ADA - Análisis y Diseño de Algoritmos, 2019-2
Tarea 2: Semanas 3 y 4

Para entregar el viernes 16 de agosto/domingo 18 de agosto de 2019

Problemas conceptuales a las 16:00 (16 de agosto) en el Departamento de Electrónica y Ciencias de la Computación

Problemas prácticos a las 23:59 (18 de agosto) en la arena de programación

---

Tanto los ejercicios como los problemas deben ser resueltos, pero únicamente las soluciones de los problemas deben ser entregadas. La intención de los ejercicios es entrenarlo para que domine el material del curso; a pesar de que no debe entregar soluciones a los ejercicios, usted es responsable del material cubierto en ellos.

Instrucciones para la entrega

Para esta tarea y todas las tareas futuras, la entrega de soluciones es *individual*. Por favor escriba claramente su nombre, código de estudiante y sección en cada hoja impresa entregada o en cada archivo de código (a modo de comentario). Adicionalmente, agregue la información de fecha y nombres de compañeros con los que colaboró; igualmente cite cualquier fuente de información que utilizó.

¿Cómo describir un algoritmo?

En algunos ejercicios y problemas se pide "dar un algoritmo" para resolver un problema. Una solución debe tomar la forma de un pequeño ensayo (es decir, un par de párrafos). En particular, una solución debe resumir en un párrafo el problema y cuáles son los resultados de la solución. Además, se deben incluir párrafos con la siguiente información:

- una descripción del algoritmo en castellano y, si es útil, pseudo-código;
- por lo menos un diagrama o ejemplo que muestre cómo funciona el algoritmo;
- una demostración de la corrección del algoritmo; y
- un análisis de la complejidad temporal del algoritmo.

Recuerde que su objetivo es comunicar claramente un algoritmo. Las soluciones algorítmicas correctas y descritas *claramente* recibirán alta calificación; soluciones complejas, obtusas o mal presentadas recibirán baja calificación.

Ejercicios

4.1-1, 4-1-2, 4.1-3 (página 74), **4-2** (página 107), 15.1-2, 15.1-3, 15.1-4, 15.1-5 (página 370).

Problemas conceptuales

Resuelva dos de los siguientes tres problemas (indique claramente su selección de los dos problemas):

1. Ejercicio 6: *String Shuffling* (Erickson, página 126).
2. Ejercicio 2: *Computer Hackers* (Kleinberg & Tardos, página 313).
3. Ejercicio 23: *Minimum Cost to a Sink* (Kleinberg & Tardös, página 331).

Problemas prácticos

Hay cinco problemas prácticos cuyos enunciados aparecen a partir de la siguiente página.

A - Train Sorting

Source file name: `train.py`
Time limit: 1 second

Erin is an engineer. She drives trains. She also arranges the cars within each train. She prefers to put the cars in decreasing order of weight, with the heaviest car at the front of the train.

Unfortunately, sorting train cars is not easy. One cannot simply pick up a car and place it somewhere else. It is impractical to insert a car within an existing train. A car may only be added to the beginning and end of the train.

Cars arrive at the train station in a predetermined order. When each car arrives, Erin can add it to the beginning or end of her train, or refuse to add it at all. The resulting train should be as long as possible, but the cars within it must be ordered by weight.

Given the weights of the cars in the order in which they arrive, what is the longest train that Erin can make?

Input

The first line is the number of test cases to follow. The test cases follow, one after another; the format of each test case is the following: the first line contains an integer $0 \leq n \leq 2000$, the number of cars. Each of the following n lines contains a non-negative integer giving the weight of a car. No two cars have the same weight.

The input must be read from standard input.

Output

Output a single integer giving the number of cars in the longest train that can be made with the given restrictions.

The output must be written to standard output.

Sample Input	**Sample Output**
1 3 1 2 3	3

B - Maximum Sub-sequence Product

Source file name: `product.py`
Time limit: 1 second

Bob needs money, and since he knows you are here, he decided to gamble intelligently. The game is rather simple: each player gets a sequence of integers. The players must determine, by using their mega-pocket computers, which is the maximum product value which can be computed with non empty sub-sequences of consecutive numbers from the given sequence. The winner is the one which gets first the right result.

Can you help Bob with a program to compute quickly the wanted product, particularly when the sequence is quite long?

Input

The input contains sequences of numbers. Each number will have at most 5 digits. There will be at most 100 numbers in each sequence. Each sequence starts on a new line and may continue on several subsequent lines. Each sequence ends with the number −999999 (which is not part of the sequence).

The input must be read from standard input.

Output

The maximum sub-sequence product for each sequence must be written on the standard output, on a different line.

The output must be written to standard output.

Sample Input	**Sample Output**

`1 2 3 -999999`
`-5 -2 2 -30 -999999`
`-8 -999999`
`-1 0 -2 -999999`
	`6`
	`120`
	`-8`
	`0`

C - Flight Planner

Source file name: `flight.py`
Time limit: 1 second

Calculating the minimal cost for a flight involves calculating an optimal flight-altitude depending on wind-strengths changing with different altitudes. It's not enough just to ask for the route with optimal wind-strength, because due to the mass of a plane you need a certain amount of fuel to rise. Moreover due to safety regulations it's forbidden to fly above a certain altitude and you can't fly under zero-level.

In order to simplify the problem for now, we assume that for each 100 miles of flight you have only three possiblities: to climb one mile, to hold your altitude or to sink one mile. Climb flight requires 60 units of fuel, holding your altitude requires 30 units, and sinking requires 20 units.

