Codeforces Round #610 (Div. 2)

A. Temporarily unavailable

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

Polycarp lives on the coordinate axis Ox and travels from the point $x = a$ to $x = b$. It moves uniformly rectilinearly at a speed of one unit of distance per minute.

On the axis Ox at the point $x = c$ the base station of the mobile operator is placed. It is known that the radius of its coverage is r. Thus, if Polycarp is at a distance less than or equal to r from the point $x = c$, then he is in the network coverage area, otherwise — no. The base station can be located both on the route of Polycarp and outside it.

Print the time in minutes during which Polycarp will **not be** in the coverage area of the network, with a rectilinear uniform movement from $x = a$ to $x = b$. His speed — one unit of distance per minute.

Input

The first line contains a positive integer t ($1 \le t \le 1000$) — the number of test cases. In the following lines are written t test cases.

The description of each test case is one line, which contains four integers a, b, c and r ($-10^8 \le a, b, c \le 10^8$, $0 \le r \le 10^8$) — the coordinates of the starting and ending points of the path, the base station, and its coverage radius, respectively.

Any of the numbers a, b and c can be equal (either any pair or all three numbers). The base station can be located both on the route of Polycarp and outside it.

Output

Print t numbers — answers to given test cases in the order they are written in the test. Each answer is an integer — the number of minutes during which Polycarp will be **unavailable** during his movement.

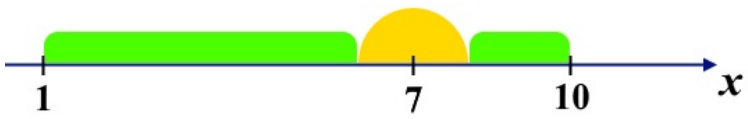
Example

input
9 1 10 7 1 3 3 3 0 8 2 10 4 8 2 10 100 -10 20 -17 2 -3 2 2 0 -3 1 2 0 2 3 2 3 -1 3 -2 2

output
7 0 4 0 30 5 4 0 3

Note

The following picture illustrates the first test case.



Polycarp goes from 1 to 10. The yellow area shows the coverage area of the station with a radius of coverage of 1, which is located at the point of 7. The green area shows a part of the path when Polycarp is out of coverage area.

B1. K for the Price of One (Easy Version)

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

This is the easy version of this problem. The only difference is the constraint on k — the number of gifts in the offer. In this version: $k = 2$.

Vasya came to the store to buy goods for his friends for the New Year. It turned out that he was very lucky — today the offer "k of goods for the price of one" is held in store. **Remember, that in this problem $k = 2$.**

Using this offer, Vasya can buy exactly k of any goods, paying only for the most expensive of them. Vasya decided to take this opportunity and buy as many goods as possible for his friends with the money he has.

More formally, for each good, its price is determined by a_i — the number of coins it costs. Initially, Vasya has p coins. He wants to buy the maximum number of goods. Vasya can perform one of the following operations as many times as necessary:

- Vasya can buy one good with the index i if he currently has enough coins (i.e $p \geq a_i$). After buying this good, the number of Vasya's coins will decrease by a_i, (i.e it becomes $p := p - a_i$).
- Vasya can buy a good with the index i, and also choose exactly $k - 1$ goods, the price of which does not exceed a_i, if he currently has enough coins (i.e $p \geq a_i$). Thus, he buys all these k goods, and his number of coins decreases by a_i (i.e it becomes $p := p - a_i$).

Please note that each good can be bought no more than once.

For example, if the store now has $n = 5$ goods worth $a_1 = 2, a_2 = 4, a_3 = 3, a_4 = 5, a_5 = 7$, respectively, $k = 2$, and Vasya has 6 coins, then he can buy 3 goods. A good with the index 1 will be bought by Vasya without using the offer and he will pay 2 coins. Goods with the indices 2 and 3 Vasya will buy using the offer and he will pay 4 coins. It can be proved that Vasya can not buy more goods with six coins.

Help Vasya to find out the maximum number of goods he can buy.

Input

The first line contains one integer t ($1 \leq t \leq 10^4$) — the number of test cases in the test.

The next lines contain a description of t test cases.

The first line of each test case contains three integers n, p, k ($2 \leq n \leq 2 \cdot 10^5$, $1 \leq p \leq 2 \cdot 10^9$, $k = 2$) — the number of goods in the store, the number of coins Vasya has and the number of goods that can be bought by the price of the most expensive of them.

The second line of each test case contains n integers a_i ($1 \leq a_i \leq 10^4$) — the prices of goods.

It is guaranteed that the sum of n for all test cases does not exceed $2 \cdot 10^5$. **It is guaranteed that in this version of the problem $k = 2$ for all test cases.**

Output

For each test case in a separate line print one integer m — the maximum number of goods that Vasya can buy.

