Problem A. Assimilation

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

An enlightened race of aliens plans to assimilate a star system, to help its inhabitants achieve perfection. They may resist, but – as you are all well aware – resistance is futile.

There are n planets in the system, inhabited by $a_1, a_2, \ldots, a_n$ people, respectively. Aliens start with k assimilation ships and are allowed to make any of the following moves:

- An *invasion* requires landing on a planet with some part of the fleet. The number of landing ships s must be greater or equal to the population m of the planet. After the invasion, these ships disappear, the planet is *conquered* and now has $m + s$ inhabitants.
- A *mobilization* creates, from a conquered planet, a number of new ships equal to the population of the planet. Every planet can be mobilized at most once.

For Aliens, invasions are easy and natural, but mobilizations turn out to be a bit tricky. Help them conquer all the planets in the system with minimal possible number of mobilizations.

Input

The first line of input contains the number of test cases z ($1 \le z \le 30$). The test cases follow, each one in the following format:

The first line of every test case contains two integers n and k ($1 \le n \le 200\,000$; $1 \le k \le 10^9$) – the number of planets, and the size of Aliens' initial fleet. The second line contains n integers $a_1, \ldots, a_n$ ($1 \le a_i \le 10^9$) – the populations of the respective planets.

The sum of n values over all test cases does not exceed 500 000.

Output

For every test case, output a single integer: the minimal number of mobilizations required to conquer all the planets. If such conquest is impossible, output -1 instead.

Example

standard input	standard output
4 3 15 6 5 26 3 15 6 5 27 2 1000000000 500123123 497000000 7 2 6 2 4 1 9 3 12	2 -1 0 4

Problem B. Little Worm

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	512 mebibytes

Little Worm is living on a tree. The tree has n vertices (and is a connected, undirected acyclic graph), and Worm occupies the whole path between the vertices a and b.

Worm would like to move to another path – the one between vertices c and d – as it is more sunny there. It is known that the paths $a \leftrightarrow b$ and $c \leftrightarrow d$ have no vertices in common.

To change its position on the tree, Worm can make some moves, which consist of entering a free vertex with Worm's either end. Formally, if Worm is currently occupying a path between x and y, it may choose a new vertex z adjacent to x, which is not on the path $x \leftrightarrow y$. Then Worm frees (stops occupying) y, taking z instead. In a similar way, Worm can choose a vertex z' adjacent to y, free x and occupy z'. After a single move Worm still occupies some path, and its length does not change.

Worm is aiming to get to the path between c and d, but being quite lazy, it doesn't plan for more than $10 \cdot n$ moves. Can you help it reach its goal within that limit?

Input

The first line of input contains the number of test cases z ($1 \le z \le 7000$). The test cases follow, each one in the following format:

The first line of a test case contains a single integer n ($4 \le n \le 100\,000$) – the number of the vertices of a tree. Each of the following $n - 1$ lines contains two integers u, v ($1 \le u \ne v \le n$), describing the endpoints of a single edge.

In the next line two integers a and b ($1 \le a \ne b \le n$) are given. These are the endpoints of the path that is Worm's starting position.

The next line contains the endpoints of the path which is Worm's goal, given as two integers c and d ($1 \le c \ne d \le n$).

The number of vertices on the path between a and b match the number of vertices on the path between c and d. You may also assume that those two paths have no common vertices.

The sum of all values of n over all test cases does not exceed 1 000 000.

Output

For every test case, if Worm cannot reach its goal in $10 \cdot n$ moves, output -1. Otherwise, output a possible sequence of Worm's moves in two lines: first consisting the number of moves q ($1 \le q \le 10 \cdot n$) and the other containing q integers $v_1, v_2, \ldots, v_q$ – the required moves. For $i = 1, 2, \ldots, q$, the value v_i should denote the vertex which is entered by Worm in the i-th move. You may output any correct sequence that moves Worm to the goal and has no more than $10 \cdot n$ moves (in particular, you do not have to minimize the number of moves). Assume that Worm is symmetrical – it can move in both directions and it can enter the goal path facing either side.

Example

standard input	standard output
3 6 1 2 1 3 1 4 4 5 4 6 2 3 5 6 15 1 2 1 6 2 3 2 4 2 5 6 7 6 8 5 9 6 10 9 11 9 12 9 13 12 14 14 15 14 13 3 6 6 1 2 1 3 2 4 4 5 5 6 4 6 3 2	-1 7 15 5 2 1 6 7 3 3 2 1 3

Note

We only consider simple paths, i.e. with no vertex appearing twice. It is a known fact that any two vertices in a tree can be connected with exactly one simple path.

