Birdman of Waikiki

Filename: `BIRDMAN`

On Waikiki Beach in Hawaii, there is a fellow known as the "Birdman of Waikiki." He collects money for various charities by taking photographs of you posing with his birds and then asking for donations. A popular time for this is at sunset (Hawaii has some spectacular sunsets). Unfortunately, the Birdman of Waikiki makes the amateur mistake of framing the shot with you in the foreground blocking the sunset in the background. He needs a program to help him.

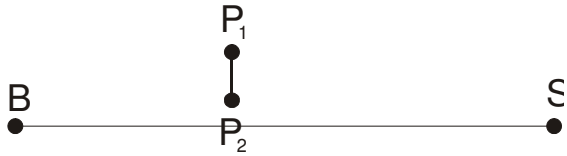
The Problem:

Given the location of the sunset, the location of the Birdman of Waikiki, and a description of you, determine whether the Birdman of Waikiki is showing the sunset in the picture or not. The sunset and the Birdman of Waikiki will each be represented as 2-D points. A line segment will represent you (you're awfully thin, after all) and will be specified by two 2-D points. The Birdman of Waikiki is properly showing the sunset if you do not obstruct the path from him to the sunset in any way (in other words, no point along the line segment describing you inclusive of the end points blocks the segment from the Birdman of Waikiki to the sun).

The Input:

The first line will contain a single integer, *n*, representing the number of photographs to check. For each *n*, there will be a line with four integer pairs. The first integer pair will be a point, *B*, and is the (x,y) location of the Birdman of Waikiki. The second integer pair will be a point, *S*, and is the (x,y) location of the sunset. The third and fourth integer pairs, points P_1 and P_2, will be the (x,y) locations of the ends of the segment that describe you.

Birdman of Waikiki did *not* show sunset properly.



Birdman of Waikiki did show sunset properly.

The Output:

For each photograph, output on a new line either "`Good picture, Birdman!`" if the sunset can be seen or "`Move to the left or right!`" if it cannot.

Sample Input:

```
2
5 0 5 100 3 2 7 2
1 2 5 1 5 2 5 4
```

Sample Output:

```
Move to the left or right!
Good picture, Birdman!
```

Double Farmland

Filename: *doubleland*
Time limit: *10 seconds*

There are a pair of twins, Danny and Donny, in Fortunate Farmland who will be inheriting several farms from their parents in the near future. Since the twins and their parents truly believe in fairness, a requirement of the inheritance documents is that both twins receive sets of farms that have the same total value. One of the twins Danny, really covets a subset of the farms his parents own. Unfortunately, of all the possible distributions of the farms to the twins where the value of the two sets the farms have been split into are equal, the parents will simply choose of those splits at random. Danny knows the values of all of his parents' farms. Help him calculate the probability that he'll actually get all of the farms in his desired subset.

The Problem

Given the values of each of the farms that Danny or Donny will inherit, as well as a list of the farms that Danny desires, determine the probability that he'll receive each farm on his list, given that the parents will pick a distribution (at random) of the farms such that the sum of the values of Danny's farms equals the sum of the values of Donny's farms.

The Input

The first line of input will contain a single positive integer, n ($n \leq 30$), representing the number of twin inheritance scenarios to consider. The input for each scenario will follow. The first line of input for each scenario will start with a positive integer, f ($2 \leq f \leq 20$), representing the number of farms Danny and Donny's parents are bequeathing to them. The rest of the line will contain f space separated positive integers, $v_1, v_2, ..., v_f$, where v_i ($1 \leq v_i \leq 10^8$) represents the value of the i^{th} farm that will be given to Danny or Donny. It's guaranteed that there will be at least one way to split the farms into two sets of equal value. The second line of each scenario will start with a single positive integer, k ($k < f$), representing the number of farms that Danny desires. The following k space separated integers on the line, which will be in increasing order and all in between 1 and f, inclusive, will represent the farms that Danny desires.

The Output

For each twin scenario, output a fraction in lowest terms representing the probability that Danny will obtain all of the farms on his wish list.

