The 2013 ACM ASIA Region Programming Contest

Kanpur Site

Onsite Contest Problems

Sponsored by IBM

Hosted by IIT Kanpur

13th December 2013

9 Problems

ACM Asia Regional (Kanpur Site) Programming Contest
December 13, 2013

Instructions

There are **Nine (9)** problems for each team to be completed in **Five hours**. Standard Input and Output files are to be used for each problem. If you test your program using PC^2, it will automatically redirect input from the sample input file to your program. Output must correspond exactly to the provided sample output format, including (mis)spelling and spacing. Multiple spaces will not be used in any of the judges' output, except where explicitly stated. A copy of the problem set will be available at **/users/acm/sw/problems.pdf**, during the contest. Sample input and corresponding output files for respective problems can be seen in the directories **/users/acm/sw/inputs** and **/users/acm/sw/outputs** respectively, during the contest.

Your solution to any problem should be submitted for judging using the PC^2 software only. Once you have submitted the solution, it will reach the judges. The time it takes for your problem to be judged will depend, among other things, on how busy the judges are. Once your submission has been judged, you will receive a message through PC^2 indicating the judgment. The judgment may be "Yes", meaning that your submission was judged to be correct. Otherwise you will get a message indicating the problem with your program. For example, the message may be "Incorrect Output", "Output Format Error", "Compilation Error", "Runtime Error", "Run Time Limit Exceeded" etc.

When submitting a program via PC^2, you are required to specify a primary source file and other source files (please see PC^2 documentation for details). If you are writing your programs in C or C++, please make sure that you have one primary source file from which all other source files are INCLUDED. Do not have several source files that need to be linked together. This main file in which all other files are included would be the primary source file for the program. If you are writing your programs in Java, please make sure that the name of the file containing the main class is the same as the name of the main class with a **.JAVA** suffix. This is your primary source file for the program.

You can use any of the standard library functions that your chosen programming language provides. In addition, you can use the math library in C/C++. You cannot use any other library that requires an extra flag to be passed to the compiler command. If you do this, the judges will probably find a code "compilation error" in your program. Your program is not permitted to invoke any external programs. For example, you cannot use in C the call system ("grep xyz abc") to determine whether the file abc contains an occurrence of the string xyz. *Violation of this rule may lead to disqualification from the contest.*

Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required. The judges will only test whether the input-output behavior of your program is correct or not. *If your program takes more than 2 minutes to execute for some input, it will be assumed to have gone into an infinite loop and judged incorrect.* A problem is considered as correctly solved when it is accepted by the judges. The judges are solely responsible for accepting or rejecting submitted runs. The regional contest director and judges are empowered to adjust for or adjudicate unforeseen events and conditions. Their decisions are final.

Teams are ranked according to the most problems solved. Teams who solve the same number of problems are ranked by least total time. The total time is the sum of the time consumed for each problem solved. The time consumed for a solved problem is the time elapsed from the beginning of the contest to the submittal of the accepted run plus 20 penalty minutes for every rejected run for that problem regardless of submittal time. There is no time consumed for a problem that is not solved.

No team is allowed to bring any printed materials in the contest area. Further, the team is not allowed to discuss/ talk with any other team by any means whatsoever, during the contest period. *Any such attempt, if it is detected, may lead to immediate disqualification of all the teams involved.*

Problem A
Harddisk

Dipu's harddisk crashed last week. He bought a new one and has already installed all necessary software. But he has lost his personal collection of media files. It is just unbearable! How can he avoid such a loss in future?

Dipu is planning to maintain a "fresh" second harddisk in his computer as a backup device. He wants to make sure that the backup harddisk is never too old, by replacing it with a new one every few months. This is what he will do: after a backup harddisk has been used for a few months, Dipu will buy a new harddisk, move all the data from the old backup disk to the new one and then sell the old one. He knows that he will have to sell an old backup disk at a price lower than its original price because of both depreciation and the drop in the market price of new harddisks. His goal is to keep the total cost of maintaining the backups as low as possible.

