1 My Two Cents

1.1 Problem

A and B are playing a collaborative game that involves n stacks of coins, numbered from 1 to n. Every round of the game, they select a nonempty stack each, but they are not allowed to choose the same stack. They then remove a coin from both the two selected stacks and then the next round begins. The players win the game if they manage to remove all the coins. Is it possible for them to win the game, and if it is, how should they play?

1.2 Input

The first line of input contains an integer n ($2 \leq n \leq 50$), the number of coin stacks. Then follows a line containing n nonnegative integers $a_1, a_2, \ldots, a_n$, where a_i is the number of coins in the i'th stack. The total number of coins is at most 1 000.

1.3 Output

If the players can win the game, output a line containing "`yes`", followed by a description of the moves. Otherwise output a line containing "`no`". When describing the moves, output one move per line, each move being described by two distinct integers a and b (between 1 and n) indicating that the players remove a coin from stacks a and b. If there are several possible solutions, output any one of them.

1.4 Sample Data

Input	Output
3 1 4 3	yes 1 2 2 3 2 3 2 3
3 1 1 1	no

2 Global Raining of Bididibus

2.1 Problem

Bididibus is an ancient two-dimensional universe populated by the Bididibusians. The Bididibusians have their own 2D water, 2D mountains, 1D television, and even amazing new 2D cinemas!

These days, Bididibusian scholars warn the public about the risks of global raining, a climate catastrophe that would fill all the valleys of the universe with water.

Bididibus consists of blocks with uniform width and height. Bididibusian orography is described by a series of symbols, from left to right, indicating that the land rises one unit (represented by "/"), descends one unit (represented by "\"), or that it is flat (represented by "_").

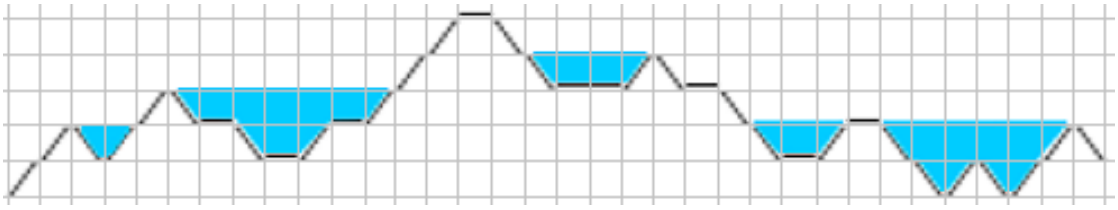
For example, we can have a 2D universe like this:

```
          _
         / \
        /   \__/\_
    /\_   _/      \   _
 /\/   \_/          \_/ \    /\
/                        \/\/
```

It goes without saying that the native geographers are big fans of ASCII art. They also use a simplified representation, where each column (from left to right) is replaced by the symbol it contains. For example, the simplified representation of the previous universe would be:

```
//\//\_\_/_///_\\__/\_\\_/_\\/\//\
```

After the global rain, water fills all the blocks of all valleys. If we represent water areas in blue, we would have the following:



Observe that a block with "/" or "\" partially filled with water corresponds to 1/2 units of water. A block completely filled with water corresponds to 1 unit of water. For example, in the previous universe we have a total of 21 units of water. Also note that the

border regions do not suffer from global raining since they are not completely enclosed by mountains.

Given a simplified representation of the universe, your task is to compute the units of water that we would have after the global rain.

2.2 Input

The first line of the input contains an integer T ($1 \leq T \leq 1000$), the number of test cases.

Each test case consists of a string composed from three possible characters: "/", "\" or "_", followed by a newline character. There will be at least one and at most 10000 characters in each input string.

2.3 Output

For each test case, the output should contain a single line with an integer indicating the amount of water in the universe in units after the raining.

2.4 Sample Data

Input	Output
5 //\//__/_///___/__/_\\/\//\ ////\\\\\\\\\ \/ \\\\\\\\\\///////// _____/////_/_/___	21 0 1 100 4

3 Guessing Data Structures

3.1 Problem

Last week Bob learned some new data structures and as an eager student implemented one of them. Can you guess which? All of the data structures have the following methods in common:

insert: Takes a value x ($0 \leq x < 100$) and inserts it into the data structure.
empty: Returns `"yes"` if the data structure does not hold any value, or `"no"` otherwise.
remove: Returns (and removes) a value from the data structure; the actual behaviour depends on the data structure Bob selected. This method call is only permitted if the data structure is not empty!

As a reminder, these are the data structures he learned about and how he named them (he sometimes is a bit lazy and prefers to use abbreviations):

queue: A data structure which works by the First-In-First-Out (FIFO) principle.
stack: A data structure which works by the Last-In-First-Out (LIFO) principle.
set: An ordered set of values without duplicates. In the context of this problem, removing an element means removing the smallest element.
pq: A data structure which allows quick access to the smallest value, i.e. removing an element means removing the smallest element.

It is guaranteed that Bob implemented one of the above data structures. Bob is a good programmer, so there are never any bugs in his code.

3.2 Interaction

This is an interactive problem. This means that your program will not first read some intput and then write some output, but instead communicates with the jury by writing queries to standard output and reading the answers from standard input.