Sample Input

```
3
4 8 10 6 4
1 3
7 1 2 3 4 5 6 7
2 3 7
10 15 15 15 10 10 5 8 8 13 1
2 1 10
```

Sample Output

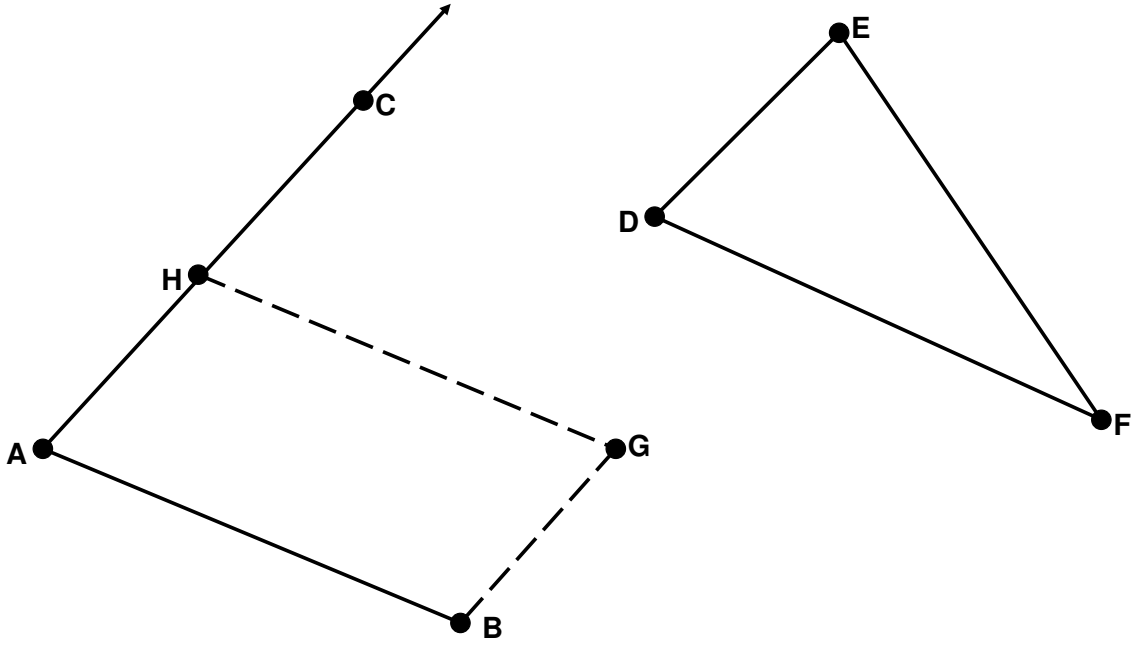
```
1/2
1/8
1/8
```

B: Euclid

In one of his notebooks, Euclid gave a complex procedure for solving the following problem. With computers, perhaps there is an easier way.

In a 2D plane, consider a line segment **AB**, another point **C** which is not collinear with **AB**, and a triangle **DEF**. The goal is to find points **G** and **H** such that:

- **H** is on the ray **AC** (it may be closer to **A** than **C** or further away, but angle **CAB** is the same as angle **HAB**)
- **ABGH** is a parallelogram (**AB** is parallel to **HG**, **AH** is parallel to **BG**)
- The area of parallelogram **ABGH** is the same as the area of triangle **DEF**



The Input

There will be several test cases. Each test case will consist of twelve real numbers, with no more than 3 decimal places each, on a single line. Those numbers will represent, in order:

AX AY BX BY CX CY DX DY EX EY FX FY

where point **A** is (**AX**,**AY**), point **B** is (**BX**,**BY**), and so on. Points **A**, **B** and **C** are guaranteed to NOT be collinear. Likewise, **D**, **E** and **F** are also guaranteed to be non-collinear. Every number is guaranteed to be in the range from **-1000.0** to **1000.0** inclusive. End of the input will be signified by a line with twelve **0.0**'s.

The Output

For each test case, print a single line with four decimal numbers. These represent points **G** and **H**, like this:

GX GY HX HY

where point **G** is (**GX**,**GY**) and point **H** is (**HX**,**HY**). Print all values rounded to 3 decimal places of precision (NOT truncated). Print a single space between numbers. Do not print any blank lines between answers.

