TASK	JABUKE	MATRIX	X3	PLES	SORT	SKAKAC
source code	`jabuke.pas` `jabuke.c` `jabuke.cpp`	`matrix.pas` `matrix.c` `matrix.cpp`	`x3.pas` `x3.c` `x3.cpp`	`ples.pas` `ples.c` `ples.cpp`	`sort.pas` `sort.c` `sort.cpp`	`skakac.pas` `skakac.c` `skakac.cpp`
input	standard input (*stdin*)					
output	standard output (*stdout*)					
time limit	1 second	1 second	1 second	1 second	1 second	1.5 second
memory limit	32 MB	32 MB	32 MB	32 MB	32 MB	64 MB
point value	**50**	**80**	**110**	**110**	**140**	**160**
	650					

Mirko has recently discovered an old video game. The screen of this game is divided into **N** columns. At the bottom of the screen, there is an **M**-columns-wide boat (**M** < **N**). The player can move this boat left or right during the game, but the boat must remain completely within the screen at all times. The boat initially occupies the **leftmost M** columns.

Apples are being dropped from the top of the screen. Each apple starts its fall at the top of one of the **N** columns, falling straight down until it reaches the bottom of the screen. The next apple starts its fall just after the current one has reached the bottom.

An apple is said to be picked up if the boat is placed so that it occupies the column down which the apple is falling when it reaches the bottom. The goal of the game is to pick up all of the apples, in a way that **minimizes the distance** that the boat must travel.

INPUT

The first line of input contains two space separated integers **N** and **M** ($1 \le \mathbf{M} < \mathbf{N} \le 10$).

The second line of input contains an integer **J** ($1 \le \mathbf{J} \le 20$), the number of falling apples.

The following **J** lines contain the column positions of those apples, in the order in which they will fall.

OUTPUT

The only line of output must contain the minimal distance that the boat must travel in order to pick up all the apples.

SAMPLE TESTS

input 5 1 3 1 5 3 output 6	input 5 2 3 1 5 3 output 4

As we all know, we live inside the **matrix** that is divided into **N** rows and **N** columns. An integer is written into each one of the **N**x**N** cells of the matrix. In order to leave the matrix, we must find the **most beautiful square** (square-shaped sub-matrix) contained in the matrix.

If we denote by **A** the sum of all integers on the main diagonal of some square, and by **B** the sum of the other diagonal, then **the beauty** of that square is **A** - **B**.

Note: The main diagonal of a square is the diagonal that runs from the top left corner to the bottom right corner.

INPUT

The first line of input contains the positive integer **N** (2 ≤ **N** ≤ 400), the size of the matrix.

The following **N** lines each contain **N** integers in the range [-1000, 1000], the elements of the matrix.

OUTPUT

The only line of output must contain the maximum beauty of a square found in the matrix.

SAMPLE TESTS

input 2 1 -2 4 5 output 4	input 3 1 2 3 4 5 6 7 8 9 output 0	input 3 -3 4 5 7 9 -2 1 0 -6 output 5

Mirko has recently been visited by **extraterrestrials** from planet **X3**, where everyone's name is a positive integer. **All residents** of the planet **know each other**. Two X3-ians calculate the strength of their friendship by converting their names to binary, aligning them one under the other, and writing a digit in each column: 0 if the two binary digits in that column are equal, 1 if they differ. The binary result is then converted back to the decimal system.

For example, the friendship value of 19 and 10 equals 25:

$$\begin{array}{ccc} 1\,0\,0\,1\,1 & = & 19 \\ \underline{0\,1\,0\,1\,0} & \underline{=} & \underline{10} \\ 1\,1\,0\,0\,1 & = & 25 \end{array}$$

The **value of a planet** in the Universe is defined as the sum of all friendship values. Mirko has asked you to help him compute the value of planet X3!

INPUT

The first line of input contains the positive integer **N** (the number of residents of planet X3, 1 ≤ **N** ≤ 1 000 000). The next **N** lines contain the names of residents - positive integers smaller than 1 000 000, one per line.

OUTPUT

The only line of output must contain the value of planet X3.

SAMPLE TESTS

input	input	input

```
2
19
10
```
	```
	3
	7
	3
	5
	```
		```
		5
		9
		13
		1
		9
		6
		```

output	output	output
25	12	84

Second sample description: The friendship value of residents 1 and 2 equals 4, for residents 1 and 3 it equals 2, and for residents 2 and 3 it equals 6. The solution is 4 + 2 + 6 = 12.