You can perform up to 32 operations to determine what Bob implemented. To do this just print `"?␣<operation>␣<parameters>"` to `stdout`. Each of those operations has to be one of the three described above. If the operation does return a value, Bob will print it to `stdin`. If you make any invalid request Bob will print `-1` and will go away. In this case, your program should immediately exit in order to make sure that your submission receives the correct verdict.

As this problem is interactive, you need to flush `stdout` after each operation, because otherwise Bob won't be able to read your request.

Java: `System.out.flush()`
FastIO: `FastIO.out.flush()`
C++: `cout << endl;`

3.3 Output

Print "! <abbreviation>" after you figured out which data structure Bob implemented. Your program should terminate after printing the result. If you have questions, first look at the sample interaction (the spacing is only for clarification. Neither you or Bob should print empty lines).

3.4 Sample Data

Input	Output
42 66 1	? insert 42 ? insert 66 ? insert 1 ? remove ? remove ? remove ! queue

4 Boxes

4.1 Problem

Bob has applied for a job at a company that ships a large number of (one-dimensional) empty boxes. To check whether Bob is qualified for the job, he was asked to minimize the number of boxes that take up space.

To accomplish his task, he is allowed to put a smaller box inside a larger one, but only if the size of the larger box is at least twice as large as the size of the smaller box. In addition, at most one box is allowed to be placed directly inside another box (the smaller box can still contain other boxes, however). The goal is to minimize the number of visible boxes at the end.

Bob has been stuck with the task for a while. Since he has heard rumours about your extraordinary programming skills, he asked you to help him out.

4.2 Input

The input starts with a line containing the number of available boxes n ($1 \leq n \leq 5 \cdot 10^5$).

The next n lines each contain the size of a single box (positive). The $(i+1)$-th line contains the size of the i-th box. It is guaranteed that all sizes will fit into a signed 32-bit integer value.

4.3 Output

Print the minimum possible number of visible boxes on a single line (don't forget the linebreak at the end).

4.4 Sample Data

Input	Output
5 1 2 3 3 5	3

5 Fences

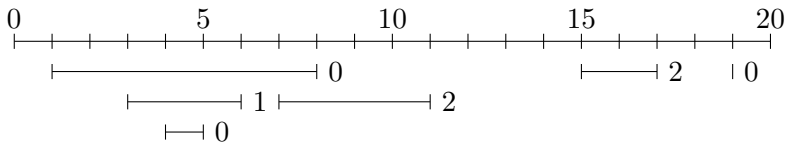
5.1 Problem

Alice is famous in Byteland for her creative fence paintings. Even the King, Giga Byte II, has heard of her paintings. Since the king has the biggest (height 1m, width 10^{12}m) and most precious fence of them all, he has hired Alice to paint it for him.

The King gave Alice the start and end position of a segment of the fence, along with a color to paint it in. However, since he didn't like the painted result, he gave Alice another such paint order, and another one, ... (he is not a very decisive king). Till, one day he shouted "This is it! It's a true masterpiece now!".

Giga Byte now wants to find out how much area of the fence is occupied by each color, so he can remember what mixture satisfies him. Note that a color that is placed on top of another color will cover the underlying color completely. Since Alice is too exhausted from all this painting, she wonders if you could help her determine the areas.

The following picture illustrates the sample input.



5.2 Input

The first line of the input contains two integers n ($1 \leq n \leq 10^5$), the number of available colors and m ($1 \leq m \leq 10^5$), the number of paint orders.

The next m lines each describe a paint order using three integers: the start position l and end position r in meters ($1 \leq l \leq r \leq 10^{12}$) as well as a color c ($0 \leq c < n$). Such an order means that Alice is supposed to paint the closed interval $[l, r]$ using color c.

5.3 Output

The output consists of n lines, containing the integers $a_0, a_1, \ldots, a_{n-1}$, where a_i is the area of fence that is occupied by color i after Alice has executed all the paint orders.

5.4 Sample Data

Input	Output
3 6 1 8 0 3 6 1 4 5 0 7 11 2 15 17 2 19 19 0	4 2 6

6 Unique Snowflakes

6.1 Problem

Emily the entrepreneur has a cool business idea: packaging and selling snowflakes. She has devised a machine that captures snowflakes as they fall, and serializes them into a stream of snowflakes that flow, one by one, into a package. Once the package is full, it is closed and shipped to be sold.

The marketing motto for the company is "bags of uniqueness". To live up to this motto, every snowflake in a package must be different from the others. Unfortunately, this is easier said than done because in reality many of the snowflakes flowing through the machine are identical. Emily would like to know the size of the largest possible package of unique snowflakes that can be created.

The machine can start filling the package at any time, but once it starts, all snowflakes flowing from the machine must go into the package until it is completed and sealed. The package can be completed and sealed before all of the snowflakes have flowed out of the machine.

6.2 Input

The first line of input contains one integer T specifying the number of test cases to follow ($1 \leq T \leq 10$). Each test case consists of a line containing an integer n ($1 \leq n \leq 10^5$), the number of snowflakes processed by the machine, followed by the n snowflakes (integers in the range from 0 to 10^9, inclusive). All the integers in each line are separated by spaces. Two snowflakes are identical iff they are represented by the same integer.

6.3 Output

For each test case, output a line containing a single integer, the maximum number of unique snowflakes that can be in a package.

6.4 Sample Data

Input	Output
2 5 1 2 3 2 1 2 1 1	3 1