Sample Input

```
0 0 5 0 0 5 3 2 7 2 0 4
1.3 2.6 12.1 4.5 8.1 13.7 2.2 0.1 9.8 6.6 1.9 6.7
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Sample Output

```
5.000 0.800 0.000 0.800
13.756 7.204 2.956 5.304
```

Jiminy's Jacuzzis

Filename: `jacuzzi`

Jiminy is opening up a new Jacuzzi business. In order to set-up his showroom, he had his assistant assemble each model of Jacuzzi. Unfortunately, his assistant made a poor choice and mixed up all the sides of all of the Jacuzzis! Now, Jiminy and his assistant are having trouble piecing sides together in order to build the Jacuzzis. They need your help!

The Problem:

Given the lengths of the sides that may make up a Jacuzzi, determine whether it is possible to build one using all of the sides or not. A Jacuzzi is formed by aligning the sides into a polygonal shape with positive area (e.g. the outline of the Jacuzzi). Note that neither Jiminy nor his assistant can cut any of the sides (they must use them as-is) and all sides must be used when constructing each Jacuzzi.

The Input:

The input will begin with a line containing a positive integer, t, representing the number of Jacuzzis to check. Each Jacuzzi then starts with a line containing an integer, n ($1 \leq n \leq 100$), representing the number of sides of the Jacuzzi being built. The next line contains n integers representing the length of each side; each integer will be separated by a single space. All of the side lengths are between 1 and 100, inclusive.

The Output:

For each Jacuzzi, output a line "`Jacuzzi #`*x*`:` *m*" where x is the number of the Jacuzzi in the input (starting from 1) and m is "`YES`" if Jiminy can build the Jacuzzi or "`NO`" otherwise.

Sample Input:

```
5
5
1 2 3 4 11
3
1 2 4
6
2 3 4 5 6 7
3
1 1 10
6
99 2 3 4 5 6
```

Sample Output:

