsequence of s and rotate it clockwise once.

For example, if $s = 1\mathbf{1}10\mathbf{100}$, he can choose a subsequence corresponding to indices (1-based) $\{2, 6, 7\}$ and rotate them clockwise. The resulting string would then be $s = 1\mathbf{0}10\mathbf{11}\mathbf{0}$.

A string a is said to be a subsequence of string b if a can be obtained from b by deleting some characters without changing the ordering of the remaining characters.

To perform a clockwise rotation on a sequence c of size k is to perform an operation which sets $c_1 := c_k, c_2 := c_1, c_3 := c_2, \ldots, c_k := c_{k-1}$ simultaneously.

Determine the minimum number of operations Naman has to perform to convert s into t or say that it is impossible.

Input

The first line contains a single integer n $(1 \leq n \leq 10^6)$ — the length of the strings.

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

The third line contains the binary string t of length n.

Output

If it is impossible to convert s to t after any number of operations, print -1.

Otherwise, print the minimum number of operations required.

Examples

input
6 010000 000001
output
1

input
10 1111100000 0000011111
output
5

input
8 10101010 01010101
output
1

input
10 1111100000 1111100001
output
-1

Note

In the first test, Naman can choose the subsequence corresponding to indices $\{2, 6\}$ and rotate it once to convert s into t.

In the second test, he can rotate the subsequence corresponding to all indices 5 times. It can be proved, that it is the minimum required number of operations.

In the last test, it is impossible to convert s into t.

F1. The Hidden Pair (Easy Version)

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Note that the only difference between the easy and hard version is the constraint on the number of queries. You can make hacks only if all versions of the problem are solved.

This is an interactive problem.

You are given a tree consisting of n nodes numbered with integers from 1 to n. Ayush and Ashish chose two secret distinct nodes in the tree. You need to find out both the nodes. You can make the following query:

- Provide a list of nodes and you will receive a node from that list whose sum of distances to both the hidden nodes is minimal (if there are multiple such nodes in the list, you will receive any one of them). You will also get the sum of distances of that node to the hidden nodes.

Recall that a tree is a connected graph without cycles. The distance between two nodes is defined as the number of edges in the simple path between them.

More formally, let's define two hidden nodes as s and f. In one query you can provide the set of nodes $\{a_1, a_2, \ldots, a_c\}$ of the tree. As a result, you will get two numbers a_i and $dist(a_i, s) + dist(a_i, f)$. The node a_i is any node from the provided set, for which the number $dist(a_i, s) + dist(a_i, f)$ is minimal.

You can ask no more than 14 queries.

Input

The first line contains a single integer t $(1 \le t \le 10)$ — the number of test cases. **Please note, how the interaction process is organized.**

The first line of each test case consists of a single integer n $(2 \le n \le 1000)$ — the number of nodes in the tree.

The next $n - 1$ lines consist of two integers u, v $(1 \le u, v \le n, u \ne v)$ — the edges of the tree.

Interaction

To ask a query print a single line:

- In the beginning print "`? c `" (without quotes) where c $(1 \le c \le n)$ denotes the number of nodes being queried, followed by c **distinct** integers in the range $[1, n]$ — the indices of nodes from the list.

For each query, you will receive two integers x, d — the node (among the queried nodes) with the minimum sum of distances to the hidden nodes and the sum of distances from that node to the hidden nodes. If the subset of nodes queried is invalid or you exceeded the number of queries then you will get $x = d = -1$. In this case, you should terminate the program immediately.

When you have guessed the hidden nodes, print a single line "`! `" (without quotes), followed by two integers in the range $[1, n]$ — the hidden nodes. You can output the hidden nodes in any order.

After this, you should read a string. If you guess the nodes correctly, you will receive the string "`Correct`". In this case, you should continue solving the remaining test cases or terminate the program, if all test cases were solved. Otherwise, you will receive the string "`Incorrect`". In this case, you should terminate the program immediately.

Guessing the hidden nodes does **not** count towards the number of queries asked.

The interactor is not adaptive. The hidden nodes do not change with queries.

Do not forget to read the string "`Correct`" / "`Incorrect`" after guessing the hidden nodes.

You need to solve each test case before receiving the input for the next test case.

The limit of 14 queries applies to each test case and not to the entire input.

After printing a query do not forget to output the end of the line and flush the output. Otherwise, you will get `Idleness limit` exceeded. To do this, use:

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

Hacks

To hack the solution, use the following test format:

The first line should contain a single integer t $(1 \le t \le 10)$ — the number of test cases. The description of the test cases follows.