Given the prices of different harddisks in the next few months, your job is to determine the minimum cost of maintaining the backup disks in the same period of time. Assume:

- Dipu will buy and sell harddisks on the first day of a month, and he will buy a new disk and sell the old one on the same day.
- He has no backup disk to sell on the first day of the first month. He will never sell the disk used in the last month.
- The amount he will get by selling a harddisk is Tk 100 less than the *current* price of a new harddisk of the same model. Also, the price of a new harddisk never goes up, and always remains more than Tk 100. Note that Tk represents ancient unit of money (Tonka) used in this region.

Input

Input consists of several datasets. Each dataset consists of the followings:

- A line containing the name of the set (which has 2 to 16 alphanumeric characters).
- A line containing 2 positive integers n and m, denoting respectively the number of different harddisks and the number of months to consider ($1 < n < 100$, $1 < m < 200$).
- Each of next n lines contains the description of one harddisk. The description of a harddisk consists of its model name followed by m positive integers denoting its price on the first day of the m months in order. The name of a harddisk consists of 2 to 16 alphanumeric characters.

The end of input is marked with a line consisting of "TheEnd". All data on a single line are separated by exactly one space. All prices are below Tk 20000.

Output

For each set of input, the output should contain the name of the set on the first line, followed by the minimum possible cost on the second line, followed by one line for each harddisk to be used in those m months, followed by an empty line. For each harddisk, its name should be followed by the number of month(s) it will be used. See the sample output for the exact format. Obviously, the harddisks should be listed in order of their use. If there are multiple possibilities, any one of them is acceptable.

The end of output should be marked with a line consisting of "TheEnd".

Sample Input	Sample Output
FirstSet 2 4 Brand1 4000 3600 3500 3400 Brand2 3500 3500 3200 3000 SecondSet 4 8 Mintor 4500 4500 4400 4200 4000 3700 3500 3200 Rivergate 4500 4400 4300 4100 3900 3800 3600 3300 EasternDigital 5000 4900 4800 4700 4500 4100 3700 3500 WesternAnalog 7500 7200 7000 6800 6500 6200 6000 5500 TheEnd	FirstSet Tk 3400 Brand2 for 1 month(s) Brand1 for 2 month(s) Brand2 for 1 month(s) SecondSet Tk 4500 Mintor for 8 month(s) TheEnd

Problem B
Walking Path

John lives in a hilly city. He wants to go from point *A* to point *B* and he is planning to walk today. Fortunately for him, the city is designed like a grid. Unfortunately, as the elevation of each point is different, finding the easiest path is not trivial. That's why John needs your help.

John wants to find a path from *A* to *B* with minimum hardness. Hardness of going from point *A* with height *h1* to adjacent point *B* with height *h2* is

$$\begin{cases} 0.5 + 0.5 \times \sqrt{(1 + (h1 - h2)^2)} & \text{if } h1 > h2 \\ -0.5 + 1.5 \times \sqrt{(1 + (h1 - h2)^2)} & \text{Otherwise} \end{cases}$$

Distance between adjacent points is considered one unit. Slope of road segments between adjacent points is never more than 0.5.

Input

The input file starts with a number *T* (*T ≤ 10*) that represents the number of test cases. Then *T* test cases follow. Each test case starts with two number *r*, *c* (*5 ≤ r, c ≤ 100*) representing the size of the grid. Next *r* lines contain *c* numbers. j^{th} number from i^{th} line corresponds to the height of the intersection of i^{th} street with j^{th} avenue. The heights are floating point numbers in the range (*0.0 <= height < 1.0*).

Next line contains an integer q *(1 ≤ q ≤ 100)*, then *q* lines follow. Each of the following lines represents a query. A query contains four numbers: *is (1 ≤ is ≤ r)*, *js (1 ≤ js ≤ c)*, *ie (1 ≤ ie ≤ r)* and *je (1 ≤ je ≤ c)*. This represents a request to find the path with minimum hardness from intersection between is^{th} street and js^{th} avenue to the intersection between ie^{th} street and je^{th} avenue.

Output

Start each test case with case number as shown in sample output. Then output *q* lines, each for one query. For each query, output a line containing the hardness of optimal path with 6 digits after decimal point.

See sample input and output in the next page for exact formatting.