There are **N** boys and **N** girls at a dance party. We have measured their heights. Each boy will only dance with a girl and each girl will only dance with a boy. Everyone will dance with at most one partner.

Each boy either wants to dance with a girl who is taller than him or with a girl who is shorter than him. Analogously, each girl either wants to dance with a boy who is taller than her or with a boy who is shorter than her. Boys and girls who are equally tall never want to dance with each other.

Respecting everyone's wishes, determine the maximum number of dancing pairs that can be achieved.

INPUT

The first line of input contains the positive integer **N** (1 ≤ **N** ≤ 100 000).

The second line of input contains **N** integers whose absolute values are between 1500 and 2500, inclusive. Their absolute values represent the height of each of the boys in millimetres. Positive height values indicate boys who want to dance with girls taller than themselves, while negative height values indicate boys who want to dance with girls shorter than themselves.

The third line of input contains **N** integers whose absolute values are between 1500 and 2500, inclusive. Their absolute values represent the height of each of the girls in millimetres. Positive height values indicate girls who want to dance with boys taller than themselves, while negative height values indicate girls who want to dance with boys shorter than themselves.

OUTPUT

The only line of output must contain the maximum number of dancing pairs.

SAMPLE TESTS

input 1 -1800 1800 output 0	input 1 1700 -1800 output 1	input 2 -1800 -2200 1900 1700 output 2

Consider the following sorting algorithm:

```
reverse-sort(sequence a)
    while (a is not in nondecreasing order)
        partition a into the minimum number of slopes
        for every slope with length greater than one
            reverse(slope)
```

A slope is defined as a decreasing consecutive subsequence of **a**. The reverse procedure will reverse the order of the elements in a slope.

You are given a permutation of the first **N** natural numbers whose slopes all have even length when partitioned for the first time. Determine the total number of times reverse is called to reverse-sort the given permutation.

INPUT

The first line of input contains the positive integer **N** (2 ≤ **N** ≤ 100 000).
The second line of input contains a permutation of the first **N** natural numbers that needs to be sorted.

OUTPUT

The only line of output must contain the number of times that reverse is called.

SAMPLE TESTS

input 2 2 1 output 1	input 4 4 3 2 1 output 1	input 4 3 1 4 2 output 3

Mirko and Slavko are playing the popular new game known as The Knight. Mirko places a knight chess piece on an **N**x**N** chessboard and, with Slavko blindfolded, makes **exactly T** moves, one per second. After that, Slavko must guess the final position of the knight in order to win.

The chessboard in this game is unusual in that each square is blocked part of the time. More precisely, each square is labelled by a positive integer. A square labelled by number **K** is **clear** only during seconds 0, **K**, 2**K**, 3**K** etc; it is **blocked** at all other times. The knight can, of course, occupy a square only while the square is clear.

The game begins in second 0. In each second Mirko **must make** a move (selecting one of 8 possible L-shaped moves, two squares in one direction and one square in the other, as per standard chess rules), provided that the destination square is **not blocked during the next second**. Help Slavko by writing a program to calculate all possible squares that the knight can possibly occupy after **T** moves.

INPUT

The first line of input contains two positive integers, **N** ($3 \le N \le 30$), the size of the chessboard, and **T** ($1 \le T \le 1\,000\,000$), the number of moves that Mirko will make.

The second line of input contains two positive integers **X** and **Y** ($1 \le X, Y \le N$), the row and column indices of the knight's starting square selected by Mirko.

The next **N** lines each contain **N** positive integers less than 10^9 (one billion), the values of **K** for the corresponding chessboard squares.

OUTPUT

The first line of output must contain the nonnegative integer **M**, the number of squares that the knight can possibly occupy after **T** moves.

The next **M** lines must contain indices of those squares, sorted by increasing row index, with cells in the same row sorted by increasing column index.

SCORING

In test cases worth **40%** of total points, the number of moves **T** will be less than 50 000.

SAMPLE TESTS

input 3 2 1 1 1 3 2 2 3 2 3 1 1 output 2 1 1 1 3	input 5 6 2 3 4 5 3 2 3 1 3 4 3 1 3 4 1 3 2 4 4 2 1 3 4 6 4 9 2 output 5 1 4 2 1 2 5 4 5 5 2	input 3 3 2 2 3 6 4 2 2 5 1 3 7 output 0

First sample description: The state of the chessboard in each second is shown below. Clear cells are denoted by '.', blocked cells by '#', and possible locations of the knight by 'K'.

0th second	1st second	2nd second

`K..`
`...`
`...`
	`.##`
	`###`
	`#K.`
		`K#K`
		`.#.`
		`#..`