```
Jacuzzi #1: NO
Jacuzzi #2: NO
Jacuzzi #3: YES
Jacuzzi #4: NO
Jacuzzi #5: NO
```

Jumping Game

Filename: *jump*

You've been hired by a playground manufacturing company to design jumping blocks for children. Typically, the blocks are set up in a row and children start at the left most block and then continually jump to the adjacent block to the right until they get to the last block. Each block is an individual piece and a set of blocks can be rearranged in any order. Unfortunately, due to safety reasons, some of the orderings of blocks are not permitted. Depending on the age of the children, there are restrictions on how far up they can jump and how far down they can jump.

Consider a situation where there are four blocks of heights 36 inches, 30 inches, 40 inches and 20 inches where the children can jump up no more than 8 inches and can jump down no more than 16 inches. In this scenario, placing the 36 inch block followed by the 40 inch block, followed by the 30 inch block and ending with the 20 inch block would be a valid arrangement since the three jumps involved would be a jump of 4 inches up, 10 inches down and 10 inches down. However, placing the 40 inch block third and the 20 inch block last would be invalid since this would result in a jump down of 20 inches, more than the maximum of 16 inches. Note that since children don't jump up to the first block from the ground and don't jump down to the ground from the last block, there's no restriction on the heights of these blocks based on how far they are from the ground. (They climb onto the first block and get help from friends getting down from the last block.)

The Problem

Given a set of blocks with distinct heights (in inches), and maximum limits for which children can jump up and jump down, determine the number of orderings of those blocks that don't contain any invalid jumping distances for the kids.

The Input

The first line of input will contain a single positive integer, n ($n \leq 100$), the number of sets of blocks to evaluate. The following n lines will each contain a single input case. All the values within an input case will be separated by spaces. The first value for each input case will be a positive integer, b ($2 \leq b \leq 10$), the number of blocks for the input case. This will be followed by the b unique positive integers, each within the range of 1 to 100, inclusive, representing the heights of each of the blocks for the input case. This will be followed by two more positive integers, u ($u \leq 100$) and d ($d \leq 100$), representing the maximum number of inches the children for that input case can jump up and jump down, respectively.

The Output

For each test case, output a single integer representing the number of orderings of the blocks with every jump from left to right being a valid jump according to the specifications given above.

Sample Input

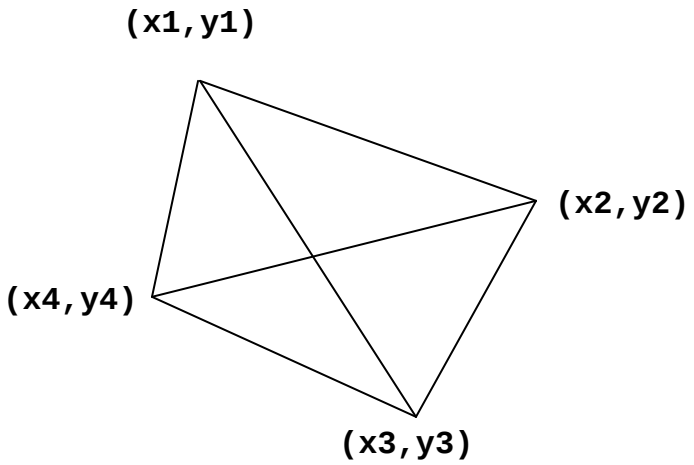
```
3
4 36 30 40 20 8 16
3 10 20 30 6 8
3 10 20 30 8 10
```

Sample Output

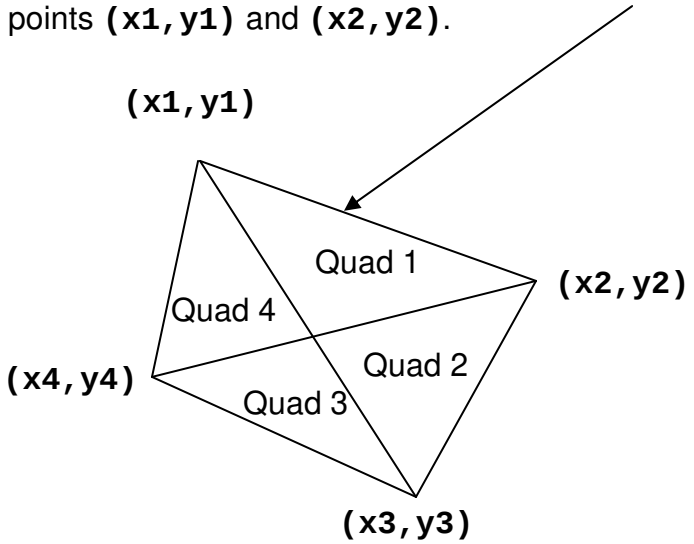
```
3
0
1
```

D: Shoring Up the Levees

The tiny country of Waterlogged is protected by a series of levees that form a quadrilateral as shown below:



The quadrilateral is defined by four vertices. The levees partition the country into four quadrants. Each quadrant is identified by a pair of vertices representing the outside edge of that quadrant. For example, Quadrant 1 shown below is defined by the points **(x1,y1)** and **(x2,y2)**.



It happens very often that the country of Waterlogged becomes flooded, and the levees need to be reinforced, but their country is poor and they have limited resources. They would like to be able to reinforce those levees that encompass the largest area first, then the next largest second, then the next largest third, and the smallest area fourth.

Help Waterlogged identify which quadrants are the largest, and the length of the levees around them

Input

There will be several sets of input. Each set will consist of eight real numbers, on a single line. Those numbers will represent, in order:

X1 Y1 X2 Y2 X3 Y3 X4 Y4

The four points are guaranteed to form a convex quadrilateral when taken in order – that is, there will be no concavities, and no lines crossing. Every number will be in the range from **`-1000.0`** to **`1000.0`** inclusive. No Quadrant will have an area or a perimeter smaller than **`0.001`**. End of the input will be a line with eight **`0.0`**'s.

Output

For each input set, print a single line with eight floating point numbers. These represent the areas and perimeters of the four Quadrants, like this:

A1 P1 A2 P2 A3 P3 A4 P4

Print them in order from largest area to smallest – so ***A1*** is the largest area. If two Quadrants have the same area when rounded to 3 decimal places, output the one with the largest perimeter first. Print all values with 3 decimal places of precision (rounded). Print spaces between numbers. Do not print any blank lines between outputs.

Sample Input

