Problem 1
PC^3

PC^3 is intended as a postprocessor for PC^2, which is popular with ACMers, to allow for an alternative way of scoring to be used for our local contest. After a local guru has written a script for extracting necessary information from the PC^2 databases in the following format:

Team number   Problem number   Time of submission   Judgment

we are now left with the job of identifying the winner according to the local scoring rules. The local scoring system, which totally ignores the time of submission, is based on adding the values associated with the solved problems for each team and then declares the team, or teams, with the highest score as winner, or winners. If a team has multiple accepted submissions for a particular problem, its value will be added only once to the team's score. The values associated with each problem are selected by the contest organizer and are made public in advance.

Your task is to write a program to use the data extracted from the PC^2 database and declare the winner, or winners.

INPUT:

Input to this problem consists of a sequence of one or more contests. Each contest is described by several lines as follows:

The first line consists of four integers: the contest label N, $0 < N < 10$; the number of problems, P, $0 < P < 12$; the number of teams, T, $1 < T < 300$; and the number of submissions, R, $0 < R < 1000$. The integers are separated by a single space.

The second line contains P integers, separated by single space, that describe the values for the P problems given in ascending order of problem numbers. Problems are numbered 1 to P, and their values will be less than or equal to 100.

Each of the following R lines describes the data about one submission, which consists of four (4) integers separated by a single space. The four integers describe team number, problem number, time of submission in milliseconds from start of the five (5) hours contest, and the judge's decision (zero for accepted, and non-zero for rejected).

The input will be terminated by a line that consists of four zeros (0 0 0 0), separated by a single space. This line should not be processed.

OUTPUT:
For each contest, the output is one line that contains the contest label and numbers of the winning team numbers (sorted in increasing order) as shown in the EXAMPLE OUTPUT below.

EXAMPLE INPUT:
```
1 5 20 6
3 3 10 30 100
6 1 16024555 0
3 2 15895629 4
3 2 765629 0
17 4 1120132 0
3 2 1895629 3
6 3 9024555 0
2 5 20 7
3 10 10 30 100
6 1 16024555 0
3 2 15895629 5
3 2 765629 0
17 4 1120139 0
5 4 1895629 0
6 3 9024555 0
10 4 2895629 0
0 0 0 0
```

EXAMPLE OUTPUT:
```
Contest 1 Winner: Team 17
Contest 2 Winner: Team 5 and Team 10 and Team 17
```

Problem 2
Birds of a feather should not play together

A group of ACMers designed a variation, and a much simpler one at that, of the popular game TARGET to help them relax while waiting for the judges' declaration of the final scores. The game consists of finding words that satisfy certain criterion (e.g., start with "dog" ("dogma") or contain "cat" ("scatological"), used in the news today ("nuclear"), etc).

For this game the group agreed that longer words are more worthy than short words and unusual words are better than common words, where commonness is defined as the number of participants who all think of the same word. The score for a word is calculated as its number of letters divided by the number of participants choosing it. Examples are: the word "direkt" chosen by a single German participant scores "6" points for her, and the word "directly" chosen by three participants scores only "8/3" for each of them. The total score for each participant is the sum of his/her individual word scores.

The group has also set some rules for the game:

1. the number of participants to be between two (2) and nine (9),
2. the number of words chosen by a participant to have a maximum of nine (9) words,
3. each of the words, which naturally has at least one letter, to have a maximum of twenty (20) letters, and
4. the length of the name that a participant chooses for himself/herself to be between one (1) and ten (10) letters.

Your task is to write a program that processes several lists of words, one for each participant, and determines the winner. You do not have to worry about checking the validity of the words against the criterion. The participants will perform this task as part of learning new and foreign words.

INPUT:

Input for this problem consists of a sequence of one or more rounds of the game. Each round is described by several lines as follows:

> The first line consists of a single integer, N, $1 < N < 10$; that describes the number of participants.
>
> Each of the following N lines starts with the name of a participant followed after a single space by a list of words, separated by a single space.

The input will be terminated by a line that consists of a single zero (0). This line should not be processed.

OUTPUT:

For each round, the output consists of two lines. The first line contains the round number starting with 1 as shown in the EXAMPLE OUTPUT below. The second line contains the names of participants with the highest score in this round, separated by a single space. The names must appear in the same order as they appear in the input.

