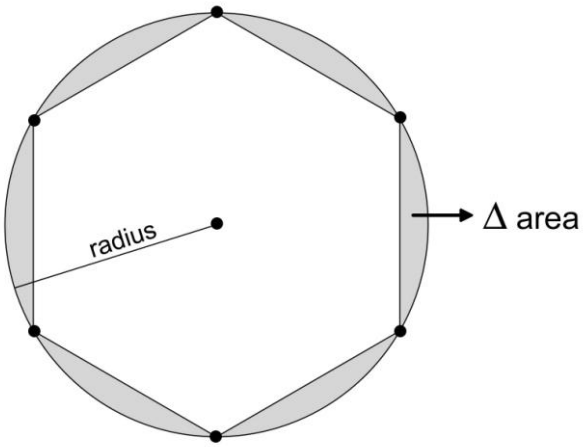
Problem A	
Almost Circle	Submit File : circle.exe | circle.class Input File : circle.in Output File : circle.out Time Limit : 1 second

Problem Description

By the definition, a circle is the set of all points in a plane at a fixed distance (radius) from a given point (center). A circle can be made from a regular polygon (all sides have the same length, and all angles are congruent) of an infinite number of corners.

In this problem, we would like to deal with an almost circle. An almost circle is a regular *N*-gon that has a difference (Δ) area with the circle less than or equal to *P*.



The picture above shows us a figure of a polygon with 6 corners (6-gon) and its Δ area. As you might figure out, the more you have corners, the smaller Δ area will be. In fact, if we have infinite corners, then Δ area will be zero (and it is a circle).

Given the circle radius *R* and maximum Δ area *P*, find out *N* the minimum corner needed to form an almost circle.

Input Specification

The first line of input contains an integer *T*, the number of test cases follow. Each case consists of two integers, *R* ($1 \le R \le 1{,}000$) the circle radius, and *P* ($1 \le P \le 1{,}000{,}000$) the maximum Δ area.

Output Specification

For each case, print in a single line, *N* the minimum number of corners needed to form an almost circle. You may safely assume that *N* will be less than 1000.

Sample Input	Output for Sample Input
4 7 60 7 100 7 40 7 27	4 3 5 6

Problem B	
Banana Party	Submit File : circle.exe | circle.class Input File : circle.in Output File : circle.out Time Limit : 1 second

Problem Description

Chimp loves banana. He can't walk without eating banana. He needs 1 banana to walk 1 km, but he needs none to climb a banana tree.

One day, Chomp, a friend of Chimp, invites Chimp to a Banana Party at his house. Unfortunately, Chomp's house is pretty far from Chimp's and there's only one route. So he has to make sure that he will have enough banana supply to walk to Chomp's (remember, he can't walk without banana).

If all bananas in his house is not enough to get him to Chomp's, he can stop by to climb and collect bananas from banana trees along the route anytime. Chimp knows all banana trees' locations and the number of bananas on each tree. Chimp is in a hurry, so he doesn't want to stop more than what is necessary.

Write a program to help Chimp finds the minimum number of stops (to climb and collect banana) needed to travel from his house to Chomp's. Assume that Chimp can bring an infinite amount of banana with him (don't ask how). By the way, Chimp and Chomp are monkeys (in case you wonder).

Input Specification

The first line of input contains an integer *T*, the number of test cases follow.

Each case begins with three integers, *H* ($0 \le H \le 100$) the number of banana in his house, *L* ($0 \le L \le 10000$) the Chomp's house location in km, and *N* ($0 \le N \le 100$) the number of banana trees along the route. The next *N* lines each contain two integers, *P* ($0 \le P \le 100$) the distance of i^{th} banana tree from Chimp's house and B ($0 \le B \le 100$) the number of bananas in that tree. Chimp's house is at 0 km.

Output Specification

For each case, print in a single line, the number minimum number of stop needed by Chimp to travel from his house to Chomp's. If there's no way he can arrive at Chomp's house, print -1.

Sample Input	Output for Sample Input
2 8 10 3 1 3 4 3 7 1 2 10 1 3 4	1 -1

Problem C	
MU vs. Chelsea	Submit File : mu.exe | mu.class Input File : mu.in Output File : mu.out Time Limit : 1 second

Problem Description

Yay! My favorite football team, Manchester United (MU/The Red Devils), is going to have a match with team Chelsea (The Blues) tonight. This is my most anticipated match, and surely I don't want to miss it. Before the match begins, Charlie (a statistic geek) asked me with the probability of each player on my favorite team to score a goal. As a perhaps-not-a-really-statistic-geek, I'm being curious with his question. Ok, there's still enough time before the match begin. I'll collect the data and do some paper and pencil works.