Problem C. Polygon

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

You are given n segments of lengths $\ell_1, \ell_2, \ldots, \ell_n$, respectively. Determine the largest possible circumference of a convex polygon that can be constructed using these segments (in any order, and not neccessarily all of them). The polygon must be non-degenerate – in other words, its area must be positive.

Input

The first line of input contains the number of test cases z ($1 \le z \le 100\,000$). The test cases follow, each one in the following format:

The first line of a test case contains the number of segments n ($1 \le n \le 100\,000$). In the second line, there are n integers $\ell_1, \ldots, \ell_n$ ($1 \le \ell_i \le 10^9$) – the lengths of the segments.

The sum of n values over all test cases does not exceed $1\,000\,000$.

Output

For each test case, output a single integer – the largest possible circumference of a convex polygon made of given segments. If no such polygon can be constructed at all, output 0.

Example

standard input	standard output
4 6 1 2 3 4 5 6 3 9 5 14 4 5 15 4 6 2 10 11	21 0 15 0

Problem D. Frogs

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

You may think that frogs are only good for leaping and croaking, but it turns out that they are also quite proficient coders! Your task is to choose three frogs which would form the best team for OpenFrogCup.

In the frogs' favourite pond there are n stones in a row, spaced 1 meter apart from each other. On every stone, a frog sits. Stones (and frogs) are numbered $1, 2, \ldots, n$ from the leftmost to the rightmost one. The i-th frog sits on i-th stone and is described by two parameters: its leap range r_i and its programming skill s_i. The frog can reach any stone which is not farther than r_i meters (in other words, any stone with index j in $[i - r_i, i + r_i]$). Each frog is willing to jump at most once.

The team for OpenFrogCup must consist of exactly three members which can train together. This means that there must be a stone that all three frogs can jump to (allowing zero-length jumps). Determine the largest possible sum of programming skills of such a team.

The limits for the problem guarantee that there always exists at least one possible three-frog team.

Input

The first line of input contains the number of test cases z ($1 \le z \le 30$). The test cases follow, each one in the following format:

The first line of a test case contains an integer n ($3 \le n \le 200\,000$) – the number of stones (and also the frogs). Each of the following n lines contain two integers r_i, s_i ($1 \le r_i, s_i \le 200\,000$) – the range and the skill of the i-th frog, respectively.

The sum of n values over all test cases does not exceed 500 000.

Output

For every test case, output a single integer – the largest possible sum of skills of a three-frog team.

Example

standard input	standard output
3 4 1 39 2 17 4 5 1 40 3 1 10 1 20 1 30 7 5 4 4 3 3 2 2 1 3 2 4 3 5 4	62 60 11

Problem E. The Great Drone Show

Input file:	*standard input*
Output file:	*standard output*
Time limit:	30 seconds
Memory limit:	1024 mebibytes

This year's Great Drone Show is going to be a stunning success! Well, if nothing goes horribly wrong. And if everybody sticks to the plan.

The plan is worked out in every detail. At the beginning, n drones are parked on the ground. To describe their movement, we introduce standard Euclidean coordinates in three dimensions, in which the ground is the $z = 0$ plane. The starting position of the i-th drone is then described as $(x_i, y_i, 0)$.

To allow communication during the show, there are m cables between pairs of drones. The cables initially also lie on the ground, in the form of straight segments connecting some pairs of drones. It is known that from every drone there is a sequence of cables to every other drone (the cable network is connected). Moreover, to avoid tangling the cables, **no two segments cross each other** (they can only have common endpoints).

During the show a sequence of k moves will be performed. Every move consists of changing the height (i.e. the z-coordinate) of one of the drones. Each move will be performed smoothly and will start only after the previous one ends. During a move, the distance between some drones may change – fortunately, the cables can stretch to some degree. For every cable we know the maximal length it can have – if its endpoint drones go further than this value, the cable breaks.

The show organizers are prepared for some cables to break. However, some pairs of drones must remain able to communicate, directly or indirectly. Given q specific, *critical* pairs of drones, determine if communication between these pairs becomes impossible at some point during the show, and if so, determine the move which will cause the connection loss.