In the case of headwind you need more fuel while you can save fuel flying with tailwind. Windstrength w will satisfy the condition $-10 \leq w \leq 10$, where negative windstrength is meant to be headwind and positive windstrength is tailwind. For one unit of tailwind you can save one unit of fuel each 100 miles; each unit of headwind will cost an extra unit of fuel. For example to climb under conditions of windstrength $w = -5$, you need 65 units of fuel for this 100 miles.

Given the windstrengths on different altitudes for a way from here to X, calculate the minimal amount of fuel you need to fly to X.

Input

The first line of the input contains the number N of test cases in the file. The first line of each test case contains a single integer X, the distance to fly, with $1 \leq X \leq 100\,000$ miles and X is a multiple of 100. Notice that it's not allowed to fly higher than 9 miles over zero and that you have to decide whether to climb, hold your altitude or to sink only for every 100 miles. For every mile of allowed altitude (starting at altitude 9 down to altitude 0) there follow $X/100$ windstrengths, starting with the windstrength at your current position up to the windstrength at position $X - 100$ in steps of 100 miles. Test cases are separated by one or more blank lines.

The input must be read from standard input.

Output

For each test case output the minimal amount of fuel used flying from your current position (at altitude 0) to X (also at altitude 0), followed by a blank line.

The output must be written to standard output.

Sample Input	**Sample Output**
2 400 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 9 9 1 1 -9 -9 1 1000 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 7 7 7 7 7 7 7 7 7 7 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -7 -3 -7 -7 -7 -7 -7 -7 -7 -7 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9	120 354

D - Term Strategy

Source file name: `term.py`
Time limit: 1 second

Student Peter was playing billiard all the term and for that reason he missed all his lectures. Unfortunately that's the time for term exams which Peter must pass.

Peter has N ($1 \leq N \leq 10$) courses, because he completely failed the previous term exams so he has to pass them too. He is already late with his exams and the deadline is coming. He agreed with all teachers to pass all the exams in one day. The idea he followed is –"Everything or nothing". After he made some considerations and an assumption that we'll play billiard only 5 hours a day, he counted the number of hours he has left to get himself prepared for exams. He has concluded that he has M ($0 < M \leq 100$) hours to get prepared. While he is preparing he has to read at least the name of the subject he is preparing for. Because of this difficult task he's preparing for at least one hour for each exam. If he has no time to prepare for exam –he fails it.

Exam is evaluated using 10-grade system. To pass an exam Peter has to receive at least 5. Also Peter is a greedy person and he wants to have the highest average mark that is possible.

As Peter is perfectly aware of his abilities he made a table L. Each line of this table represents a course and each column represents a number of hours Peter is preparing for a particular exam. The number L_{ij} denotes the grade Peter expects for the exam of course i by using j hours of his available time, with $1 \leq i \leq N$ and $1 \leq j \leq M$.

Suddenly Peter remembers that he has a good friend. You're actually the person he remembered. So as he is completely desperate he asks for your help! Help Peter in his mission of passing *all* exams and achieving the maximum possible average mark.

Input

The number of tests T ($T \geq 1$) is given on the first line of input. First line of each test contains two integers N and M. Then N lines follow each containing M integers defining the L_{ij} values.

The input must be read from standard input.

Output

For each test case find whether Peter passes all exams. If he is that lucky to pass them, print the line '`Maximal possible average mark - S.`', where S is the highest average possible mark. In case when he is not that lucky and he has at least one exam that he's unable to pass in the given time limits, print a line '`Peter, you shouldn't have played billiard that much.`'.

The output must be written to standard output.

Sample Input	**Sample Output**
2 4 5 5 5 6 7 8 5 5 6 7 8 5 6 7 8 8 6 7 8 9 9 4 5 4 5 6 7 8 4 5 6 7 8 5 6 7 8 8 6 7 8 9 9	Maximal possible average mark - 5.50. Peter, you shouldn't have played billiard that much.

E - Exact Change

Source file name: `exact.py`
Time limit: 4 seconds

- Seller: That will be fourteen dollars.
- Buyer: Here's a twenty.
- Seller: Sorry, I don't have any change.
- Buyer: OK, here's a ten and a five. Keep the change.

When travelling to remote locations, it is often helpful to bring cash, in case you want to buy something from someone who does not accept credit or debit cards. It is also helpful to bring a variety of denominations in case the seller does not have change. Even so, you may not have the exact amount, and will have to pay a little bit more than full price. The same problem can arise even in urban locations, for example with vending machines that do not return change.

Of course, you would like to minimize the amount you pay (though you must pay at least as much as the value of the item). Moreover, while paying the minimum amount, you would like to minimize the number of coins or bills that you pay out.

Input

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing an integer, the price in cents of the item you would like to buy. The price will not exceed 10 000 cents (i.e., $100). The following line contains a single integer n, the number of bills and coins that you have. The number n is at most 100. The following n lines each contain one integer, the value in cents of each bill or coin that you have. Note that the denominations can be any number of cents; they are not limited to the values of coins and bills that we usually use in Canada. However, no bill or coin will have a value greater than 10 000 cents ($100). The total value of your bills and coins will always be equal to or greater than the price of the item.

The input must be read from standard input.

Output

For each test case, output a single line containing two integers: the total amount paid (in cents), and the total number of coins and bills used.

The output must be written to standard output.

Sample Input	**Sample Output**
1 1400 3 500 1000 2000	1500 2