To solve this problem, I've come up with an idea. What I do need are Ferguson's strategy (note: Ferguson is MU's coach) and the statistic of several previous matches of MU vs. Chelsea.

For the Ferguson's strategy, of course I know it very well. Any player who has the ball may either shoot, or pass it to other player in his team. Ferguson also won't be happy at all if The Red Devils fooling around, so the team is not allowed to do more than 60 consecutive passes.

After doing careful analysis, I came up with this important information of The Red Devils team when they have match with The Blues team:

1. A matrix containing the success rate of each MU's player to pass the ball to any player in their team. Note that a player can also pass the ball to himself (there's a trick to do it).
2. The success rate of each MU's player to score goal from his shot.

For this problem, let's consider that at the beginning, the ball will always be on MU goal keeper's hand (player number 1). I've got the idea, I've got the data, but I can't do the math. Now, please help me.

Input Specification

The first line of input contains an integer *T*, the number of test cases follow.

Each test case consists of three parts:

The first part is a list of 11 player names (number 1 to 11), one player in each row. Player names will be no longer than 20 characters and may contain spaces.

The second part is a matrix P (11x11), the success rate of each player to pass the ball to any player in the team. The cell at *i*-th row and *j*-th column of the matrix means the success passing rate of player number *i* to player number *j* in percentage ($0 \le P[i][j] \le 100$).

The third part is the success rate of each player to score goal from his shot. The *i*-th element is the success shooting rate of player number *i*.

Output Specification

For each case, print "`Case #N:`" (without quote) where N is the case number in a single line.

The next 11 lines contain the highest success rate of each player to score goal in descending order. If there is a tie, player with higher number will be printed first. For example, if there is a tie between player no. 10 and player no. 11, then print player no. 11 first and followed player no. 10.

Line Format: `xx.xx% player_name` where xx.xx represents the success rate of scoring goal.

See sample input/output for clarity.

Sample Input	Output for Sample Input
1 Edwin van der Sar Gary Neville Patrice Evra Owen Hargreaves Rio Ferdinand Wes Brown Cristiano Ronaldo Anderson Louis Saha Wayne Rooney Ryan Giggs 69 82 17 79 66 42 15 28 79 30 2 55 6 71 70 44 35 66 60 87 36 55 65 74 92 13 97 22 3 40 74 43 61 39 19 10 89 78 36 79 32 89 59 65 51 40 49 69 15 16 96 4 42 14 6 79 13 51 29 92 23 63 2 46 5 14 49 59 100 70 47 3 89 4 81 29 98 95 99 99 97 37 44 37 75 71 87 84 26 73 50 19 62 33 18 76 26 38 18 87 68 21 64 5 35 26 68 53 48 36 47 25 49 20 92 37 47 45 83 51 94 8 46 18 52 28 66 97 45 85 95 88	Case #1: 67.15% Louis Saha 61.46% Cristiano Ronaldo 54.64% Ryan Giggs 49.62% Wayne Rooney 41.08% Owen Hargreaves 37.72% Gary Neville 27.72% Wes Brown 27.02% Anderson 18.48% Rio Ferdinand 11.40% Patrice Evra 8.00% Edwin van der Sar

Problem D

Number of Divisor

Submit File : number.exe | number.class
Input File : number.in
Output File : number.out
Time Limit : 1 second

Problem Description

Given two integers *X* and *Y*, calculate how many numbers are there that divides X^Y. The term *divide* means that you can divides two integers without leaving any remainder.

For Example,

Let *X* = 6 and *Y* = 2.

X^Y is equal to 6^2, which is 36.

There are 9 numbers which divides 36, which are: {1, 2, 3, 4, 6, 9, 12, 18, and 36}

Input Specification

The first line of input contains an integer *T*, the number of test cases follow.

Each test case consists of two integers *X* ($1 \le X \le 1000$) and *Y* ($0 \le Y \le 6$).

Output Specification

For each case, print in a single line the number of divisor of X^Y in the format: "`Number of divisor of <X^Y> is <P>.`" (without quotes). See sample output for clarity.

Sample Input	Output for Sample Input
2 6 2 10 1	Number of divisor of 36 is 9. Number of divisor of 10 is 4.

Problem E	
Lining Up	Submit File : line.exe | line.class Input File : line.in Output File : line.out Time Limit : 1 second

Problem Description

N people P_1, P_2, P_3, …., P_N go to the cinema to watch Jackie Chan's new movie, Rush Hour 3. There's so many people want to watch the movie so the cinema is very crowded. To buy a ticket, first one should get an ID number from a ticket machine (everyone uses credit card at the weekend). After he has an ID number, he should immediately join the queue to buy the movie's ticket.