Input

The first line of input contains the number of test cases z ($1 \le z \le 400$). The test cases follow, each one in the following format:

The first line contains the number of drones n ($2 \le n \le 500\,000$). Each of the following n lines contains two integers x_i, y_i ($|x_i|, |y_i| \le 10^8$) – the x and y coordinates of the i-th drone. No two drones occupy the same starting location.

The next line contains an integer m ($1 \le m \le 3 \cdot n$) – the number of cables. Each of the following m lines describes a single cable, and contains three integers u, v, l ($1 \le u \ne v \le n$; $1 \le l \le 10^9$) – the numbers of connected drones and its maximal length, respectively. A pair of drones can be connected by at most one cable. Every cable's length at its starting position fits within the given length limit.

The next line contains the number of moves k ($1 \le k \le 500\,000$). Each of the following k lines contain two integers v, h ($1 \le v \le n$; $|h| \le 10^9$) – number of the moving drone and its change of height (positive it the drone raises, negative if it falls). You may assume that no drone ever falls below the ground (the z coordinates remain non-negative).

Finally, the following line contains an integer q ($1 \le q \le 500\,000$) – the number of critical pairs to be checked. In the next q lines, these pairs are described – each one contains two drone numbers u, v ($1 \le u \ne v \le n$).

The sum of n values over all test cases does not exceed $1\,000\,000$. Similarly, both the sum of k values and the sum of q values also do not exceed $1\,000\,000$.

Output

For every test case, output in separate lines q integers – the answers for each critical pair. For every such pair of drones, output the number of the first move after which the drones lost the ability to communicate. The moves are numbered starting from 1. If a critical pair remains connected during the whole show, output -1 instead.

Example

standard input	standard output
1 4 0 0 0 12 0 24 0 25 3 1 2 13 2 3 13 3 4 1 4 3 1 2 6 3 1 2 -6 4 1 2 2 3 3 4 1 4	2 -1 1 1

Problem F. Fantastic Compression

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	512 mebibytes

Franek had one job: to memorize a permutation P of the sequence $(1, 2, \ldots, n)$. This, however, proved too boring. Instead, he compressed the numbers in a new, fantastic way he devised: he took a small integer k and memorized only the sums of all connected k-length fragments of P. In other words, Franek now has a sequence $S = (S_1, S_2, \ldots, S_{n-k+1})$, where:

- $S_1 = P_1 + P_2 + \ldots + P_k$, - $S_2 = P_2 + P_3 + \ldots + P_{k+1}$, - $\ldots$ - $S_{n-k+1} = P_{n-k+1} + P_{n-k+2} + \ldots + P_n$.

The method swiftly proved not-so-fantastic, though. First, Franek discovered, to his horror, that sometimes there are several permutations which all compress to the same sequence. Also, he is not sure anymore if he remembered the compressed sequence correctly – the initial permutation may now be lost forever!

Given a compressed sequence S, help Franek find all permutations P which correspond to S.

Input

The first line of input contains the number of test cases z ($1 \le z \le 1000$). The test cases follow, each one in the following format:

The first line of a test case contains the length of the permutation n and the small integer k chosen by Franek ($2 \le n \le 25\,000; 2 \le k \le \min(n, 6)$). The second line contains $n - k + 1$ integers: the elements of the compressed sequence S ($1 \le S_i \le 1\,000\,000$).

The total length of permutations in all testcases does not exceed 250 000.

Output

For every test case, output first the number c of permutations that correspond to the given sequence S. In the next c lines, output these permutations in lexicographic order. Every permutation should be given as n integers in a single line, separated by spaces.

Assume that for the given tests, c is never greater than 1 000.

Example

standard input	standard output
2 5 3 8 10 12 5 3 10 10 10	2 1 2 5 3 4 2 1 5 4 3 0

Problem G. Bookstore

Input file:	*standard input*
Output file:	*standard output*
Time limit:	7 seconds
Memory limit:	512 mebibytes

You own a very peculiar bookstore, which sells old books, but you store all of them on a single shelf, in random order, and you do not care about the books' content. Nor do your clients – they tend to come into the store and simply ask for "all the books on that shelf, starting from *this* one and ending *here*". To be precise, every client buys some connected (and non-empty) fragment of books from the shelf.

Sometimes, though, you get more picky clients, who expect more from a book – actually, they expect it to have the right size. A picky client wants a fragment of shelf in which all the books have their height not smaller than l and not greater than h.