Sample Input

```
2
6 5
0.0 0.1 0.2 0.3 0.4
0.1 0.0 0.3 0.3 0.5
0.2 0.2 0.2 0.2 0.2
0.3 0.4 0.5 0.5 0.3
0.3 0.5 0.5 0.5 0.6
0.6 0.4 0.4 0.2 0.2
5
1 1 4 4
1 1 6 5
1 1 1 5
3 1 1 3
6 5 2 3
6 5
0.0 0.1 0.2 0.3 0.4
0.1 0.0 0.3 0.3 0.5
0.2 0.2 0.2 0.2 0.2
0.3 0.4 0.5 0.5 0.3
0.3 0.5 0.5 0.5 0.6
0.6 0.4 0.4 0.2 0.2
3
1 5 6 1
6 1 1 1
6 3 2 5
```

Sample Output

```
Case 1:
6.037407
9.049802
4.029925
4.009975
6.066684
Case 2:
9.074374
5.029497
6.066684
```

Problem C
Rich Rhyme

In the land of Rhyme, the King and his people always speak in rhymes. They are so accustomed in conversing using rhymes that at one point it became monotonous. So what was once a source of happiness and amusement is now creating a strange emotion within the heart of the people. The productivity of the people has become less and more importantly, the happiness index of the land has plummeted. The experts have investigated the whole matter and reported that speaking in rhyme is not the problem; it is the monotony of the same thing that is affecting the people. In fact they suggested changing the definition of rhyme slightly. One of their recommendations is to introduce a new concept of "richness" in a sentence.

A sentence would be termed rich if it starts and ends with the same word! The King seems to like this definition. But he is finding it difficult to even say a simple meaningful sentence that is "rich". The King became quite anxious. Your boss, King's ICT adviser, suggested that a competition among the general people should be thrown where the participants would have to submit a piece (story/poem/essay etc.) full of rich sentences. Then the richest pieces would be judged and be awarded. The King made a decree accordingly. Your boss was ordered to formulate a judging scheme for the pieces and write a program immediately to do the judgment. And as usual he formulated something and asked you to implement that. What he has asked you to do is as follows. You will take as input a piece as a character array. And output another array where each position will give the richness of the piece up to that position.

"What is richness?" You asked.

Your boss replied, "The richness is the size of the identical proper substring which is present at the beginning and at the end of the piece. But, the complete string cannot be considered as its substring."

"But you said to calculate the richness at each position"! You inquired.

"Yes, off course! When you calculate up to a position, you will have to assume that the piece ends at that position." Your boss smiled. He seemed to be quite happy in formulating this.

You asked again, "Don't I need to think about the word boundaries?"

"Not now, my son. We will think about it later. Do it without considering word boundaries for now. You will consider the input character by character". He assured you. He seemed all the more content with his own formulation and probably was not seeing the possible pitfalls. He continued, "And one other point; richness of the first position would always be 0."

Meaningful, you thought. But you had to ask a final question. "How would you then finally compute the richness of the total piece using all these information? What will be the formula?"

Your boss seemed perplexed. He did not think this through, as it seemed. So, he said, "You need not worry about that. Do as I said. Just remember that the input size could be quite large."

You knew that you have to stop asking and start coding now. "Yes, boss!" you said. You also realized the implication of the last direction of your boss. Few years back there was another competition on literary works. And the size of some pieces was 4 million characters!

Input

Input consists of several lines. Each line contains a string of lowercase English letters. End of input is indicated by a line containing the string “End”. No output line should be produced for this line.

Output

For each input line, you must output a line containing several integers, separated by space. The number of integers must be same as the length of the input string. The Integer at a specific position must represent the richness of the input string up to that position. There must not be any trailing space in any output line.

Sample Input

```
oebmgoca
bdblhc
jgojeqp
atofpgrwk
fdcdd
ababbacdbacdbacd
dacc
cbda
accb
dcbb
dacbbcbcbcbcdcbbbcbcbcbcbcbcdacbbcbcdcbbbcbcdcbbdcbb
End
```

Sample Output

```
0 0 0 0 0 1 0 0
0 0 1 0 0 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0
0 0 1 2 0 1 0 0 0 1 0 0 0 1 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0
```

Problem D
Family of Recurrences

Suppose we have recurrences of the following form, characterized by three parameters m, δ, f where $m > 0$ and $\delta_i \in \{0, 1\}\ \forall i \in \{0, 1, .., m\text{-}1\}$.

$$s_n = \begin{cases} f_n & \text{if } 0 \le n < m \\ \sum_{i=0}^{i=m-1} \delta_i \ s_{n-m+i} & \text{if } m \le n \end{cases}$$

Given m, n, δ, f, compute s_n *modulo* $10^9 + 7$.