EXAMPLE INPUT:

```
4
John dire direct direction
Jane direkt
Joan direction
Janet vondire direction
2
john great admired politician
Kim great loss poorer
2
peter moral choice
paul moral choice
0
```

EXAMPLE OUTPUT:

```
Round 1
John
Round 2
john
Round 3
peter paul
```

Problem 3
Simple Encryption

Chip and Dale have devised an encryption method to hide their messages. They first agree secretly on a number that will be used as the number of columns in a matrix. The sender prepares an intermediate format by removing capitalizations and punctuations and spaces from the message. The sender then enters the letters of the intermediate format along the diagonals of the matrix and pads with extra random letters so as to make a rectangular array of letters. For example, if the message is "There's no place like my office on a muggy day" and there are five columns, Dale would write down

```
t h r n a
e e o c k
s p e e f
l l m f o
i y i n g
o c a g a
e m y y x
u d x x x
```

Note how Dale includes only the letters and writes them all in lower case. Dale then sends the message to Chip by writing the letters in each row. So, the message in its intermediate format would be encrypted as

`thrnaeeockspeefllmfoiyingocagaemyyxudxxx`

In this example, Dale used the character 'x' to pad the message out to make a rectangle, although he could have used any letter.

Your job is to recover for Chip the message in its intermediate format from the encrypted one.

INPUT:

There will be multiple input sets. Input for each set will consist of two lines. The first line will contain an integer in the range 2 . . . 20 indicating the number of columns used. The next line is a string of up to 200 lower case letters. The last input set is followed by a line containing a single zero (0). This line should not be processed.

OUTPUT:

Each input set should generate one line of output, giving the message in its intermediate format followed by the padding letters.

EXAMPLE INPUT:

```
5
thrnaeeockspeefllmfoiyingocagaemyyxudxxx
3
thsiiesaysxx
0
```

EXAMPLE OUTPUT:

```
theresnoplacelikemyofficeonamuggydayxxxx
thisiseasyxx
```

Problem 4
Safe Packing

The manager of a packing warehouse, which specialises in packing breakable items, contracted a supplier for boxes of sizes from the Fibonacci sequence. I am not sure of the reason behind it, but it is rumoured to be related to the recent "Da Vinci code" movie. An item whose size is in the sequence can be packed in a box of equal size with no filling, but an item whose size is not in the sequence must be packed with sufficient filling material to fill the box and protect the item from breaking. An item cannot be split between two boxes, and each item must be packed separately in its own box.

The second twist in this story, which makes it the more bizarre, is that the company only receives a daily filling material delivery of size *F* to be used. At the end of each day, any unused filling material is discarded.

Your task is to maximize the number of items shipped each day for a given amount of filling material *F* and a given list of items.

Fibonacci Numbers are defined as:

$$F(n) = n \qquad \text{for } n < 2$$
$$= F(n-1) + F(n-2) \qquad \text{for } n > 1$$

A list of the first few Fibonacci numbers are:

0 1 1 2 3 5 8 13 21 34 55 ……

Note that each number, with the exception of the first two, is obtained by adding the preceding two numbers.

INPUT:

Input to this problem consists of packing tasks for one or more days. The tasks for each day are described by two lines as follows:

> The first line consists of three integers: the number of items *W*, $0 < W < 1000$ to be packed; the available size of filling material, *F*, $1 < F < 1000$; and maximum size of each item, *S*, $1 < S < 100000000$. The integers are separated by a single space.
>
> The following line contains *W* integers separated by a single space that describe the sizes of items to be packed.

The input will be terminated by a line that consists of three zeros (0 0 0), separated by a single space. This line should not be processed.

OUTPUT:

For each day, the output consists of one line that contains the number of items that can be packed for that day.

EXAMPLE INPUT:

