1 Minimum Daily Accomplishment Quotient

A toy maker's primary job is to make toys based on a predefined plan. He receives orders to make toys. Each order indicates number of toys to be delivered. Imagine orders are stored in *orders[].* The orders must be accomplished in the same order of receiving them.

In *d* days he must accomplish all orders. He must work on at least one order every day. He must ensure to see that before taking up order[i], he must have finished all orders[j], where 0<=j<i.

Daily Order Accomplishment Factor (DOAF) is the maximum number of toys manufactured on a day from one (or many) orders chosen on that particular day.

Given the number of days, array of orders, find the minimum sum of Daily Order Accomplishment Factors. Return -1 if determining OAQ is not possible.

Input/Output

Input	Output	Comments
1 10 2 20 3 30 4 40 **6**	**76**	• First line **1 10 2 20 3 30 4 40** represents orders received • Second line **6 r**epresents the number of days to accomplish all orders. Explanation • First day he can finish the 1st order, DOAF = 1. • Second day you can finish the 2nd order, DOAF = 10 • Third day you can finish the 3rd order, DOAF = 2 • Fourth day you can finish the 4th order, DOAF = 20 • Fifth day you can finish the 5th order, DOAF = 3 • Sixth day you can finish the 6th, 7th and 8th orders. Maximum number of toys made = DOAF = 40 • Sum of DOAFs = 1 + 10 + 2 + 20 + 3 + 40 = **76**
3 2 1 4 **5**	**-1**	• There are four orders to accomplish but 5 days to finish. Even if he chooses to accomplish one order per day, on fifth day he will be idle. That is not acceptable. Hence DOAF is -1

2 Fencing Critical Military Patch

Around a ***Military Cantonment Area (MCA),*** several residential colonies came up and the military thinks it could potentially breach their security. So, it came up with a plan to fence its area. Imagine the whole area mapped into N x N matrix. Military Area is represented as 1 while the residential area as 0. So, each cell of the matrix is either 1 or 0. Each military area (cell of the grid) is a square of length 1 unit and the cost of fencing it is USD 1 per 1 unit length.

Critical Military Patch (CMP) is a collection of connected military areas (1) horizontally or vertically. Diagonally connected areas are not considered. ***CMP*** always surrounded by residential areas (0) and there could be more than one ***CMP***s separately or joined together.

Given the matrix of the ***MCA***, find the total cost of the fencing the CMPs identified.

Input/Output

<table>
<tr><th>Input</th><th>Output</th><th>Comments</th></tr>
<tr><td>3 3
0 1 0
0 1 0
0 1 0</td><td>8</td><td>

- First line **3 3** corresponds to the size of the grid.
- Next three lines corresponds to the composition of the grid.
  0 – Residential Area, 1 – Military Area

<table>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
</table>

- The shaded area in red forms the Critical Military Patch.
- Only one CMP with 3 squares. Fencing needed for 8 units (8 unit lengths).
- The cost of fencing CMP is 8 * 1 USD = **8 USD**

</td></tr>
<tr><td>4 4
0 0 0 0
0 0 0 0
1 1 0 0
0 0 0 1</td><td>10</td><td>

<table>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td></tr>
</table>

There are two CMPs.

- Cost of fencing 1st CMP (2 squares, fencing needed for 6 unit lengths)
  = 6 * 1 USD = **6 USD**
- Cost of fencing 2nd CMP (1 square, fencing needed for 4 unit lengths)
  = 4 * 1 USD = **4 USD**
- Total cost of fencing = (6 + 4) = **10 USD.**

</td></tr>
</table>

3 Ball Bouncing Game

Ball Bouncing is a kids' game that displays a bunch of buildings and the ball bounces from one building to another based on its ***Maximum Bouncing Power (MBP). MBP*** is the maximum number of buildings that the ball can bounce at a given time.

Assume that the heights of the buildings are given as an integer array, ***buildings[]***. The game has a few rules in order for the ball to bounce from one building to another.

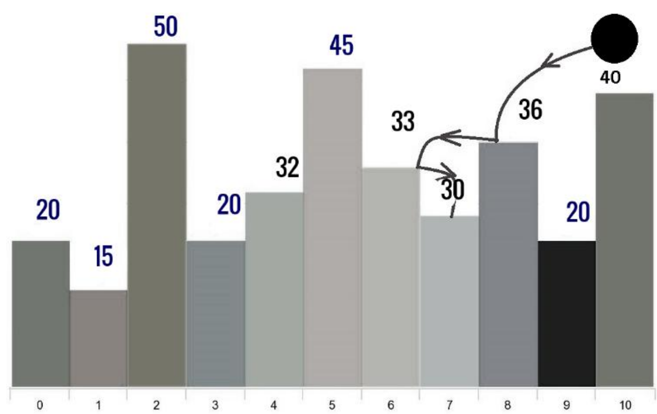
Here are the rules for each bounce:

1. For bouncing from *buildings[k]* to another *buildings[k + m]*. the condition is **k + m < *buildings[].length* and 0 < k <= *MBP***.
2. For bouncing from *buildings[k]* to another *buildings[k - m]*. the condition is **k - m >=0 and 0 < k <= *MBP***
3. Bouncing from index **p** to **q** is possible only if
    a. buildings[**p**] > buildings[**q**] and
    b. buildings[**p**] > buildings[**r**] for all indices of **r** between **p** and **q**
4. **Never cross the boundaries of buildings array.**
5. **Never cross buildings more than *MBP*.**
6. **Can start bouncing from any index of buildings array.**

Given the array of heights of buildings and the maximum bouncing power, find the maximum number of buildings the ball can visit.

Input/Output

Input	Output	Comments
20 15 50 20 32 45 33 30 36 20 40 2	4	● First line corresponds to the heights of ***buildings[ ]*** ● Second line **2** corresponds to the Maximum Bouncing Power Explanation: ● The ball can start at 40 (index 10). ● It can bounce in the following order **40(10) --> 36(8) --> 33(6) --> 30(7)** as shown in picture below. ● Note that if it starts at 33(index 6) it can only bounce to 30(index 7). ● It cannot bounce to index 5 because 45 > 33. ● It cannot bounce to 32 (index 4) because 45(index 5) is between 32(index 4) and 33(index 6) and 45 > 33.

		• Similarly it cannot bounce from 20(index 3) to 50(index 2) or 15(index 1). 
1 2 3 4 5 6 1	6	• Start at 6(index 5). It can visit all the buildings.
		•