Input

First line of input contains a positive integer T, then T test cases follow. For each test case, first line contains positive integer m and non-negative integer n separated by a single space. Second line contains m integers of f (f_0 to f_{m-1}) each between -10^9 and 10^9 inclusive, separated by single space. Third line contains m numbers of δ (δ_0 to δ_{m-1}) each either 0 or 1, separated by single space. Maximum value of m is 100 and maximum value of n is 10^9.

Output

For each test case, output should contain a single number s_n *modulo* $10^9 + 7$ followed by a newline.

Sample Input	Sample Output
2 2 100 1 1 1 1 3 10 1 1 1 1 1 1	782204094 193

Problem E
Chocolate Division

Alice and Bob have got some chocolates as a gift. There are 2 types of chocolates. Some chocolates have the shape of a sphere and some chocolates have the shape of a stick. Each chocolate stick connects a pair of chocolate sphere. They want to divide these chocolates and they seek your help. They want all of the chocolate spheres for themselves. Alice loves chocolate sticks. For all of the chocolate spheres she gets, each pair of them should be connected by a stick. Bob dislikes chocolate sticks. For all of the chocolate spheres he gets, no pair of them should be connected by a stick. To be clear, if any two chocolate spheres are connected by a chocolate stick initially he will take at most one of them but not both. They do not care about the sticks that connect a sphere of Alice and a sphere of Bob. You can choose to eat them. Given the description of the chocolate spheres and chocolate sticks, determine if you can divide those chocolate spheres among Alice and Bob, satisfying both of their preferences.

Input

First line of the input contains *T*, the number of test cases. Each test case starts with 2 integers *N* and *M*. *N* is the number of chocolate spheres (The chocolate spheres are numbered from 1 to *N*) and *M* is the number of chocolate sticks connecting them. Each of the next *M* lines contains 2 integers, *a* and *b*, denoting that the chocolate sphere with number *a* is connected to a chocolate sphere with number *b*. The number of chocolate spheres is between 1 and 500 inclusive and the number of chocolate sticks is between 0 and 250000. For a pair of chocolate spheres, there will be at most one stick that is connecting them.

Output

For each test case output "YES" if you can divide the chocolate spheres and "NO" otherwise.

Sample Input	Sample Output
4 4 4 1 2 1 4 2 3 3 4 5 4 1 2 1 3 1 4 1 5 10 0 3 3 1 2 1 3 2 3	NO YES YES YES

Problem F
Merging Files

In olden days, when primary memory was not as available as it is now, files were stored in external storage devices like tapes, and then two files or their parts were loaded in memory for merging. In merging two sorted files, in the worst case, number of comparisons required equals sum of lengths of the two files. Assume, in k-way (where at most k files can be merged together) merging, the number of comparisons equals sum of lengths of files to be merged. Given length of a set of files you are required to find order in which these files will have to be merged in minimum number of comparisons.

Input

The first line of input is an integer *T* (*T* <= 50), denoting the number of test cases. From the next line, test cases follow. For each case, the first line contains two integers, *N* (1 <= *N* <= 1000) and *K* (2 <= *K* <= *N*), representing respectively number of files to be merged and maximum number of files that can be merged at the same time. Next line contains *N* positive integers representing the length of each file. The maximum length of a file can be 10^8.

Output

For each test case, you must print a single line with the test case number and the minimum number of comparisons incurred to merge the files. Please refer to the sample input and output for exact formatting.

Sample Input	Sample Output
2 11 2 3 1 4 5 2 7 6 8 3 9 4 11 3 3 1 4 5 2 7 6 8 3 9 4	Case 1: 172 Case 2: 110

Problem G
Changing Money

Nikhil wants to repay his debts to Sita but would like to do so using only highest denominations possible. Once he has decided to use a denomination, he wants to use as many notes/coins of that denomination as possible. Amounts to be paid are given as numbers with two decimal places. Available denominations are given in an array.