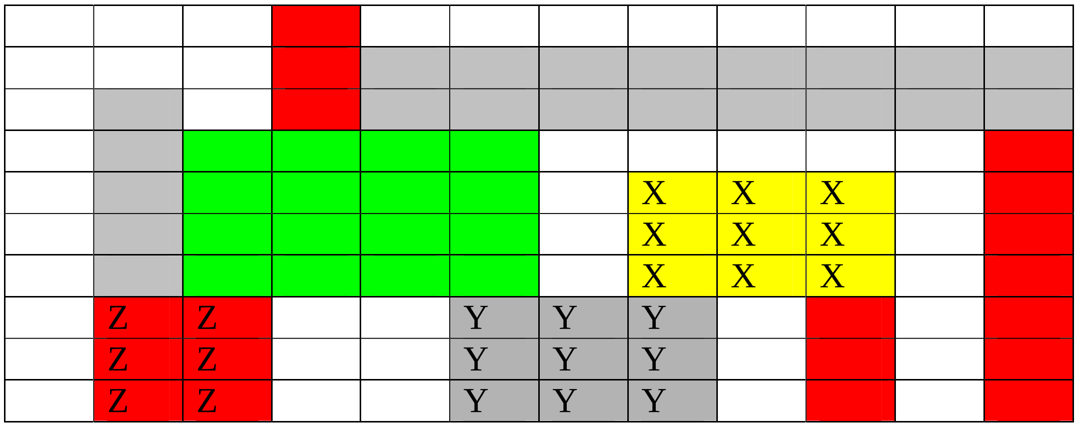
```
4 10 30
7 15 30 5
11 100 5812167
20 40 30 15 17 5812167 23 43 33 13 37
0 0 0
```

EXAMPLE OUTPUT:

```
3
10
```

Problem 5
The Flat Cupboard

The reading of “Flatland: A Romance of Many Dimensions” by *Edwin Abbott* had a profound, and rather unusual, effect on young Bob and his mother too. Bob started to only use things that are either flat or with flat sides: Soup bowls, dishes, glasses, mugs, and even the cutlery. However the annoying part, for his mother that is, was the way he re-organized the cupboard. Bob removed all the shelves and drew a grid, with integer coordinates, (0 0) for the lower left corner and (1000 1000) for the upper right corner, on the back of the cupboard. Bob will always stack the crockery neatly on top of each other. Neatly means aligned with his grid, so that he can record the location of each item by four integers that describe the coordinates of its lower left corner and its upper right corner. For the example, the red mug (labelled with Zs) is described by the four integers (1 0 3 3).



Bob’s mother will not move a crockery item unless no other item in the cupboard exists vertically above it. This way she can be sure that none of Bob’s precious mugs/glasses gets broken. Examples are: taking the yellow mug (labelled with Xs) requires her to move the grey “rectangular” soup bowl first, and taking the grey mug (labelled with Ys) requires her to move four items first.

Your task is to write a program to calculate for Bob's mother the number of items she must move before she can remove the item of her choice.

INPUT:

Input to this problem consists of a sequence of one or more situations. Several lines describe each situation as follows:

The first line contains the number of crockery items M, $0 < M < 100$; given as an integer.

The second line consists of 4*M integers (i.e., M pairs of 2-dimensional coordinates), separated by a single space, that describe the exact positions of crockery items in the flat cupboard.

The third line consists of four (4) integers (i.e., a pair of 2-dimensional coordinates), separated by a single space, that describe the exact position of the item to be removed.

The input will be terminated by a line that consists of a zero (0). This line should not be processed.

OUTPUT:

For each situation, the output is a single line that contains the number of objects to be removed before the desired item can be removed in accordance with Bob's mother desires.

EXAMPLE INPUT:

```
5
0 0 4 4 3 4 6 6 5 0 7 4 4 6 5 8 20 0 21 21
20 0 21 21
4
0 0 4 4 3 4 6 6 5 0 7 4 4 6 5 8
0 0 4 4
0
```