```
1 2 1 5 5 2 2 0
3.5 2.2 4.8 -9.6 -1.2 -4.4 -8.9 12.4
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Sample Output

```
5.100 11.459 3.400 9.045 0.900 6.659 0.600 4.876
44.548 38.972 21.982 25.997 20.342 38.374 10.038 19.043
```

Linear Equation Solver

Filename: *linear*
Time Limit: *1 second*

Your friend Mitchell is taking Arup's Discrete Structures I class. For his homework, he has to find integer solutions for x and y to the equation ax + by = 1, where a and b are given positive integers which are relatively prime to one another. Arup has added one final sinister twist to the homework questions. For each one, he's bounded the desired value(s) for x with a lower and upper bound. Arup's question is simply to find the number of ordered pairs (x, y) of integers that satisfy the equation within the given bounds.

For example, consider the equation $3x + 4y = 1$ with the constraint that $-10 \leq x \leq 5$. In this case, the following ordered pairs satisfy the equation and the constraint on x:

(-9, 7), (-5, 4), (-1, 1), and (3, -2).

Thus, there are 4 solutions to the problem.

Help Mitchell by writing a program to solve all of his homework questions!

The Problem

Given positive integers, a and b, with $\gcd(a, b) = 1$, and integers L and H, with $L \leq H$, find the number of integer solutions (x, y) to the equation $ax + by = 1$, with $L \leq x \leq H$.

The Input

The first line of input will consist of a single positive integer, c ($c \leq 100$), representing the number of input cases to process. Each input case will appear on a single line with the following four space separated values: a ($1 \leq a \leq 10^9$), b ($1 \leq b \leq 10^9$), L ($-10^9 \leq L$) and H ($L \leq H \leq 10^9$), with $\gcd(a, b) = 1$.

The Output

For each input case, output a single integer, representing the number of solutions for the given problem.

Sample Input	Sample Output
2 3 4 -10 5 17 6 0 50	4 8

Treasure Island

Input file: `search.in`

Marketing of movies is a multi-million dollar industry. The Educational and Recreational Auxiliaries Unit (ERAU) creates word search puzzles with names of movie characters, actors, and other words to distribute to promote movies. Word search puzzles are grids of letters with words embedded horizontally, vertically, and diagonally. A word can be in the grid either forward or backward, but its letters must be contiguous. Consider the 9 by 9 grid below:

A	E	I	O	U	B	C	H	R
J	E	D	I	O	D	U	F	E
A	A	G	H	J	T	K	L	K
R	Q	B	P	T	N	T	M	L
J	R	S	B	T	V	W	A	A
A	Y	Z	K	A	Z	Y	X	W
R	X	O	W	V	A	D	O	Y
T	W	S	R	Q	P	N	M	K
E	D	F	G	H	J	K	L	S

The words "EWOK," "HUTT," "JABBA," "JARJAR," "JEDI," "SKYWALKER," "WATTO", "YODA" are all in the puzzle (and are shaded).

ERAU has had complaints that some of the words they think are in the puzzle can't be found, so they want to find a way to speed up the verification of the puzzles they create. Your job is to write a program to verify the list of words that ERAU has created that can be found in a search puzzle.

Input:

The input will consist of a number of puzzle descriptions. The first line of a puzzle description will be an integer n, $0 \le n \le 20$, being the number of search words. The next n lines will each contain a search word of no more than 80 uppercase letters. The following line will contain two integers separated by white space, r and c, $1 \le r, c \le 20$, being respectively the number of rows and the number of columns. The following r lines will each contain c uppercase letters. The last puzzle description has $n = 0$, and should not be processed.

Output:

The first output line for a puzzle should be "`Puzzle number n:`" where n is the position of the puzzle in the list (starting at 1). This is followed by a list of the words (one per line)in the word list that cannot be found in the puzzle, in the same order as they are in the word list. If all the words on the list can be found in the puzzle, then "`ALL WORDS FOUND`" should be output. A blank line should separate each puzzle's output from the next.

