ADA - Análisis y Diseño de Algoritmos, 2017-2
Tarea 3: Sesiones 9-14

Para entregar el viernes 08 de septiembre/lunes 11 de septiembre de 2017

Problemas conceptuales a las 16:00 (08 de septiembre) en el Departamento de Electrónica y Ciencias de la Computación

Problemas prácticos a las 23:59 (11 de septiembre) 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

16.1-1, 16.1-2, 16.1-3, 16.1-4, 16.1-5 (página 422), 16.2-1, 16.2-3, 16.2-5, 16.2-7 (páginas 427 y 428),

Problemas conceptuales

1. Ejercicio 7.4: *Stabs* (Erickson página 263).
2. Ejercicio 7.8: *Hybrid Car* (Erickson página 264).
3. Ejercicio 7.9: *Parentheses Balance* (Erickson página 265).

Problemas prácticos

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

A - Ants

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

An army of ants walk on a horizontal pole of length *l* cm, each with a constant speed of 1 cm/s. When a walking ant reaches an end of the pole, it immediatelly falls off it. When two ants meet they turn back and start walking in opposite directions. We know the original positions of ants on the pole, unfortunately, we do not know the directions in which the ants are walking. Your task is to compute the earliest and the latest possible times needed for all ants to fall off the pole.

Input

The first line of input contains one integer giving the number of cases that follow. The data for each case start with two integer numbers: the length of the pole (in cm) and *n*, the number of ants residing on the pole. These two numbers are followed by *n* integers giving the position of each ant on the pole as the distance measured from the left end of the pole, in no particular order. All input integers are not bigger than 1 000 000 and they are separated by whitespace.

The input must be read from standard input.

Output

For each case of input, output two numbers separated by a single space. The first number is the earliest possible time when all ants fall off the pole (if the directions of their walks are chosen appropriately) and the second number is the latest possible such time.

The output must be written to standard output.

Sample Input	**Sample Output**

`2`
`10 3`
`2 6 7`
`214 7`
`11 12 7 13 176 23 191`
	`4 8`
	`38 207`

B - Bit Mask

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

In bit-wise expression, *mask* is a common term. You can get a certain bit-pattern using mask. For example, if you want to make first 4 bits of a 32-bit number zero, you can use `0xFFFFFFF0` as mask and perform a bit-wise AND operation. Here you have to find such a bit-mask.

Consider you are given a 32-bit unsigned integer N. You have to find a mask M such that $L \leq M \leq U$ and N OR M is maximum. For example, if N is 100 and $L = 50$, $U = 60$ then M will be 59 and N OR M will be 127 which is maximum. If several value of M satisfies the same criteria then you have to print the minimum value of M.

Input

Each input starts with 3 unsigned integers N, L, U where $L \leq U$.

The input must be read from standard input.

Output

For each input, print in a line the minimum value of M, which makes N OR M maximum.

The output must be written to standard output.

Sample Input	Sample Output
`100 50 60`	`59`
`100 50 50`	`50`
`100 0 100`	`27`
`1 0 100`	`100`
`15 1 15`	`1`

C - Commando War

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

There is a war and it doesn't look very promising for your country. Now it's time to act. You have a commando squad at your disposal and planning an ambush on an important enemy camp located nearby. You have N soldiers in your squad. In your master-plan, every single soldier has a unique responsibility and you don't want any of your soldier to know the plan for other soldiers so that everyone can focus on his task only. In order to enforce this, you brief every individual soldier about his tasks separately and just before sending him to the battlefield. You know that every single soldier needs a certain amount of time to execute his job. You also know very clearly how much time you need to brief every single soldier. Being anxious to finish the total operation as soon as possible, you need to find an order of briefing your soldiers that will minimize the time necessary for all the soldiers to complete their tasks. You may assume that, no soldier has a plan that depends on the tasks of his fellows. In other words, once a soldier begins a task, he can finish it without the necessity of pausing in between.

Input

There will be multiple test cases in the input file. Every test case starts with an integer N ($1 \leq N \leq 1\,000$), denoting the number of soldiers. Each of the following N lines describe a soldier with two integers B ($1 \leq B \leq 10\,000$) and J ($1 \leq J \leq 10\,000$). B seconds are needed to brief the soldier while completing his job needs J seconds. The end of input will be denoted by a case with $N = 0$. This case should not be processed.

The input must be read from standard input.

Output

For each test case, print a line in the format, '`Case X : Y`', where X is the case number and Y is the total number of seconds counted from the start of your first briefing till the completion of all jobs.

The output must be written to standard output.

Sample Input

```
3
2 5
3 2
2 1
3
3 3
4 4
5 5
0
```

Sample Output

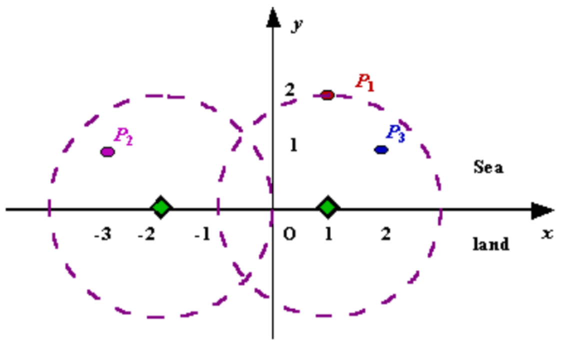
```
Case 1: 8
Case 2: 15
```

D - Radar Installation

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

Assume the coasting is an in nite straight line. Land is in one side of coasting, sea in the other. Each small island is a point locating in the sea side. And any radar installation, locating on the coasting, can only cover d distance, so an island in the sea can be covered by a radius installation, if the distance between them is at most d.

We use Cartesian coordinate system, de ning the coasting is the x-axis. The sea side is above x-axis, and the land side below. Given the position of each island in the sea, and given the distance of the coverage of the radar installation, your task is to write a program to find the minimal number of radar installations to cover all the islands. Note that the position of an island is represented by its x-y coordinates.



Input

The input consists of several test cases. The first line of each case contains two integers n ($1 \leq n \leq 1\,000$) and d, where n is the number of islands in the sea and d is the distance of coverage of the radar installation. This is followed by n lines each containing two integers representing the coordinate of the position of each island. Then a blank line follows to separate the cases. The input is terminated by a line containing pair of zeros.

The input must be read from standard input.

Output

For each test case output one line consisting of the test case number followed by the minimal number of radar installations needed. '`-1`' installation means no solution for that case.

The output must be written to standard output.

Sample Input	Sample Output

```
3 2
1 2
-3 1
2 1

1 2
0 2

0 0
```
	```
	Case 1: 2
	Case 2: 1
	```