Example

input
6 5 6 2 2 4 3 5 7 5 11 2 2 4 3 5 7 2 10000 2 10000 10000 2 9999 2 10000 10000 5 13 2 8 2 8 2 5 3 18 2 1 2 3

output
3 4 2 0 4 3

B2. K for the Price of One (Hard Version)

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

This is the hard version of this problem. The only difference is the constraint on k — the number of gifts in the offer. In this version: $2 \leq k \leq n$.

Vasya came to the store to buy goods for his friends for the New Year. It turned out that he was very lucky — today the offer "k of goods for the price of one" is held in store.

Using this offer, Vasya can buy **exactly** k of any goods, paying only for the most expensive of them. Vasya decided to take this opportunity and buy as many goods as possible for his friends with the money he has.

More formally, for each good, its price is determined by a_i — the number of coins it costs. Initially, Vasya has p coins. He wants to buy the maximum number of goods. Vasya can perform one of the following operations as many times as necessary:

- Vasya can buy one good with the index i if he currently has enough coins (i.e $p \geq a_i$). After buying this good, the number of Vasya's coins will decrease by a_i, (i.e it becomes $p := p - a_i$).
- Vasya can buy a good with the index i, and also choose exactly $k - 1$ goods, the price of which does not exceed a_i, if he currently has enough coins (i.e $p \geq a_i$). Thus, he buys all these k goods, and his number of coins decreases by a_i (i.e it becomes $p := p - a_i$).

Please note that each good can be bought no more than once.

For example, if the store now has $n = 5$ goods worth $a_1 = 2, a_2 = 4, a_3 = 3, a_4 = 5, a_5 = 7$, respectively, $k = 2$, and Vasya has 6 coins, then he can buy 3 goods. A good with the index 1 will be bought by Vasya without using the offer and he will pay 2 coins. Goods with the indices 2 and 3 Vasya will buy using the offer and he will pay 4 coins. It can be proved that Vasya can not buy more goods with six coins.

Help Vasya to find out the maximum number of goods he can buy.

Input

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

The next lines contain a description of t test cases.

The first line of each test case contains three integers n, p, k ($2 \le n \le 2 \cdot 10^5$, $1 \le p \le 2 \cdot 10^9$, $2 \le k \le n$) — the number of goods in the store, the number of coins Vasya has and the number of goods that can be bought by the price of the most expensive of them.

The second line of each test case contains n integers a_i ($1 \le a_i \le 10^4$) — the prices of goods.

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

Output

For each test case in a separate line print one integer m — the maximum number of goods that Vasya can buy.

Example

input
8 5 6 2 2 4 3 5 7 5 11 2 2 4 3 5 7 3 2 3 4 2 6 5 2 3 10 1 3 9 2 2 10000 2 10000 10000 2 9999 2 10000 10000 4 6 4 3 2 3 2 5 5 3 1 2 2 1 2

output
3 4 1 1 2 0 4 5

C. Petya and Exam

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

Petya has come to the math exam and wants to solve as many problems as possible. He prepared and carefully studied the rules by which the exam passes.

The exam consists of n problems that can be solved in T minutes. Thus, the exam begins at time 0 and ends at time T. Petya can leave the exam at any integer time from 0 to T, inclusive.

All problems are divided into two types:

- easy problems — Petya takes exactly a minutes to solve any easy problem;
- hard problems — Petya takes exactly b minutes ($b > a$) to solve any hard problem.

Thus, if Petya starts solving an easy problem at time x, then it will be solved at time $x + a$. Similarly, if at a time x Petya starts to solve a hard problem, then it will be solved at time $x + b$.

For every problem, Petya knows if it is easy or hard. Also, for each problem is determined time t_i ($0 \le t_i \le T$) at which it will become *mandatory* (required). If Petya leaves the exam at time s and there is such a problem i that $t_i \le s$ and he didn't solve it, then he will receive 0 points for the whole exam. Otherwise (i.e if he has solved all such problems for which $t_i \le s$) he will receive a number of points equal to the number of solved problems. Note that leaving at time s Petya can have both "mandatory" and "non-mandatory" problems solved.

For example, if $n = 2$, $T = 5$, $a = 2$, $b = 3$, the first problem is hard and $t_1 = 3$ and the second problem is easy and $t_2 = 2$. Then:

- if he leaves at time $s = 0$, then he will receive 0 points since he will not have time to solve any problems;
- if he leaves at time $s = 1$, he will receive 0 points since he will not have time to solve any problems;
- if he leaves at time $s = 2$, then he can get a 1 point by solving the problem with the number 2 (it must be solved in the range from 0 to 2);
- if he leaves at time $s = 3$, then he will receive 0 points since at this moment both problems will be mandatory, but he will not be able to solve both of them;
- if he leaves at time $s = 4$, then he will receive 0 points since at this moment both problems will be mandatory, but he will not be

able to solve both of them;
- if he leaves at time $s = 5$, then he can get 2 points by solving all problems.

