ADA - Análisis y Diseño de Algoritmos, 2019-2
Tarea 3: Semanas 6 y 7

Para entregar el martes 10 de septiembre/lunes 9 de septiembre de 2019

Problemas conceptuales a las 9:00 (10 de septiembre) en el Departamento de Electrónica y Ciencias de la Computación

Problemas prácticos a las 23:59 (09 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 4-1: *Greedy Schedule* (Erickson, página 176).
2. Ejercicio 4.3: *Stabbing Points* (Erickson, página 177).

Problemas prácticos

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

A - Setting Problems

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

As you see, setting problems for a programming contest is a tough job. There are so many things to do like creating problems, solutions, data files, verification of problem statements, writing alternate judge solutions etc.

Given the responsibility of creating the problemset for a programming contest, Sultan and GolapiBaba have realized that special measures need to be taken: they have agreed to set N problems for the contest. For each of the problems, first Sultan will create the problem statement, solution, and i/o data. After he finishes his work, GolapiBaba does the verification and alternate solution writing part for that particular problem. Each of them needs a particular amount of time to complete their tasks for any given problem. Also, none of them works on more than one problem at a time. Note that, GolapiBaba can start working on a problem immediately after Sultan finishes that problem or he may wish to start that problem later.

You will be given the times that Sultan and GolapiBaba requires to complete their respective tasks for every single problem. Determine the minimum possible time required to complete the whole problemset.

Input

There are many test cases. Each test case starts with a single integer N ($1 \leq N \leq 20\,000$), the number of problems in the contest. The next line contains N integers S_i ($1 \leq S_i \leq 100$, $1 \leq i \leq N$) where S_i denotes the time required for Sultan to complete his tasks for problem i. The next line has N more integers G_i ($1 \leq G_i \leq 100$, $1 \leq i \leq N$) where G_i denotes the time required for Golapibaba to complete his tasks on problem i.

The input must be read from standard input.

Output

For each test case, output the minimum time required to complete the problemset.

The output must be written to standard output.

Sample Input	**Sample Output**
3 8 1 6 1 6 3 3 4 5 6 1 1 6	16 16

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`
`100 50 50`
`100 0 100`
`1 0 100`
`15 1 15`
	`59`
	`50`
	`27`
	`100`
	`1`

C - Moliu Number Generator

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

Let's play a number game. We start with $N = 0$ and we want to make N equal to a given integer S. Only three types of operations are allowed in this process:

INC increment N by 1, i.e., $N \leftarrow N + 1$.

DEC decrement N by 1, i.e., $N \leftarrow N - 1$.

DBL double N, i.e., $N \leftarrow 2 \times N$.

Of course, we want to make $N = S$ with the minimum number of operations. Consider the following example with S = 7 in which only 5 steps are required, for instance:

INC $N = 0 + 1 = 1$.

INC $N = 1 + 1 = 2$.

DBL $N = 2 \times 2 = 4$.

DBL $N = 2 \times 4 = 8$.

DEC $N = 8 - 1 = 7$ ... DONE!!

Input

The input contains several test cases. Each test case comprises a line containing one integer S ($0 \leq S <= 2^{31}$).

The input must be read from standard input.

Output

For each S, output the minimum number of operations required to make $N = S$.

The output must be written to standard output.

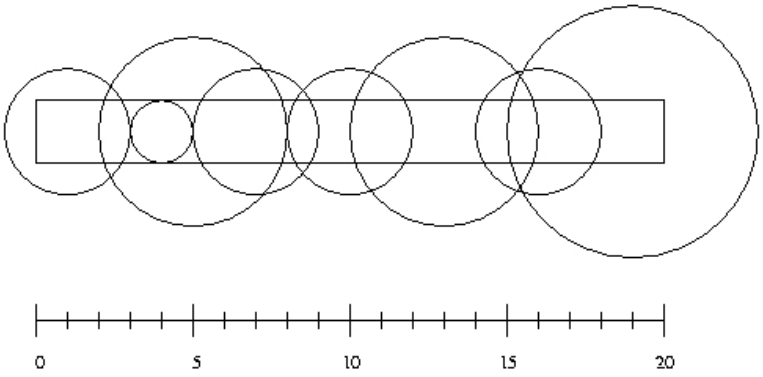
Sample Input	**Sample Output**
7	5

D - Watering Grass

Source file name: `grass.py`
Time limit: x seconds

Consider n sprinklers that are installed in a horizontal strip of grass l meters long and w meters wide. Each sprinkler is installed at the horizontal center line of the strip. For each sprinkler we are given its position as the distance from the left end of the center line and its radius of operation.

What is the minimum number of sprinklers to turn on in order to water the entire strip of grass?



Input

Input consists of a number of cases. The first line for each case contains integer numbers n, l, and w with $n \leq 10\,000$. The next n lines contain two integers giving the position of a sprinkler and its radius of operation. (The picture above illustrates the first case from the sample input.)

The input must be read from standard input.

Output

For each test case output the minimum number of sprinklers needed to water the entire strip of grass. If it is impossible to water the entire strip, output -1.

The output must be written to standard output.

Sample Input	**Sample Output**
8 20 2 5 3 4 1 1 2 7 2 10 2 13 3 16 2 19 4 3 10 1 3 5 9 3 6 1 3 10 1 5 3 1 1 9 1	6 2 -1