The first line of each test case should contain a single integer n $(2 \le n \le 1000)$ — the number of nodes in the tree. The second line

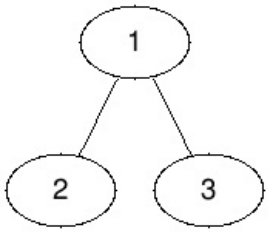
should contain two distinct integers in the range $[1, n]$ — the hidden nodes. The next $n - 1$ lines should contain two integers u, v $(1 \le u, v \le n, u \ne v)$ — the edges of the tree.

Example

input
1 3 1 2 1 3 1 1 2 3 3 1 3 1 Correct
output
? 1 1 ? 1 2 ? 1 3 ? 2 2 3 ! 1 3

Note

The tree from the first test is shown below, and the hidden nodes are 1 and 3.



F2. The Hidden Pair (Hard Version)

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Note that the only difference between the easy and hard version is the constraint on the number of queries. You can make hacks only if all versions of the problem are solved.

This is an interactive problem.

You are given a tree consisting of n nodes numbered with integers from 1 to n. Ayush and Ashish chose two secret distinct nodes in the tree. You need to find out both the nodes. You can make the following query:

- Provide a list of nodes and you will receive a node from that list whose sum of distances to both the hidden nodes is minimal (if there are multiple such nodes in the list, you will receive any one of them). You will also get the sum of distances of that node to the hidden nodes.

Recall that a tree is a connected graph without cycles. The distance between two nodes is defined as the number of edges in the simple path between them.

More formally, let's define two hidden nodes as s and f. In one query you can provide the set of nodes $\{a_1, a_2, \ldots, a_c\}$ of the tree. As a result, you will get two numbers a_i and $dist(a_i, s) + dist(a_i, f)$. The node a_i is any node from the provided set, for which the number $dist(a_i, s) + dist(a_i, f)$ is minimal.

You can ask no more than 11 queries.

Input

The first line contains a single integer t $(1 \le t \le 10)$ — the number of test cases. **Please note, how the interaction process is organized.**

The first line of each test case consists of a single integer n $(2 \le n \le 1000)$ — the number of nodes in the tree.

The next $n - 1$ lines consist of two integers u, v $(1 \le u, v \le n, u \ne v)$ — the edges of the tree.

Interaction

To ask a query print a single line:

- In the beginning print "? c " (without quotes) where c $(1 \le c \le n)$ denotes the number of nodes being queried, followed by c **distinct** integers in the range $[1, n]$ — the indices of nodes from the list.

For each query, you will receive two integers x, d — the node (among the queried nodes) with the minimum sum of distances to the hidden nodes and the sum of distances from that node to the hidden nodes. If the subset of nodes queried is invalid or you exceeded the number of queries then you will get $x = d = -1$. In this case, you should terminate the program immediately.

When you have guessed the hidden nodes, print a single line "! " (without quotes), followed by two integers in the range $[1, n]$ — the hidden nodes. You can output the hidden nodes in any order.

After this, you should read a string. If you guess the nodes correctly, you will receive the string "`Correct`". In this case, you should continue solving the remaining test cases or terminate the program, if all test cases were solved. Otherwise, you will receive the string "`Incorrect`". In this case, you should terminate the program immediately.

Guessing the hidden nodes does **not** count towards the number of queries asked.

The interactor is not adaptive. The hidden nodes do not change with queries.

Do not forget to read the string "`Correct`" / "`Incorrect`" after guessing the hidden nodes.

You need to solve each test case before receiving the input for the next test case.

The limit of 11 queries applies to each test case and not to the entire input.

After printing a query do not forget to output the end of the line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

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

Hacks

To hack the solution, use the following test format:

The first line should contain a single integer t $(1 \le t \le 10)$ — the number of test cases. The description of the test cases follows.

The first line of each test case should contain a single integer n $(2 \le n \le 1000)$ — the number of nodes in the tree. The second line should contain two distinct integers in the range $[1, n]$ — the hidden nodes. The next $n - 1$ lines should contain two integers u, v $(1 \le u, v \le n, u \ne v)$ — the edges of the tree.

Example

input
1 3 1 2 1 3 1 1 2 3 3 1 3 1 Correct
output
? 1 1 ? 1 2 ? 1 3 ? 2 2 3 ! 1 3

Note

The tree from the first test is shown below, and the hidden nodes are 1 and 3.