Thus, the answer to this test is 2.

Help Petya to determine the maximal number of points that he can receive, before leaving the exam.

Input
The first line contains the integer m ($1 \le m \le 10^4$) — the number of test cases in the test.

The next lines contain a description of m test cases.

The first line of each test case contains four integers n, T, a, b ($2 \le n \le 2 \cdot 10^5$, $1 \le T \le 10^9$, $1 \le a < b \le 10^9$) — the number of problems, minutes given for the exam and the time to solve an easy and hard problem, respectively.

The second line of each test case contains n numbers 0 or 1, separated by single space: the i-th number means the type of the i-th problem. A value of 0 means that the problem is easy, and a value of 1 that the problem is hard.

The third line of each test case contains n integers t_i ($0 \le t_i \le T$), where the i-th number means the time at which the i-th problem will become mandatory.

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

Output
Print the answers to m test cases. For each set, print a single integer — maximal number of points that he can receive, before leaving the exam.

Example

input
10 3 5 1 3 0 0 1 2 1 4 2 5 2 3 1 0 3 2 1 20 2 4 0 16 6 20 2 5 1 1 0 1 0 0 0 8 2 9 11 6 4 16 3 6 1 0 1 1 8 3 5 6 6 20 3 6 0 1 0 0 1 0 20 11 3 20 16 17 7 17 1 6 1 1 0 1 0 0 0 1 7 0 11 10 15 10 6 17 2 6 0 0 1 0 0 1 7 6 3 7 10 12 5 17 2 5 1 1 1 1 0 17 11 10 6 4 1 1 1 2 0 1

output
3 2 1 0 1 4 0 1 2 1

D. Enchanted Artifact

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

This is an interactive problem.

After completing the last level of the enchanted temple, you received a powerful artifact of the 255th level. Do not rush to celebrate, because this artifact has a powerful rune that can be destroyed with a single *spell* s, which you are going to find.

We define the spell as some **non-empty string** consisting only of the letters a and b.

At any time, you can cast an arbitrary non-empty spell t, and the rune on the artifact will begin to resist. *Resistance* of the rune is the edit distance between the strings that specify the casted spell t and the rune-destroying spell s.

Edit distance of two strings s and t is a value equal to the minimum number of one-character operations of replacing, inserting and deleting characters in s to get t. For example, the distance between ababa and aaa is 2, the distance between aaa and aba is 1, the distance between bbaba and abb is 3. The edit distance is 0 if and only if the strings are equal.

It is also worth considering that the artifact has a resistance limit — if you cast more than $n + 2$ spells, where n is the length of spell s, the rune will be blocked.

Thus, it takes $n + 2$ or fewer spells to destroy the rune that is on your artifact. Keep in mind that the required destructive spell s must also be counted among these $n + 2$ spells.

Note that the length n of the rune-destroying spell s is not known to you in advance. It is only known that its length n does not exceed 300.

Interaction

Interaction is happening through queries.

Each request consists of a single **non-empty** string t — the spell you want to cast. The length of string t should not exceed 300. Each string should consist only of the letters a and b.

In response to the query, you will get *resistance* runes — the edit distance between the strings that specify the casted spell t and the secret rune-destroying spell s. Remember that s contains only the letters a and b.

After breaking the rune, your program should end immediately. A rune is destroyed when you get a response with resistance 0. After receiving the value 0, your program should terminate normally.

In this problem interactor **is not adaptive**. This means that during any test the rune-destroying spell s **does not change**.

If your query is invalid, `-1` will be returned. After receiving this your program should immediately terminate normally (for example, by calling `exit(0)`), otherwise, the testing system may issue an arbitrary verdict.

If the number of spells exceeds limit ($n + 2$, where n is the length of the spell s, which is unknown to you), you will get the *Wrong Answer* verdict.

Your solution may receive the verdict *Idleness Limit Exceeded* if you don't output anything or forget to flush the output buffer.

To flush the output buffer, you need to do the following immediately after printing the query and the line end:

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

Hacks

For hacks, use the following format:

In a single line print the string s ($1 \le |s| \le 300$) of letters a and b, which defines the rune-destroying spell.

The hacked solution will not have direct access to the unknown spell.