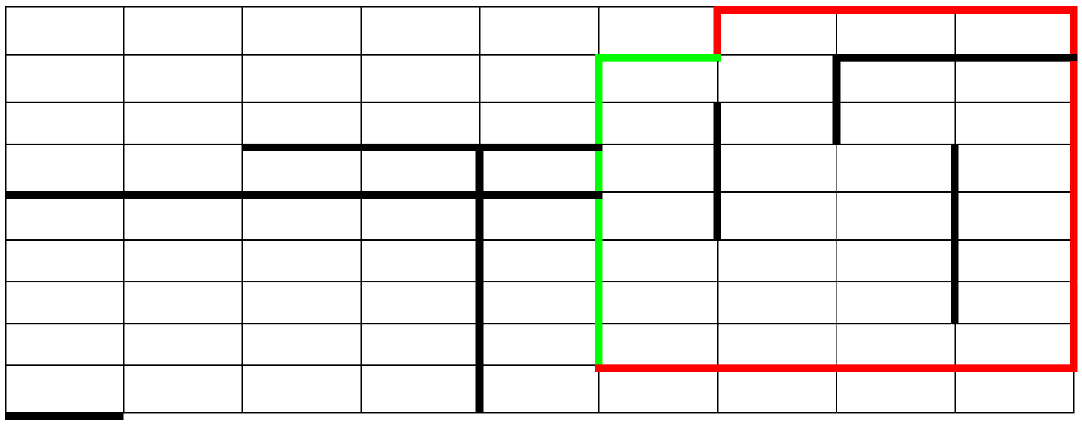
EXAMPLE OUTPUT:

```
0
2
```

Problem 6
Wiring Assistant

A printed circuit board has a number of horizontal and vertical tracks that may be used to connect electronic components. The example below demonstrates a square board with 10 vertical tracks that we shall assign labels from 0 to 9 starting from the left end, 10 horizontal tracks that we shall assign labels from 0 to 9 starting from the bottom end, and 8 laid down wires (shown in black). Each wire occupies a single horizontal or a single vertical track, and is described by its lower left coordinate followed by its upper right coordinate. In the example below, the 8 wires are described by

0 0 1 0   4 0 4 6   0 5 5 5   8 2 8 6   6 4 6 7   7 6 7 8   2 6 5 6   7 8 9 8



A horizontal wire may overlap with a vertical wire in a single grid point. A horizontal wire may not overlap with another horizontal wire, except at their endpoints. Similarly a vertical wire may not overlap with another vertical wire, except at their endpoints.

Your task is to write a program to identify a path on a printed circuit board to connect two given points on the grid, which adheres to the above rules, such that the number of points your path has in common with the existing wires is minimised. The four red wires, in the above example, connect the points (6 8) and (5 1) with minimum number of 1 overlap. The green wires connect the same two points with 2 overlaps.

In some cases a path may not exist, but your program will only be used to connect a pair of points in printed circuit boards where such a path is known to exist. Those cases will be identified and removed by a sophisticated imaging system, but that is another story.

INPUT:

Input to this problem consists of a sequence of one or more design situations. Several lines describe each design situation as follows:

> The first line consists of two integers: the number of existing wires, M, $0 < M < 100$; the number of horizontal tracks (and also the number of vertical tracks) on a square printed circuit board S, 0 < S < 1000000000. The integers are separated by a single space.
>
> The second line consists of 4*M integers (i.e., M pairs of 2-dimensional coordinates), separated by a single space, that describe the exact positions of the existing wires in the design.
>
> The third line consists of four (4) integers (i.e., a pair of 2-dimensional coordinates), separated by a single space, that describe the exact positions of the two points to be connected.

The input will be terminated by a line that consists of two zeros (0 0). This line should not be processed.

OUTPUT:

For each design situation, the output is a single line that contains the minimum number of overlaps.

EXAMPLE INPUT:

```
8 10
0 0 1 0 4 0 4 6 0 5 5 5 8 2 8 6 6 4 6 7 7 6 7 8 2 6 5 6 7 8 9 8
6 8 5 1
7 10
0 0 1 0 4 0 4 6 0 5 5 5 8 2 8 6 6 4 6 7 2 6 5 6 6 4 8 4
6 8 5 1
0 0
```

EXAMPLE OUTPUT:

```
1
0
```

Problem 7
Word Find – *The Second Generation*

The Sudoku craze is sweeping the world, and that has got the "Word Find" problem preparation company worried. The company wants to head off their losses in the market place by introducing a variation of the old game. This example, with 5 words and table size 8, is a typical *second generation word find puzzle*:

P	C	Q	U	E	E	N	J	CLASS
E	O	L	B	E	A	O	R	ROCK
N	K	P	A	P	E	R	X	BOOK
C	L	A	S	S	O	B	R	PENCIL
I	B	P	G	C	S	K	Y	PAPER
L	O	E	K	N	O	N	G	
T	O	R	R	O	C	K	O	
Y	K	I	B	U	R	O	N	

Each of the given words may appear multiple times in the table more but no more than four (4) times. Each word may be oriented in any of the 8 possible directions. The aim is to locate the given words, which will be different to each other, with the added challenge that no two words may share a letter. A generated puzzle is "soluble" if all the words can be found with no two words sharing a letter, and "challenging" if at least one pair of the given words shares a letter in the table.

For the sake of saving development cost and reducing time-to-market, the company decided to use a tweaked version of their software to create the new puzzles and then reject those that are not suitable.

Your task is to write a program to test the generated puzzles and to accept the ones that are both "challenging" and "soluble" and to reject all other puzzles. For the above example, your program should accept the generated puzzle and report "YES" for being "challenging" and "YES" for being "soluble".