Given a sequence of integers – the heights of all the books on the shelf – determine the number of possible connected fragments which satisfy these requirements.

Also, we mentioned that the books are in *random* order. Formally, the input sequence was generated with the following program, for some values of $N \in \{1, 2, \ldots, 100\,000\}$ and $M = 10^q$ with $q \in \{1, 2, \ldots, 6\}$.

```
srand48(N + M);
for (int i = 0; i < N; ++i)
  a[i] = 1 + lrand48() % M;
```

You do not actually need to know how the RAND48 library works. It is enough to assume that the function `lrand48` returns 31-bit non-negative integers picked uniformly at random.

Input

The first line of input contains the number of test cases z ($1 \le z \le 5$). The test cases follow, each one in the following format:

The first line of a test case contains the number of books n and the number of picky clients k ($1 \le n \le 200\,000$, $1 \le k \le 500\,000$).

The second line contains a sequence of n positive integers not exceeding $1\,000\,000$ – the heights of all the books, from the first (leftmost) to the last (rightmost) one.

The final k lines describe the clients' requirements. The i-th of these lines contains two integers l_i, h_i ($1 \le l_i \le h_i \le 1\,000\,000$), describing a client that wants books to be not smaller than l_i and not greater than h_i.

The total number of books in all test cases does not exceed 600 000, and the total number of clients in all test cases does not exceed 1 500 000.

Output

For every client, output the number of non-empty connected fragments of the book sequence which safisty the client's requirements.

Example

standard input	standard output
2 10 3 9 9 3 2 1 9 6 9 1 7 1 13 6 6 2 9 5 1 66575 45720 67904 18764 35162 20000 80000	55 1 17 7

Problem H. Cheese Game

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

After taking part in the annual *Two-player Games and Applied Cryptography Symposium*, Alice and Bob want to relax by playing their favourite game. They have arranged n cheese slices in a row, numbered from 1 do n. As we all know, though cheese is tasty in general, some slices can be better than others – this is why the i-th slice is described by its deliciousness o_i.

Alice starts the game and the players alternate their moves. In a move, a player may eat any set of cheese slices that are still left on the board, providing that the set contains no two neighbouring slices (i.e. numbered i and $i+1$ for any $1 \le i \le n-1$). We assume that the numbers of the slices do not change, so during the game no new neighbouring pairs appear.

Of course, both players aim to maximize the total deliciousness of their eaten pieces. Assuming that they both play optimally, what is the maximal score that Alice can achive?

Input

The first line of input contains the number of test cases z ($1 \le z \le 20$). The test cases follow, each one in the following format:

The first line of a test case contains the number of cheese slices n ($1 \le n \le 100\,000$). The second line contains n integers $o_1, o_2, \ldots, o_n$ ($1 \le o_i \le 1\,000\,000$) – the values of the pieces' deliciousness.

Output

For every test case, output a single integer – the total deliciousness of the slices eaten by Alice, assuming that both players play optimally.

Example

standard input	standard output
2 3 10 10 10 4 1 2 3 4	20 7

Problem I. Harry Potter and the Palindromic Radius

Input file:	*standard input*
Output file:	*standard output*
Time limit:	25 seconds
Memory limit:	512 mebibytes

A young wizard, Henry Porter, has just received sad news – the eldest of his family, uncle Markus Radius Palindromus Black, passed away. Uncle Markus had a reputation of being a quite eccentric person, using complicated binary magic, and was also known to be very, very rich.

Black's will states that Henry should inherit his mysterious chamber of treasures. To enter and claim it, however, the young wizard must say the right password H, which is a word of length n, consisting of characters '0' and '1'. Uncle Markus did not tell Henry the password – it certainly wouldn't be his style. Instead, he computed, for every $x = 1, 2, \ldots, n$, the *palindromic radius* p_x – the largest possible integer such that the word $H[x - p_x \,..\, x + p_x]$ of length $2p_x + 1$ centered at $H[x]$ exists and is a palindrome. Henry then only received the values $p_1, \ldots, p_n$. For example, if the password was 10111010, Henry would get the sequence $(0, 1, 0, 3, 0, 1, 1, 0)$.

Henry would prefer Uncle Markus not to test his algorithmic skills while being dead, but, well, there is no one to complain. And he has good friends who can help him! Given the sequence left by Markus in his will, determine all possible passwords that correspond to it. As the will is battered and stained, it might even happen that there is no solution at all.

