Problem 1: Asphalting Roads

(Taken from: http://codeforces.com/problemset/problem/583/A)

City X consists of n vertical and n horizontal infinite roads, forming $n \times n$ intersections. Roads (both vertical and horizontal) are numbered from 1 to n, and the intersections are indicated by the numbers of the roads that form them.

Sand roads have long been recognized as out of date, so the decision was made to asphalt them. To do this, a team of workers was hired and a schedule of work was made, according to which the intersections should be asphalted.

Road repairs are planned for n^2 days. On the i-th day of the schedule the team arrives at the i-th intersection in the list. If **neither** of the two roads that form the intersection were already asphalted they asphalt both roads. Otherwise, the team leaves the intersection, without doing anything with the roads.

According to the schedule, give the days in which at least one road will be asphalted.

Input
The first line contains the integer n ($1 \leq n \leq 50$) - the number of vertical and horizontal roads in the city.

The next n^2 lines contain the order of intersections in the schedule. The i-th of them contains two numbers h_i, v_i ($1 \leq h_i, v_i \leq n$), separated by a space. This means that the i-th intersection in the schedule is the intersection of the h_i-th horizontal and v_i-th vertical roads. It is guaranteed that all of the intersections in the schedule are distinct.

Output
On a single line print the numbers of the days when road work will be done in ascending order, separated by spaces. The days are numbered starting from 1.

Examples

- **Input**
  2
  1 1
  1 2
  2 1
  2 2

  Output
  1 4

- **Input**
  1
  1 1

  Output
  1

Solution

We keep two arrays of length n to store which horizontal and vertical roads have already been paved. We initialize both of these lists to be all false.

Next, we iterate through the schedule of intersections. If both the horizontal and vertical roads have not been paved, we pave them, update our arrays, and add the day to our output. Otherwise, we do nothing and go to the next day.

The complexity of this solution is $O(n^2)$. This is linear in the length of the input, so it is the best possible.

Problem 2: Robot's Task

(Taken from: http://codeforces.com/problemset/problem/583/B)

Robot Doc is located in the hall, where n computers stand in a line,e numbered from left to right from 1 to n. Each computer contains **exactly one** piece of information, each of which Doc wants to get eventually. The computers are equipped with a security system, so to crack the i-th of them, the robot needs to collect at least a_i pieces of information from the other computers. Doc can hack the computer only if he is right next to it.

The robot is assembled using modern technologies and can move along the line of computers in either of the two possible directions, but the change of direction requires a large amount of resources from Doc. Give the minimum number of changes of direction that the robot will have to make to collect all n parts of information if initially it is next to the computer numbered 1.

It is guaranteed that there exists at least one sequence of the robot's actions that leads to the collection of all information. Initially Doc doesn't have any pieces of information.

Input
The first line contains the number n ($1 \leq n \leq 1000$). The second line contains n non-negative integers $a_1, a_2, \ldots, a_n$ ($0 \leq a_i < n$), separated by a space.

Output
Print a single number - the minimum number of changes of direction that the robot will have to make in order to collect all n pieces of information.

Examples

- **Input**
  3
  0 2 0

  Output
  1

- **Input**
  5
  4 2 3 0 1

  Output
  3

- **Input**
  7
  0 3 1 0 5 2 6

  Output
  2

Solution

We observe that we can assume that the robot goes all the way to the end of our row of computers each time before turning around. So we scan back and forth along the array. Each time we pass an available piece of information that has not yet been picked up, we take it.

We will pick up at least one new piece of information after each pass, so we need $O(n)$ passes across the list. The total complexity of this solution is $O(n^2)$. A faster algorithm may exist, but for $n <= 1000$, this will be sufficiently fast.

Problem 3: H-Index

(Taken from: https://leetcode.com/problems/h-index/)

Given an array of citations (each citation is a non-negative integer) of a researcher, write a function to computer the researcher's h-index. A scientist has index h if h of his/her N papers have **at least** h citations each, and the other $N - h$ papers have **no more than** h citations each.

For example, given the array $\textit{citations} = [3, 0, 6, 1, 5]$, we know that the researcher has five papers in total, which have received $3, 0, 6, 1$, and 5 citations, respectively. Since the researcher has three papers with **at least** three citations each and the remaining two with **no more than** three citations each, his h-index is 3.

Note: If there are several possible values for h, the maximum one is taken as the h-index.

Solution

The simplest solution is to first sort the array from papers with the least citations to papers with the most. Then we scan the array, at each step testing whether we can include all of the papers to the right of our current location as part of the h-index. Specifically, we test whether $n - i \leq \textit{citations}[i]$. If so, we return i.

However, we can modify this algorithm to take linear time as follows. We use a bucket sort - that is, we make a new array in which we count how many papers have i citations, for each i. However, we lump anything with at least n citations into the same bucket, because they will all be included in counting the h-index, regardless of the actual amount.

Then we use a similar scan as we did in the earlier solution, at each step checking whether we can include all of the papers to the right of our current location in the h-index.

So this algorithm only scans our list twice - one scan to perform the bucket sort, and a scan of the rearranged list to compute the h-index.