Sample Input:

```
9
SKYWALKER
WATTO
HUTT
JABBA
DARTHVADER
JARJAR
JEDI
YODA
EWOK
9 9
AEIOUBCHR
JEDIODUFE
AAGHJTKLK
RQBPTNTML
JRSBTVWAA
AYZKAZYXW
RXOWVADOY
TWSRQPNMK
EDFGHJKLS
2
ADD
BAD
4 4
AADD
BBBB
CCCA
DDDD
3
HELLO
GOODBYE
NOTFOUND
5 10
NEXTLINEBA
UOYOLLEHKC
GOODBYEFOR
NOWWASTHIS
FUNFORYOUQ
0
```

Output Corresponding to sample Input:

```
Puzzle number 1:
DARTHVADER

Puzzle number 2:
ALL WORDS FOUND

Puzzle number 3:
NOTFOUND
```

Adding Sequences

Filename: *sequences*
Time Limit: *1 second*

Given a sequence of *n* integers, we can create a new sequence of *n-1* integers by adding pairs of successive integers in the original sequence. For example, given the following input sequence of 7 values:

3 9 8 6 5 4 12

by adding each pair of consecutive items in the sequence we create the following 6 integer sequence:

12 17 14 11 9 16

Given an original sequence S of *n* integers, we define the sequence S_1 of *n - 1* integers to be the sequence obtained by performing this process on the sequence S. We define S_k to be the sequence obtained by performing this process on the sequence S_{k-1}, for all integers $2 \leq k < n$.

The Problem

Given an integer sequence, S, of *n* integers, as well as a positive integer value, *k*, determine the terms, in order, in the sequence S_k.

The Input

The first line of input will consist of a single positive integer, *c* ($c \leq 100$), representing the number of input cases to process. The first line of each input case contains a two space separated positive integers, *n* ($2 \leq n \leq 100$), representing the number of values in the original sequence, and *k* ($1 \leq k \leq n-1$), the value of *k* for the query. The second line of each input case will contain *n* space separated integers, representing the sequence of integers, in order, for the input case. The integers will be such that all of the values of the sequences S, S_1, S_2, S_3, ..., S_k will be valid 32-bit signed integers.

The Output

For each input case, output each integer in the sequence S_k, followed by a space, on a single line.

Sample Input

```
2
7 1
3 9 8 6 5 4 12
6 5
1 1 1 1 1 1
```

Sample Output

```
12 17 14 11 9 16
32
```

D: The End of the World

Legend says that there is a group of monks who are solving a large Towers of Hanoi puzzle. The Towers of Hanoi is a well-known puzzle, consisting of three pegs, with a stack of disks, each a different size. At the start, all of the disks are stacked on one of the pegs, and ordered from largest (on the bottom) to smallest (on the top). The object is to move this stack of disks to another peg, subject to two rules: 1) you can only move one disk at a time, and 2) you cannot move a disk onto a peg if that peg already has a smaller disk on it.

The monks believe that when they finish, the world will end. Suppose you know how far they've gotten. Assuming that the monks are pursuing the most efficient solution, how much time does the world have left?

Input

There will be several test cases in the input. Each test case will consist of a string of length 1 to 63, on a single line. This string will contain only (capital) **A**s, **B**s and **C**s. The length of the string indicates the number of disks, and each character indicates the position of one disk. The first character tells the position of the smallest disk, the second character tells the position of the second smallest disk, and so on, until the last character, which tells the position of the largest disk. The character will be **A**, **B** or **C**, indicating which peg the disk is currently on. You may assume that the monks' overall goal is to move the disks from peg **A** to peg **B**, and that the input represents a legitimate position in the optimal solution. The input will end with a line with a single capital **X**.

Output

For each test case, print a single number on its own line indicating the number of moves remaining until the given Towers of Hanoi problem is solved. Output no extra spaces, and do not separate answers with blank lines. All possible inputs yield answers which will fit in a signed 64-bit integer.

Sample Input	Sample Output
AAA BBB X	7 0