Input

The first line of input contains the number of test cases z ($1 \le z \le 200\,000$). The test cases follow, each one in the following format:

A test case consists of two lines. The first line contains a single integer n – the length of both the password and Black's sequence ($2 \le n \le 1\,000\,000$). The second line contains n integers $p_1, p_2, \ldots, p_n$ ($0 \le p_i \le n$) – the palindromic radii for all the characters in the password.

The sum of n values over all test cases does not exceed $5 \cdot 10^7$.

Output

For every test case, output first the number k of possible passwords. If $k > 0$, output in the next k lines all the solutions as

$$0, 1$$

-sequences. The sequences must be given in lexicographic order.

You may assume that k does not exceed 100.

Example

standard input	standard output
1 8 0 1 0 3 0 1 1 0	4 00010000 01000101 10111010 11101111

Problem J. Antennas

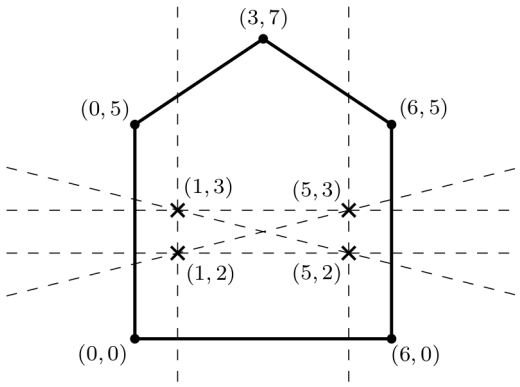
Input file:	*standard input*
Output file:	*standard output*
Time limit:	8 seconds
Memory limit:	512 mebibytes

In a secret military base, a new communication technology is being tested. For the experiment, m antennas were constructed inside.

The terrain around the base is perfectly flat, and the base, seen from above, is a convex polygon. The boundary of the polygon is a wall that protects the base from intruders, as well as blocks the radio waves from leaving the base to be possibly intercepted by foreign agents.

Unfortunately, some construction works are required in the facility, and two of the polygon's walls must be torn down. This creates a security risk: if two spies are placed outside the base in such a way that two of the antennas lie on the line between them, and there is no wall blocking this line, then the spies may listen to the communication between those two antennas.

Your goal is, for some possible scenarios of removal of two walls, to determine the number of pairs of antennas which are compromised in the way described above.



The picture above corresponds to the first case of the example input from the "Example" section. In this case, the base is a pentagon with four antennas, denoted by little crosses. All the lines between pairs of antennas are also shown.

Input

The first line of input contains the number of test cases z ($1 \le z \le 200\,000$). The test cases follow, each one in the following format:

The first line of a test case contains an integer n ($3 \le n \le 10$) – the number of vertices of the polygon. The next n lines contain two integers – the coordinates of the vertices, presented clockwise. The vertices are numbered $0, 1, \ldots, n-1$ in order in which they appear.

The next line contains an integer m ($2 \le m \le 50\,000$) – the number of antennas inside the base – and the m following lines contain the coordinates of the antennas.

The next line contains another integer q ($1 \le q \le 10$) – the number of scenarios to consider. The last q lines describe scenarios – the i-th line contains two integers a_i, b_i ($0 \le a_i < b_i \le n-1$). Such a pair denotes removing the walls a_i and b_i and requires to compute the number of distinct lines that go through some two antennas and do not cross neither the segment between the vertices a_i and a_i+1 nor the segment between b_i and $(b_i+1) \bmod n$.

All coordinates are integers whose absolute values do not exceed 10^9. In any single testcase, all points of the input are distinct and no three of them are collinear.

Every test case, including the first, is preceded by a single empty line.

The sum of all m values in all test cases does not exceed 300 000.

Output

For every testcase output, in separate lines, the answers to all given scenarios.

Example

standard input	standard output
2 5 0 0 0 5 3 7 6 5 6 0 4 1 2 1 3 5 2 5 3 3 0 3 1 4 1 2 4 -1 -1 -1 1 2 1 2 -1 2 0 0 1 0 6 0 1 0 2 0 3 1 2 1 3 2 3	4 1 0 0 1 0 0 0 0

Problem K. Ghost

Input file:	*standard input*
Output file:	*standard output*
Time limit:	15 seconds
Memory limit:	512 mebibytes

While sightseeing the Wawel castle in Krakow, your team has been trapped in an ancient chamber by the Ghost. He will not let you out, unless you answer his questions.