Input

The first line of the input will contain the number of test cases, *T (1 <= T <= 50)*. The first line of each test case will contain 2 integers *N* (2 <= *N* <= 50) and *M* (2 <= *M* <= 50) denoting respectively the number of denomination values and the number of scenarios on that test case. Next line will contain the *N* denominations. Note that, a denomination can be either an integer *x* (1 <= *x* <= 10000) or a fractional number *y* (0 < *y* < 1), with exactly 2 digits after the decimal point and a zero (0) before the decimal point. It is guaranteed that there will always be a denomination of value 1 and 0.01.

Then there will be *M* numbers in the next line. Each of these numbers *z* (0 < *z* <= 10000) can either be an integer or a decimal number with exactly 2 digits after the decimal point (i.e. 100.50, 55.01). Each number represents the amount owed by Nikhil in the respective scenario.

Output

For each case the first line of output will contain “Case <case_number>:”. Then for each scenario the first line of output will be “Scenario <scenario_number>:”. Then output how Nikhil should repay his debt – In each line output two integers representing respectively denomination and number of notes/coin of that denomination, in descending order of denomination. Please represent notes as integer values and coins as decimal values with exactly 2 digits after the decimal point and a zero (0) before the decimal point.

Please refer to the sample input and output for exact formatting.

Sample Input

```
2
17 2
1000 500 200 100 50 20 10 5 2 1 0.50 0.25 0.20 0.10 0.05 0.02 0.01
5878.83
1000
3 1
1000 1 0.01
1000
```

Sample Output

```
Case 1:
Scenario 1:
1000 5
500 1
200 1
100 1
50 1
20 1
5 1
2 1
1 1
0.50 1
0.25 1
0.05 1
0.02 1
0.01 1
Scenario 2:
1000 1
Case 2:
Scenario 1:
1000 1
```

Problem H
Multi-Peg Tower of Hanoi

Tower of Hanoi problem is a classical recurrence problem that is often used to teach recurrence relations to computer science students. In the classical 3-Peg Tower of Hanoi problem initially the peg *A* contains all disks top to bottom in a strictly small to big order. This tower of disks is to be moved to peg *C* using peg *B* as intermediate. The rule of the game is that in every move the top disk from one of the towers can be moved to the top of another tower. Never ever can we put a large disk on top of a smaller one. In this problem, in order to move a tower of *n* disks, we must shift the top sub-tower of *n* - 1 disks to the only intermediate peg, then shift the largest to peg *C* and then the sub-tower of size *n* - 1 from intermediate to destination peg. Whatever the value of *n* may be exactly *n* - 1 disks have to be shifted to the only intermediate peg. We have no choice of fewer or more disks, and no choice for the intermediate peg.

In multi-Peg Tower of Hanoi problem we have *p* >= 3 pegs (P_1, P_2, … P_p) and *n* disks to move from P_1 to P_p using minimum number of moves. We refer to this as (*n*, *p*) system. Now we have *p* - 2 intermediate pegs and not all of *n* - 1 disks need to be shifted to one intermediate peg. However, before the largest disk moves to the destination peg, all top *n* - 1 disks have to be distributed among all intermediate pegs. While the optimal strategy still baffles algorithm designers, many believe that in the optimal solution, certain optimal number n_1 disks have to be shifted to an intermediate peg. Then the remaining $n - n_1$ disks must reach destination, while the disks in the intermediate peg will remain untouched. Then the disks in the intermediate peg must reach destination. In transferring any sub-tower to any peg, this strategy will be followed recursively. Let *M*(*n*, *p*) be the minimum number of moves required to shift a tower of *n* disks in *p* peg system. This strategy satisfies the following recurrence relations:

$$M(n,p) = \min_{0<n_1<n} (2M(n_1,p) + M(n-n_1,p-1))$$

Assume further that, if multiple strategies result in minimum number of moves, then the strategy for which n_1 is the smallest is chosen. Your job is to find out the move number at which a particular disk *D* moves ultimately to the destination peg in an (*n*, *p*) system.

Input

The first line of input will be an integer *T* (1 <= *T* <= 10000), denoting the number of test cases. Each test case is represented by a triple of integers *N* (1 <= *N* <= 50), *P* (3 <= *P* <= 50) and *K* (1 <= *K* <= *N*) in the same line separated by space. *N* denotes the number of disks. *P* denotes the number of pegs. *K* denotes the index (1-based from the top) of the disk, whose move number need to be found out.