Normally, the ticket machine will produce ID numbers in ascending order (first come, first serve). But unfortunately, on that day the ticket machine produced a random order of ID number. To solve this problem, the wise manager of the BowoCinema, TheWiseMarcadian decided that each people would be served (be able to buy the ticket) by the ID number which they got from the ticket machine, not by the queue arrangement.

Many people in the queue got bored of waiting. Each of them wanted to know how many people behind him who had smaller ID number. The manager called this the Waiting-Sequence.

For Example, let there be 4 persons in the queue whose ID numbers were 3 4 2 1 consecutively (the left most one would be the most front people's ID number, while the right most would be the most behind one's in the queue)
There were 2 persons behind the first one (ID = 3) who had smaller ID number (ID 2 and 1).
There were 2 persons behind the second one (ID = 4) who had smaller ID number (ID 2 and 1).
There was 1 person behind the third one (ID = 2) who had smaller ID number (ID 1).
There was no people behind the fourth one (ID = 1) who had smaller number ID.

Therefore, the Waiting-Sequence for queue arrangement "3 4 2 1" is "2 2 1 0".

Supposed that you have the Waiting-Sequence, could you construct the queue arrangement from that sequence?

Input Specification

The first line of input contains an integer *T*, the number of test cases follow.

Each test case begins with an integer N ($1 \le N \le 1000$), the number of person(s) in the queue. The next line contains *N* integer(s) indicating the Waiting-Sequence from the most front to the most behind one in the queue.

Output Specification

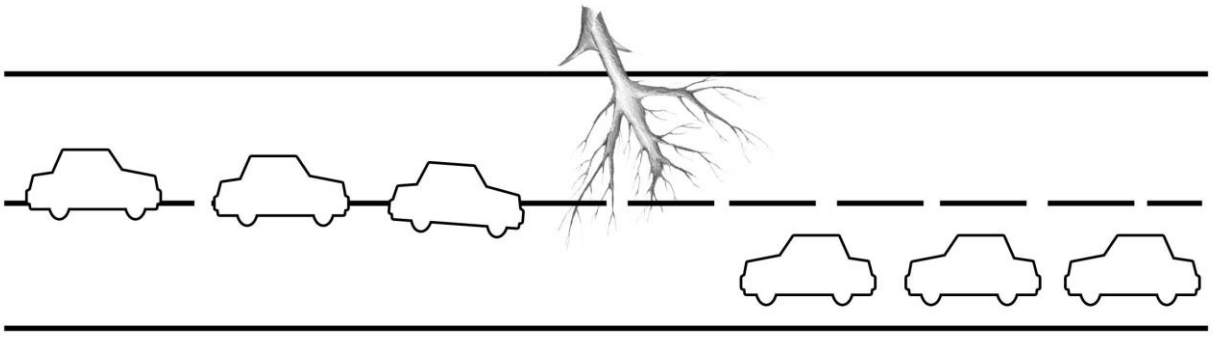
For each case, print in a single line the queue arrangement from the given Waiting-Sequence. Each number should be separated by a single space.

Sample Input	Output for Sample Input
3 4 2 2 1 0 4 0 0 0 0 4 3 2 1 0	3 4 2 1 1 2 3 4 4 3 2 1

Problem F	
Fallen Tree	Submit File : tree.exe | tree.class Input File : tree.in Output File : tree.out Time Limit : 1 second

Problem Description

Early on this morning, the old tree in John's garden had collapsed onto the street in front of his house. Fortunately no one was injured and no cars were damaged. The tree was big enough to cover one side of the street so cars from one direction can't go through. Although it's still morning, this little accident has tied up the traffic because cars from both directions can only use the only remaining clear side, which creates bottleneck. And because of this, everyone is getting impatient and starts to honk their car.



Feeling bad, John decided to help control the traffic. First, he forbids more cars to get into the street. And then, he controls the cars that already caught in the traffic to escape from there. Unfortunately, no car is willingly to turn back, so the only option is to allow each car moving forward and escape from the clear side.

Each car driver has various type of temperament (he can tell it by the number of their honk in a minute), but easy to predict. When cars from his/her side are moving, he/she will not honk. If they're stop moving (because of the opposite direction cars turn to use the clear side), then he/she will start to honk again.

The most front car needs exactly one minute to escape from the bottleneck. To alternate the turn from one direction to another, you need one extra minute, and other cars that still queued in both directions will also honk on that minute.

Supposed that you know the number of cars in each direction, and the number of honk in a minute for each car, help John to calculate the least amount of total honk he will receive from all cars.

Input Specification

The first line of input contains an integer *T*, the number of test cases follow.