On the wall there are n paintings – if we treat the wall as a standard Euclidean plane, the paintings are axis-aligned rectangles. For every painting you know precisely its dimensions and starting location. In some moment – let us call it the moment 0 – Ghost starts moving the paintings, each one in its own direction and speed. As you are an observant team, for every painting you can easily guess its exact speed.

After some time, the Ghost stops the show and starts asking tough questions. Every question consists of two numbers l and r denoting some moments of the show. You must tell Ghost if there was a moment between l and r when some spot on the wall was simultaneously covered by all the paintings. If so, you must also determine the maximal possible common area for all paintings between the moments l and r.

If you want to ever leave this room, better give Ghost the right answers!

Input

The first line of input contains the number of test cases z ($1 \le z \le 4000$). The test cases follow, each one in the following format:

The first line of a test case contains the number of paintings n ($1 \le n \le 100\,000$). Each of the following n lines contains six numbers $x_1, y_1, x_2, y_2, v_x, v_y$ ($-1\,000\,000 \le x_1 < x_2 \le 1\,000\,000$; $-1\,000\,000 \le y_1 < y_2 \le 1\,000\,000$; $-1\,000\,000 \le v_x, v_y \le 1\,000\,000$), where (x_1, y_1) are the coordinates of the lower left corner of the painting, (x_2, y_2) – the upper right corner, and (v_x, v_y) is its speed vector. This means that in the moment t the lower left corner is at the spot $(x_1 + tv_x, y_1 + tv_y)$, and the upper right corner is at $(x_2 + tv_x, y_2 + tv_y)$.

The next line contains the number of Ghost's questions q ($1 \le q \le 100\,000$). Each of the following q lines contains two real numbers l, r ($0 \le l \le r \le 1\,000\,000$) given with at most 4 decimal places after the separator, meaning that Ghost asks for a closed time interval $[l, r]$.

The total number of paintings in all test cases does not exceed 1 000 000. The total number of questions in all test cases also does not exceed 1 000 000.

Output

For every Ghost's question output a single real number – the maximal area achieved by the intersection of all the paintings in the given time interval. Your answer will be considered correct if the absolute or relative error is at most 10^{-6}. In other words, if your program outputs a and the correct value is b, the answer is accepted if $\frac{|a-b|}{\max(1,b)} \le 10^{-6}$.

The intersection may be empty – in that case, your program should output 0 ($\pm 10^{-6}$).

Example

standard input	standard output
2 2 0 0 1 1 1 1 1 1 2 2 -1 -1 3 0 0 0.25 0.25 0 2 3 0 0 1 1 2 2 0 0 1 1 1 1 1 1 2 2 -1 -1 1 0 2	0.000000000 0.250000000 1.000000000 0.444444444

Problem L. Donuts (8MiB ML!)

Input file:	*standard input*
Output file:	*standard output*
Time limit:	30 seconds
Memory limit:	**8** mebibytes

Please note an exceptionally low memory limit (8MB) for this problem.

A set S of integer coordinate points in a plane is a *donut*, if there exists a midpoint (a, b) and two radii L and R (with integer a, b, L, R and non-negative radii) such that S is precisely the set of all points whose distance from (a, b) is in the interval $(L, R]$. Formally, $S = \{(x, y) \in \mathbb{Z} \times \mathbb{Z} : L < dist((x, y), (a, b)) \leq R\}$, where $dist$ denotes standard plane distance.

We begin with an empty set and add points one by one. Determine, after every added point, if the set is currently a donut.

Input

The first line of input contains the number of points n ($2 \cdot 10^7 \leq n \leq 2.5 \cdot 10^7$). Each of the next n lines describes a single added point, giving its coordinates separated by a single space. The coordinates are integers of absolute value not greater than 5000. All the given points are distinct.

Output

For every point output (in a separate line) "`TAK`", if after adding this point the set is a donut, and "`NIE`", if it isn't.

Example

standard input	standard output
12 4 1 3 2 3 0 2 3 1 0 0 1 1 2 2 -1 2 2 3 1 2 0 1 1	NIE NIE NIE NIE NIE NIE NIE TAK NIE NIE NIE TAK

Note

The example is given only for explaining the input format, and it obviously does not satisfy the $n \geq 2 \cdot 10^7$ condition (though it satisfies all the others). Your program will not be checked on the example test.