INPUT:

Input to this problem consists of a sequence of one or more puzzles. Each puzzle is described by several lines as follows:

> The first line consists of two integers: the number of words, W, $1 < W < 10$; and the table size, N, $1 < N < 100$. The integers are separated by a single space.
>
> Each of the following W lines contains one word, which does not contain any spaces.
>
> Each of the following N lines describes one row of the table, which consists of N letters separated by a single space. All letters are upper case.

The input will be terminated by a line that consists of two zeros (0 0), separated by a single space. This line should not be processed.

OUTPUT:

For each puzzle, the output consists of two lines. The first line contains the puzzle number starting with the value of one (1), followed by a ": " and followed by ACCEPT or REJECT as shown in the EXAMPLE OUTPUT below. The second line contains YES if the puzzle is challenging and NO otherwise, followed by a single space, and followed by YES if soluble and NO otherwise.

EXAMPLE INPUT:

```
3 4
PEN
NET
PET
P E N S
O N E T
P E T K
S P E N
2 3
NO
OR
O R M
X R Y
T N O
2 2
NO
OR
N O
X R
0 0
```

EXAMPLE OUTPUT:

```
Puzzle 1: ACCEPT
YES YES
Puzzle 2: ACCEPT
YES YES
Puzzle 3: REJECT
YES NO
```

Problem 8
Software Dispatcher

The new boss of a rural fire service of the “rectangular state”, whose map is drawn on a grid, with integer coordinates that are multiples of a meter, (0 0) for the lower left corner and (10000 10000) for the upper right corner, in a nation far away used to be an IT person. She believes that a good computer system, especially one based on provably correct algorithms, can outperform humans in stressful situations. That is the reason, when asked; she could not pass the opportunity of posing this problem to the best problem-solvers in the southern hemisphere.

75																			
70																			
65																			
60																			
55																			
50	F										F								
45																			
40																			
35																			
30																			
25						H													
20																			
15																			
10																			
5																			
0	H					F					H					F			
	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90

It is easy to describe, the boss said: At the height of the fire season, the department maintains a database of fire fighter unit locations and the reported sightings of fire. The horizontal and vertical distance, of the closest grid point, from the left lower corner of the state is used to describe the location of a fire or a fire fighter unit. An experienced team of officers dispatches each of the units to one of the fires, with the objective that the longest time for a unit to reach a fire is minimized. At the end of a long day, members of the team are exhausted and their judgement may not be the best possible. The above example shows 4 fires (marked by the letter F) and 3 units (marked by the letter H) already deployed in the field.

Your task is to program a method for dispatching the fire fighter units to fire locations so that they arrive at their destinations in the shortest possible time.

INPUT:

Input to this problem consists of a sequence of one or more scenarios. Several lines describe each scenario as follows:

> The first line consists of three integers: the number of fire fighters, *H*, $0 < H < 100$; the number of fires, *F*, $H < F < 200$, and the speed that all fire fighters' travels, *V* (in meters/second). The integers are separated by a single space.
>
> The third line consists of 2**H* integers (i.e., *H* 2-dimensional coordinates) that describe positions of the fire fighters to be handled. The integers are separated by a single space.
>
> The second line consists of 2**F* integers (i.e., *F* 2-dimensional coordinates) that describe positions of the bush fires to be handled. The integers are separated by a single space.

The input will be terminated by a line that consists of three zeros (0 0 0). This line should not be processed.

OUTPUT:

For each scenario, the output is a single line that contains the shortest time, rounded-up if necessary (e.g., all values larger than "3" and less than "4" are rounded-up to "4"), in seconds for the *H* fire fighters to arrive at *H* different fire locations.

EXAMPLE INPUT:

```
3 4 10
0 0 25 25 50 0
0 50 50 50 25 0 75 0
0 0 0
```

EXAMPLE OUTPUT:

```
4
```

Problem 9
Video Watcher

Rob runs a one-man security firm. When he is not out on the street doing what PIs usually do, he makes his money watching videos on his broadband connected computer (Security video streams, that is). Being a very stressful job, even with its meager pay of no more than 200 dollars for a video stream, as he is required to raise the alarm immediately when a potential security breach occurs, he elected to impose the rule of not watching more than two video streams at any time. For each job Rob knows the starting time (given in minutes with "0" as mid-night and all times belong to the same day), duration (in minutes) and payment (in dollars).