Example

input
2 2 1 2 3 0
output
aaa aaab abba bba abaaa aabba

E. The Cake Is a Lie

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

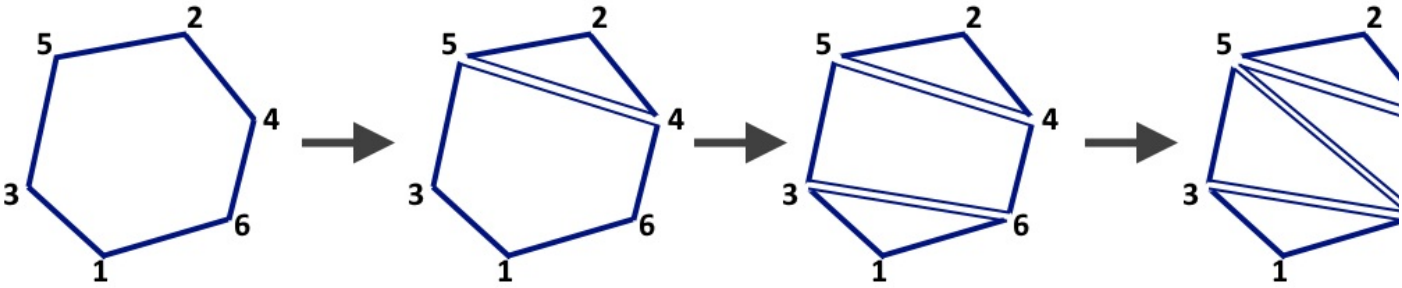
We are committed to the well being of all participants. Therefore, instead of the problem, we suggest you enjoy a piece of cake.

Uh oh. Somebody cut the cake. We told them to wait for you, but they did it anyway. There is still some left, though, if you hurry back. Of course, before you taste the cake, you thought about how the cake was cut.

It is known that the cake was originally a regular n-sided polygon, each vertex of which had a unique number from 1 to n. The vertices were numbered in random order.

Each piece of the cake is a triangle. The cake was cut into $n - 2$ pieces as follows: each time one cut was made with a knife (from one vertex to another) such that exactly one triangular piece was separated from the current cake, and the rest continued to be a convex polygon. In other words, each time three consecutive vertices of the polygon were selected and the corresponding triangle was cut off.

A possible process of cutting the cake is presented in the picture below.



Example of 6-sided cake slicing.

You are given a set of $n - 2$ triangular pieces in random order. The vertices of each piece are given in random order — clockwise or counterclockwise. Each piece is defined by three numbers — the numbers of the corresponding n-sided cake vertices.

For example, for the situation in the picture above, you could be given a set of pieces: $[3, 6, 5], [5, 2, 4], [5, 4, 6], [6, 3, 1]$.

You are interested in two questions.

- What was the enumeration of the n-sided cake vertices?
- In what order were the pieces cut?

Formally, you have to find two permutations $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$) and $q_1, q_2, \ldots, q_{n-2}$ ($1 \le q_i \le n - 2$) such that if the cake vertices are numbered with the numbers $p_1, p_2, \ldots, p_n$ in order clockwise or counterclockwise, then when cutting pieces of the cake in the order $q_1, q_2, \ldots, q_{n-2}$ always cuts off a triangular piece so that the remaining part forms one convex polygon.

For example, in the picture above the answer permutations could be: $p = [2, 4, 6, 1, 3, 5]$ (or any of its cyclic shifts, or its reversal and after that any cyclic shift) and $q = [2, 4, 1, 3]$.

Write a program that, based on the given triangular pieces, finds any suitable permutations p and q.

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. Then there are t independent sets of input data.

The first line of each set consists of a single integer n ($3 \le n \le 10^5$) — the number of vertices in the cake.

The following $n - 2$ lines describe the numbers of the pieces vertices: each line consists of three different integers a, b, c ($1 \le a, b, c \le n$) — the numbers of the pieces vertices of cake given in random order. The pieces are given in random order.

It is guaranteed that the answer to each of the tests exists. It is also guaranteed that the sum of n for all test cases does not exceed 10^5.

Output

Print $2t$ lines — answers to given t test cases in the order in which they are written in the input. Each answer should consist of 2 lines.

In the first line of an answer on a test case print n distinct numbers $p_1, p_2, \ldots, p_n$($1 \le p_i \le n$) — the numbers of the cake vertices in clockwise or counterclockwise order.

In the second line of an answer on a test case print $n - 2$ distinct numbers $q_1, q_2, \ldots, q_{n-2}$($1 \le q_i \le n - 2$) — the order of cutting pieces of the cake. The number of a piece of the cake corresponds to its number in the input.

If there are several answers, print any. It is guaranteed that the answer to each of the tests exists.

Example

input
3 6 3 6 5 5 2 4 5 4 6 6 3 1 6 2 5 6 2 5 1 4 1 2 1 3 5 3 1 2 3
output
1 6 4 2 5 3 4 2 3 1 1 4 2 6 5 3 3 4 2 1 1 3 2 1

The only programming contests Web 2.0 platform