SUM

Given two integers, we consider all integers between them, including both given ones. Let us make the sum of these numbers.

Task

Write a program that computes the above described sum.

Input

Two integers *a* and *b*, separated by a space

Output

One integer, that is equal to the desired sum.

Constraints

$-1000 < a < 1000$
$-1000 < b < 1000$

Example

Input

```
2 4
```

Output

```
9
```

GUESS THE NUMBER

You play a game against jury's program. Your program must guess a number *N*, chosen by the jury's program at the start of the game. The rules are as follows: your program determines a number and sends it to jury's program. Then you get a response by the jury that can be one of three possibilities – "This is the number", "Your number is smaller than jury's number" or "Your number is greater than jury's number". If you didn't guess, then the process repeats. When you are sure you know the jury's number, you have to send your answer to the jury and the game is over.

Task

Write a function guess(), which will be compiled with the jury's program. This function will maintain the dialog with jury's program until the number is guessed.

Implementation details

You are given the following two functions:

int quest(long long n);

and

void res(long long answer);

By calling function **quest** your program asks jury's program about a number, passed as parameter n. The function then returns one of the following three results:

0 – this is jury's number
1 – your number is smaller than jury's number
2 – your number is greater than jury's number

This function is to be called multiple times until your program decides that it's ready with the answer.

By calling function **res** your program should pass the guessed number as a value of parameter answer. This function should be called only once during the execution of your program – it will finish the game.

You should submit to the grading system a file **guess.cpp,** which contains a function *void guess()*. It may contain other code, necessary for correct execution of the program, but should not contain *main()*.

At the beginning your file must contain #include “guess.h”

Constraints

$1 \le N \le 10^{18}$

Example

Let 5 be the number of the jury. The dialogue between your program and jury’s program can look like the following one:

guess calls function:	The function returns:
quest(1)	1
quest(2)	1
quest(3)	1
quest(4)	1
quest(5)	0
res(5)	

Grading

If your program cannot guess the number within the given time limit or if the answer is wrong, you will receive 0 points for the test case. If you guess the number with Q questions, you will receive P=min(10,[(Q_{min}/Q)*10]) points, where: [x] is the integer part of x and Q_{min} is the minimum count of questions, by which is guaranteed that any number in the considered interval can be guessed.

Local testing

In order to be able to test your function *guess()* locally, you will get files *Lgrader.cpp* and *guess.h*. Save them in your working folder and compile them together with your file **guess.cpp.** In this way, you will make a program that reads a number from the standard input and calls *guess()*. On the standard output the program will print out the number that your function finds to be the answer, and also prints out the count of questions that were used during the process.

LAMPS

The street on which Elly lives is a row of **N** 1-by-1 meter tiles. This means the street's length is **N** meters, and its width is one meter. As you will see, the width is irrelevant for this problem.

There is a street lamp in the center of each tile. Each lamp provides light with certain strength (intensity). The strengths of the lamps are the integers $\mathbf{S_1}$, $\mathbf{S_2}$, …, $\mathbf{S_N}$.

Recently the city management decided to save some energy by turning off some of the lamps. The mayor is concerned whether this won't make the street too dark, so she wants the light reaching tile *i* to be some integer $\mathbf{R_i}$. The money that the city will save by turning off the lamp at tile *i* is $\mathbf{M_i}$ leva (the Bulgarian currency).

As you may know, the light dissipates quickly. For the purposes of this task we will assume that it loses half of its strength for every meter traveled (this is a special light). Furthermore, for simplicity we will ignore the lamp's height and consider them to be on the ground level.

With this setup, the lamp at position *i* provides $\mathbf{S_i}$ units of light to its tile, but $\mathbf{S_i}/2$ (rounded down) to the tiles at positions *i-1* and *i+1*; $\mathbf{S_i}/4$ (rounded down) to the ones at *i-2* and *i+2*, and so on. In general, the lamp at position *i* provides $\mathbf{S_i}/2^{|i-j|}$ (rounded down) units of light to the tile at position *j*. The total amount of light for tile *i* is the sum of all amounts of light reaching it.

Task

You decide to help preserving the Earth's natural resources by proposing a plan to the city council that is most energy-efficient and still leaves the street lit well enough. You now need to find which lamps to be turned off, such that the amount of light reaching each tile *i* be at least $\mathbf{R_i}$ and in the same time the total saved money (the sum of $\mathbf{M_i}$ for the lamps you are turning off) is as high as possible. Write a program that returns this maximal sum.

Input

On the first line will be given the integer **N** – the number of lamps on the street. On each of the following **N** lines will be given three integers $\mathbf{S_i}$ $\mathbf{R_i}$ $\mathbf{M_i}$ – the strength of the lamp's light, the minimum required light reaching this tile, and the saved money by turning it off.

Output

Print one integer on a single line of the standard output – the maximum amount of money that can be saved by turning off lamps, such that the street is still lit well-enough. If the requirements cannot be fulfilled even if no lamp is turned off, print "-1" instead.

Constraints

$1 \leq \mathbf{N} \leq 100$
$1 \leq \mathbf{S}_i, \mathbf{R}_i, \mathbf{M}_i \leq 200$

Examples

Input

```
7
13 3 12
7 6 10
10 12 9
10 15 11
8 10 13
11 5 12
9 2 7
```

Output

```
42
```

Input

```
4
7 9 42
10 20 42
8 9 42
9 11 42
```

Output

```
-1
```

Explanation

In the first example one possible answer is the set of lamps {1, 2, 5, 7}, which saves 12 + 10 + 13 + 7 = 42 leva. You can check that the remaining lamps (on tiles {3, 4, 6}) provide enough light to each of the seven tiles. For example, the fifth tile receives 2 units of light from lamp three, five units from lamp 4, and 5 units from lamp 6. In total, it receives 12 units of light, when 10 are needed.

In the second example even if all lamps are lit, the one at the position two will receive a total of 19 units – one less than required for this position.