Each case begins with two integers, *L* ($0 \le L \le 1{,}000$) the number of cars in the left part (who want to go right), and *R* ($0 \le R \le 1{,}000$) the number of cars in the right part (who want to go left). The next two lines, each contains *L* and *R* integers consecutively. Each line contains H_i the number of honk per minute ($0 \le H_i \le 100{,}000$) from each car. The first element is the most front car in each queue, while the last element is the most back in the queue (last to escape).

Output Specification

For each case, print in a single line the number of least amount of honk that John will possibly receive from all cars.

Sample Input	Output for Sample Input
2 4 4 5 1 1 1 2 2 2 2 5 5 6 1 1 1 1 2 2 2 2 2	34 48

Problem G

The Adventure in Panda Land Part 2: **Panda Tower**	Submit File : panda.exe | panda.class Input File : panda.in Output File : panda.out Time Limit : 3 second

Problem Description

There are three altars in Panda Land. On the first altar, there is a stone tower built by arranging *N* stones of various sizes on top of each other. The arrangement was made such that there's no larger stone put on top of a smaller one.

Li Ming, the panda chief, has decided to move the stone tower to the P^{th} altar. However, he can only move a single stone (one on the top) at a time from one altar to another. No stone is allowed to be put on any other places except on the altars. There is at most one stack of stones on any altars and no larger stone should be put on top of smaller stone.

Days passed, and Li Ming hasn't completed the job, because he just moved stones in a random order, although he's still following to the given rules. Frustrated by this problem, he decided to come to the human world and asked for help (yeah, you understand what he wants).

Given the current condition for each altar, help Li Ming to find the moves sequence needed to completely move all stones to the P^{th} altar. The number of moves should not exceed 2^N-1, otherwise Li Ming will get more frustrated than ever (and if it happens, run for your life).

Input Specification

The first line of input contains an integer *T*, the number of test cases follow.

Each test case begins with *N* ($4 \le N \le 15$), the number of stones. Each of the three altar's condition will be described in two lines. The first line contains S_i, the number of stone on i^{th} altar. The second line contains S_i integers denoting each stone's size on i^{th} altar in descending order. The sum of all stones in three altars will be equal to *N*. The last line of each case contains *P* ($1 \le P \le 3$), the target altar.

Output Specification

For each case, print the *L* ($0 \le L \le 2^N$-1), the number of moves needed to completely moves all stones to the P^{th} altar. The next *L* line should describe the moves sequence. Each line contains two integer *A* and *B*

($1 \le A, B \le 3$), which means "move the top stone on A^{th} altar into the top of B^{th} altar". If there is more than one such sequence, just output any of them.

If there is no such sequence below 2^N-1, print "impossible" (without quotes) in a single line, no need to print *L* and the sequence.

Sample Input	Output for Sample Input
2 4 4 4 3 2 1 0 0 3 5 1 5 2 4 2 2 3 1 2	15 1 2 1 3 2 3 1 2 3 1 3 2 1 2 1 3 2 3 2 1 3 1 2 3 1 2 1 3 2 3 29 3 2 3 1 2 3 2 1 3 1 2 3 1 3 1 2 3 2 1 3 2 1 2 3 1 3 1 2 3 1 3 2 1 2 3 1 2 3 2 1 3 1 3 2 1 2 1 3 2 3 1 2 3 1 3 2 1 2

Problem H	
1 2 Hop!	Submit File : hop.exe | hop.class Input File : hop.in Output File : hop.out Time Limit : 1 second

Problem Description

Lie is a good kindergarten's teacher. He has taught the children to count numbers from 1 to 9 using a simple game. Here is the game:

Lie will say a number to the children. For the number that he says, he wants the children to count from 1 to *N* for *N* times, with every number *N* replaced by a yell "hop!" For example:
If he says 3, then the children will count: "1 2 hop! 1 2 hop! 1 2 hop!"
If he says 4, then the children will count: "1 2 3 hop! 1 2 3 hop! 1 2 3 hop! 1 2 3 hop!"
However if he says any number larger than 9, then the children will shout: "what?"

Now, write a program to simulate Lie's teaching. You should do it FAST or other contestants will beat you.

Input Specification

The first line of input contains an integer *T*, the number of test cases follow.

Each test case contains a single integer *N* ($1 \le N \le 20$), the number that Lie says.

Output Specification

For each case, print the children's yelling in a single line. Each counting should be separated by a single space (see output example).

Sample Input	Output for Sample Input
3 3 4 15	1 2 hop! 1 2 hop! 1 2 hop! 1 2 3 hop! 1 2 3 hop! 1 2 3 hop! 1 2 3 hop! what?