Your task is to write a program to help Rob to select the videos he must watch to maximize his pay for a given day.

INPUT:

Input to this problem consists of a sequence of one or more scenarios. Three lines describe the situation of each scenario as follows:

The first line consists of two integers: the label for the scenario, N, $0 < N < 100$; and the number of videos; V, $0 < V < 100$.

The second line contains V triples of integers, separated by single space, which describe the payment, start-time (in minutes, with "0" as midnight) and duration (in minutes) for each of the V videos.

The input will be terminated by a line that consists of two zeros (0 0), separated by a single space. This line should not be processed.

OUTPUT:

For each scenario, the output is a single line that contains Rob's maximum possible pay for the day.

EXAMPLE INPUT:

```
1 6
20 500 120 10 600 100 100 640 30 50 700 200 90 1100 1200 200 650 1000
0 0
```

EXAMPLE OUTPUT:

```
460
```

Problem 10
The Prank

A local guru has written a script for extracting necessary information from the PC^2 databases in the following format:

Team number Problem number Time of submission Judgment

to allow for an alternative way of scoring to be used for our local contest. The local scoring system, which totally ignores the time of submission, is based on adding the values associated with the solved problems for each team and then declares the team, or teams, with the highest score as winner, or winners. If a team has multiple accepted submissions for a particular problem, its value will be added only once to the team's score. The values associated with each problem are selected by the contest organizer and are made public in advance.

Our guru also scrambled the extracted information, by permuting the "Team number" entries, and provided the permutation number as the clue for the organizer. He claims this is a good method to maintain privacy when number of submissions is small. For example, for the contest with four submissions, shown on the left side below, the "Team number" entries have been permuted according to the 9th permutation of four items and shown on the right side.

3 2 765629 0
17 4 1120132 0
3 2 1895629 3
6 3 9024555 0
	17 2 765629 0
	3 4 1120132 0
	3 2 1895629 3
	6 3 9024555 0

You may recall that the number of permutations of "*x*" items is "*x*!". The table below shows the permutations of four (4) items sorted in lexicographically increasing order, as an example, along with their permutation numbers from 1 to 24.

1	1	1	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3	4	4	4	4	4	4
2	2	3	3	4	4	1	1	3	3	4	4	1	1	2	2	4	4	1	1	2	2	3	3
3	4	2	4	2	3	3	4	1	4	1	3	2	4	1	4	1	2	2	3	1	3	1	2
4	3	4	2	3	2	4	3	4	1	3	1	4	2	4	1	2	1	3	2	3	1	2	1
🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Your task is to write a program to use the data provided by our guru, descramble the “Team number” entries, and to declare the winner, or winners.

INPUT:

Input to this problem consists of a sequence of one or more contests. Each contest is described by several lines as follows:

> The first line consists of four integers: the contest label *N*, 0< *N* <10; the number of problems, *P*, 0 < *P* < 12; the number of teams, *T*, 1 < *T* < 300; and the number of submissions, *R*, 1 < *R* < 13. The integers are separated by a single space.
>
> The second line contains the permutation number, given as an integer, which is used to scramble the “Team number” entries.
>
> The third line contains *P* integers, separated by single space, that describe the values for the *P* problems given in ascending order of problem numbers. Problems are numbered 1 to *P*, and their values will be less than or equal to 100.
>
> Each of the following *R* lines describes the data about one submission, which consists of four (4) integers separated by a single space. The four integers describe team number, problem number, time of submission in milliseconds from start of the five (5) hours contest, and the judge’s decision (zero for accepted, and non-zero for rejected).

The input will be terminated by a line that consists of four zeros (0 0 0 0), separated by a single space. This line should not be processed.

OUTPUT:

For each contest, the output is one line that contains the contest label and numbers of the winning team numbers (sorted in increasing order) as shown in the Example OUTPUT below.

EXAMPLE INPUT:

```
1 5 20 6
6
3 3 10 30 100
6 1 16024555 0
3 2 15895629 4
3 2 765629 0
6 4 1120132 0
3 2 1895629 3
17 3 9024555 0
2 5 20 7
120
3 10 10 30 100
10 4 2895629 0
6 1 16024555 0
6 2 15895629 5
5 2 765629 0
17 4 1120139 0
3 4 1895629 0
3 3 9024555 0
0 0 0 0
```

EXAMPLE OUTPUT:

```
Contest 1 Winner: Team 17
Contest 2 Winner: Team 5 and Team 10 and Team 17
```