The Stone House

Arben has inherited a beautiful plot of land in the hills overlooking Gjirokastër. Inspired by the city's famous stone-roofed houses, he plans to build a traditional house of his own.

The land is represented as an $N \times N$ square grid, where the rows and columns are numbered from 1 to N. The rocky terrain has a peculiar feature: there is exactly one large, immovable boulder in each row and each column of the grid.

To lay the foundation for his house, Arben must select an **empty square subgrid** — a region of size $S \times S$ that does not contain any boulders.

Let M be the maximum integer such that, for *all* arrangements of boulders satisfying the conditions above, an empty square subgrid of size $M \times M$ is guaranteed to exist. Arben wants to find *any* empty square subgrid of size at least $M \times M$.

To survey the land, Arben asks his friend Besa for help. Besa has brought a special surveying device called the **Gurëmatës** ("stone counter").

In a single operation, Besa can configure the Gurëmatës to scan a rectangular subgrid given by its top-left corner (x_1, y_1) and its bottom-right corner (x_2, y_2). The device returns the exact number of boulders in that region. The Gurëmatës is powered by a small battery and every scan consumes a lot of energy, so Besa must use it as few times as possible.

Write a program that uses the Gurëmatës to locate an empty square subgrid of size $S \times S$ with $S \geq M$, and reports its location.

Implementation details

You should implement the following procedure:

```
std::vector < int > solve(int N);
```

- N: the size of the grid.
- The procedure must return exactly four integers $\{x_1, y_1, x_2, y_2\}$: the top-left and bottom-right corners of the empty square subgrid it found. The condition $x_2 - x_1 = y_2 - y_1 = S - 1$ with $S \geq M$ must hold.
- This procedure is called exactly once for each test case.

The procedure can make calls to the following procedure:

```
int ask(int x1, int y1, int x2, int y2);
```

- x_1, y_1: the row and column of the top-left corner of the scanned subgrid.
- x_2, y_2: the row and column of the bottom-right corner of the scanned subgrid.
- $1 \le x_1 \le x_2 \le N$ and $1 \le y_1 \le y_2 \le N$ must hold. Otherwise, your program is terminated immediately and receives 0 points for that test case.
- The procedure returns the number of boulders in the scanned subgrid.

The grader is not adaptive: the positions of the boulders are fixed before `solve` is called and do not depend on your calls to `ask`.

Constraints

- $2 \le N \le 500$
- You can make at most $2.5 \cdot 10^5$ calls to `ask`.

Subtasks

Subtask	Points	Additional constraints
1	15	$N \le 40$
2	25	$N \le 150$
3	60	No additional constraints.

Scoring

If `solve` returns an invalid region, makes an invalid call to `ask`, or calls `ask` more than $2.5 \cdot 10^5$ times, you receive 0 points for that test case.

Otherwise, let Q be the number of calls to `ask`. For that test case you receive the subtask's points multiplied by $\min(1, f(Q))$, where

$$f(Q) = \begin{cases} \max(0, -0.05 \cdot Q + 1.55), & \text{if } Q \le 15 \\ \max(0, -0.19 \cdot \log_{10}(Q) + 1.02), & \text{if } Q > 15 \end{cases}$$

Your score for a subtask is the minimum score over all test cases in that subtask. The final score is the sum of the scores of the subtasks.

To help you estimate your score, the table below shows the approximate multiplier for some specific values of Q:

Number of queries (Q)	Multiplier ($\approx$)
$Q \leq 11$	1.00
12	0.95
15	0.80
50	0.70
100	0.64
1000	0.45
10000	0.26
100000	0.07
$Q \geq 233573$	0.00

Sample grader

The sample grader reads the input in the following format:

```
N
c_1 c_2 ... c_N
```

Here, c_i is the column of the boulder in row i ($1 \leq c_i \leq N$).

The sample grader prints a single line containing the multiplier $\min(1, f(Q))$ earned by your queries, or 0 if the returned subgrid or any call to `ask` is invalid.

Examples

Example 1

Sample grader input:

```
4
1 2 3 4
```

The boulders are at $(1,1)$, $(2,2)$, $(3,3)$ and $(4,4)$. Your program does not know this.

The grader calls `solve(4)`. One possible sequence of calls is:

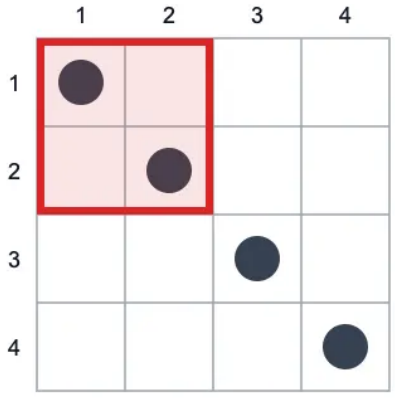
- `ask(1, 1, 2, 2)` returns `2`.
- `ask(2, 4, 4, 4)` returns `1`.
- `ask(3, 1, 3, 2)` returns `0`.

Then `solve` returns `{1, 3, 2, 4}`.

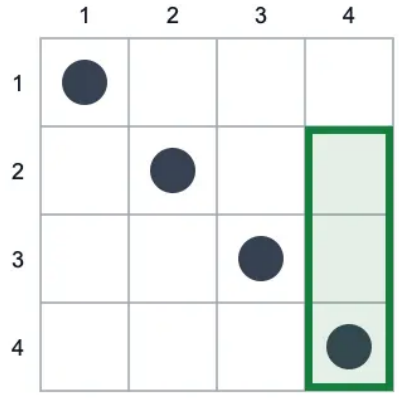
Sample grader output:

```
1
```

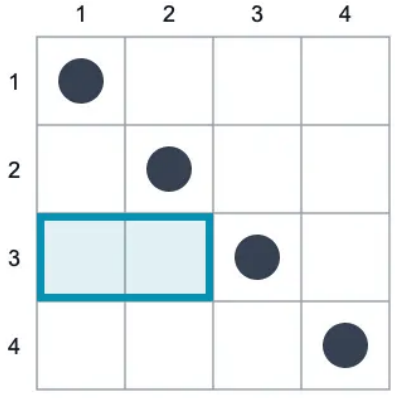
The first call scans the top-left 2×2 block and finds the boulders at $(1,1)$ and $(2,2)$. The second scans rows 2 to 4 of column 4 and finds the boulder at $(4,4)$. The third scans the first two cells of row 3 and finds no boulders. The returned region covers rows 1 to 2 and columns 3 to 4, which is an empty 2×2 subgrid.



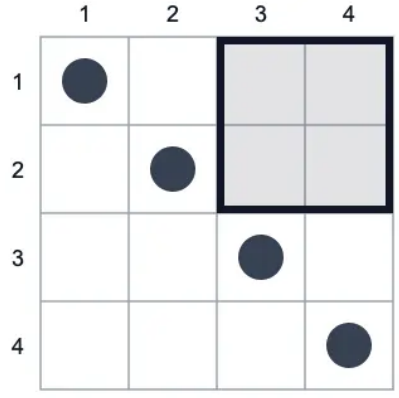
ask(1, 1, 2, 2) returns 2



ask(2, 4, 4, 4) returns 1



ask(3, 1, 3, 2) returns 0



return {1, 3, 2, 4}

For $N = 4$, the program used $Q = 3$ calls, so $f(Q) = 1.4$ and the multiplier is 1. Other answers are also accepted, for example `{3, 1, 4, 2}`, `{1, 2, 1, 2}`, `{3, 2, 3, 2}`, `{3, 4, 3, 4}` or any empty cell since $M = 1$ for $N = 4$.