Output

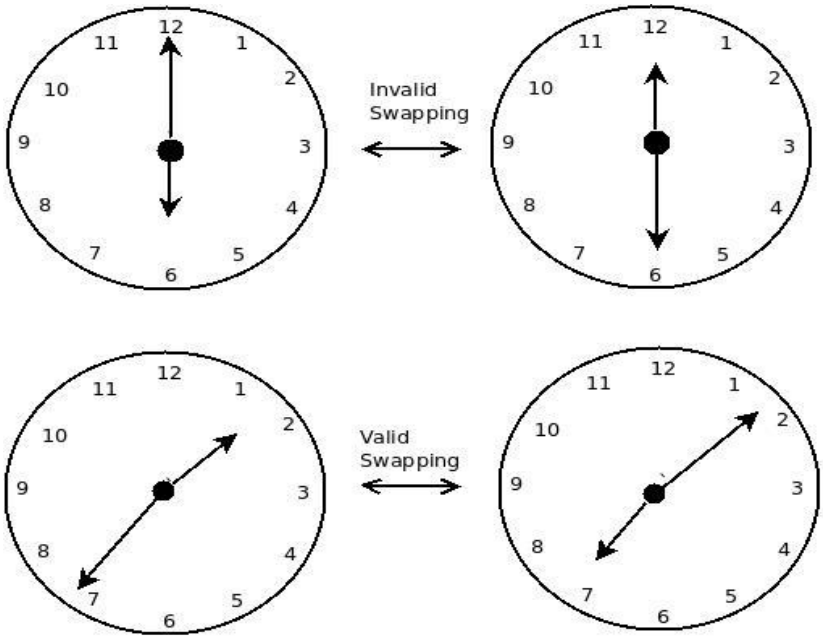
For each test case, there should be a single line of output. The line should contain “Case <case_number>: ”, followed by an integer representing the move number at which disk *K* moves ultimately to the destination peg. Refer to the sample input and output for exact formatting.

Sample Input	Sample Output
3 3 3 2 50 3 1 50 50 1	Case 1: 6 Case 2: 1125899906842623 Case 3: 101

Problem I
Rewarding the Priests

Emperor of the galaxy has a number of planets where His common citizens live and each planet runs a different clock system. However, the clock has three hands- hour, minute and second. *H* hours make a day, *M* minutes make an hour and *M* seconds make a minute, where values of *H* and *M* can be different for different planets; however they are positive integers. The Emperor is a lazy person and doesn't like to see the hand corresponding to second moving in the clock. That's why all the clocks in his empire have only two moving hands - hour and minute.

There is a priest in each of the planets. Each priest spends some time in Emperor's Hall and gets a number of gold coins for his services. The number of gold coins is equal to the number of times swapping hour and minute hands that represents a "valid time", during his interval of stay. Examples of valid and invalid time are given below.



Start and finish time of staying of a priest are given in hours, minutes and seconds, all in non-negative integers and fraction of seconds expressed in lowest terms. Start and finish time are provided in two successive rows. Stay of a priest will never exceed *H* hours. You need to calculate number of gold coins that each priest will get.

Input

The first line of input will be a positive integer *T* (*T* <= 50), denoting the number of test cases. For each test case, there will be 3 lines of input. The first line of each test case contains 2 integers *H* and *M* (both are lying between 2 and 100; both inclusive), separated by a space. The next line contains 5 integers, *h* (1 <= *h* <= *H*), *m* (0 <= *m* < *M*), *s* (0 <= *s* < *M*), *sn* (0 <= *sn* <= 10000), *sd* (1 <= *sd* <= 10000) respectively denoting hour, minute, integer second and numerator and denominator of the fractional parts of the second. These integers represent the start time. The next line represents finish time. It also contains 5 integers in the same format as previous line. The start and finish time are inclusive.

Output

Output for each case will be an integer representing the number of gold coins that the priest will get from the Emperor for his service. The format is: “Case <case_number>: <output>”. Please refer to the sample input and output for exact formatting.

Sample Input	Sample Output
3 12 60 12 0 0 0 1 11 59 59 999 1000 12 60 12 5 2 14 143 12 10 4 28 143 84 87 10 25 17 10 21 73 18 28 3747 5038	Case 1: 143 Case 2: 2 Case 3: 5284